TAMPEREEN
AMMATTIKORKEAKOULU

PISTEPILVIDATAN MERKKAAMINEN UNITYLLÄ VIRTUAALITODELLISUUDESSA

Samu Aronen

Opinnäytetyö
Joulukuu 2018
Tietojenkäsittely
Pelituotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Pelituotanto

ARONEN, SAMU:
Pistepilvidatan merkkaaminen Unityllä virtuaalitodellisuudessa

Opinnäytetyö 52 sivua
Joulukuu 2018

Opinnäytetyön tavoite oli selvittää, miten pistepilvidataa pystytään merkkaamaan Unity-pelimoottorissa virtuaalitodellisuudessa. Lisäksi osana tavoitetta selvitettiin alustavat toimenpiteet pistepilvien merkkaamisen mahdollistamiseksi. Näihin kuuluivat pistepilvidatan lukeminen Unityyn, pisteiden piirtäminen kuvaruudulle ja reaaliaikaisen suorituskyvyn ylläpito pisteitä näytettäessä. Opinnäytetyön tarkoitus oli toteuttaa se osa tietokoneohjelmasta, jolla pisteet merkataan. Käytännön työ toteutettiin opinnäytetyön toimeksiantajan Intopalo Digital Oy:n asiakkaalle Sandvik Mining and Construction Oy:lle. Tutkimusmenetelmänä työssä käytettiin kvalitatiivista tutkimusta ja lähestymistapana konstruktiivista tutkimusta.

Opinnäytetyön tuloksena luotiin yksittäisten pisteiden tarkkuudella toimiva merkkaustekniikka, joka hyödyntää säikeistystä reaaliaikaisen suorituskyvyn ylläpitämiseksi. Lisäksi alustavien toimenpiteiden selvityksen tuloksena saatiin kerättyä tietoa binäärimuotoisen pistepilvidatan lukemisesta ja näyttämisestä Unityssä sekä suorituskyvyn optimoinnista osittamalla pistepilviä tietorakenteen avulla.

Opinnäytetyön perusteella pystyttiin toteamaan, että Unity soveltuu hyvin pistepilvidatan merkkaamiseen. Valmiin pelimoottorin hyödyntäminen ohjelman kehityksessä nopeuttaa kehitysprosessia sisäänrakennettujen ominaisuuksiensa ansiosta. Toisaalta osa toteutuksessa käytetyistä Unityn tarjoamista rajapinnoista on suoritusnopeudeltaan hitaita, mikä tekee sujuvan suorituskyvyn saavuttamisesta välillä haastavaa. Työtä on mahdollista jatkokehittää selvittämällä muita pisteille tehtäviä toimenpiteitä, kuten pisteiden siirtämistä tai poistamista. Lisäksi on mahdollista, että Unityn kehittyessä sen uudemmat versiot mahdollistavat työn aihealueeseen soveltuvia vaihtoehtoisia lähestymistapoja, joita voisi verrata työssä esitettyihin ratkaisuihin.

Asiasanat: pistepilvi, unity, virtuaalitodellisuus, säikeistys

ABSTRACT

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

ARONEN, SAMU:
Marking Point Cloud Data in Virtual Reality with Unity

Bachelor's thesis 52 pages
December 2018

The aim of this study was to research the means of marking point cloud data in virtual reality by using the Unity game engine. Prerequisites, which were needed to enable point marking, were examined as well. These prerequisites include reading the point cloud data into Unity and displaying the points while maintaining sufficient performance. The purpose was to create the part of a computer programme responsible of the point marking process. The client for this thesis was Intopalo Digital Oy and the project was developed for their customer, Sandvik Mining and Construction Oy.

The thesis was conducted by using qualitative research approach. Constructive research methodology was used by collecting information from publicly available sources and by testing the derived solutions in practice.

As a result, a point marking technique with a precision to differentiate between single points was created. This solution made use of multithreading to maintain smooth real-time performance.

In conclusion, the findings indicate that Unity is well-suited for marking point cloud data. However, some of the used Unity programming interfaces are quite slow. It would be interesting to compare different approaches for achieving point marking in the future, due to Unity being constantly developed further. The solution could also be extended by adding more point processing features, such as moving or deleting points.

Key words: point cloud, unity, virtual reality, multithreading

SISÄLLYS

LYHENTEET JA TERMIT

.NET	Microsoftin kehittämä ohjelmistokehys, joka tukee useita ohjelmointikieliä
AABB	Axis Aligned Bounding Box, koordinaattiakselien suuntainen reunalaatikko
ASCII	American Standard Code for Information Interchange, 7-bittinen tietokoneiden merkistö
C#	Microsoftin kehittämä yleiskäyttöinen ohjelmointikieli
FPS	Frames Per Second -kuvataajuus, näytettyjen kuvien määrä sekunnissa
LOD	Level of Detail, objektin yksityiskohtien vähentäminen ja lisääminen sulavan kuvataajuuden ylläpitämiseksi
PCD	Point Cloud Data, Point Cloud Libraryn kehittämä pistepilvi tiedostoformaatti
PCL	Point Cloud Library
XR	Yhdistelmätermi, joka kattaa alleen lisätyn, sekoitetun ja virtuaalitodellisuuden

1 JOHDANTO

Unity-pelimoottoria hyödynnetään nykyään monissa digitaalisen tuotannon osa-alueissa perinteisen pelituotannon lisäksi. Unityn verkkosivuilla kuvaillaan sen tarjoavan ratkaisuja aloille, joissa käytetään reaaliaikaisia työkaluja. Näitä aloja ovat esimerkiksi elokuvat, autoteollisuus, arkkitehtuuri, mainonta ja XR. (Unity 2018j.) Tämä opinnäytetyö käsittelee aihetta, jossa Unityä käytetään pelituotannon sijasta toteutuksessa, joka on alueeltaan lähempänä XR:ää ja perinteistä ohjelmistotuotantoa.

Opinnäytetyön tavoitteena on selvittää, miten pistepilvidataa pystytään ohjelmistoteknisesti merkkaamaan Unity-pelimoottorissa virtuaalitodellisuudessa. Pisteiden merkkaamisella tarkoitetaan opinnäytetyössä menetelmää, jossa pelimoottorille ladatusta pistepilvidatasta erotetaan osa pisteistä. Tälle erotetulle osalle voidaan esimerkiksi antaa oma tunniste, tai valittu osa voidaan tallentaa kovalevylle. Opinnäytetyön tarkoitus on toteuttaa se osa tietokoneohjelmasta, jolla pistepilvipilvidataa virtuaalitodellisuudessa merkataan. Tutkimusmenetelmänä työssä käytettiin kvalitatiivista tutkimusta. Aihetta on lähestytty konstruktiivisella tutkimuksella, jossa opinnäytetyön pohjalla käytetty tieto kerättiin julkisista lähteistä ja kokeilemalla ratkaisuja käytännössä.

Opinnäytetyön toimeksiantaja Intopalo Digital Oy on Tampereella toimiva ohjelmistoalan yritys, joka tuottaa ratkaisuja monilla eri ohjelmistotuotannon osa-alueilla kuten XR, web ja tietoturvallisuus. Käytännön toteutus tehtiin asiakastyönä Sandvik Mining and Construction Oy:lle, joka osa maailmanlaajuista metalli- ja kaivosteollisuuden konsernia.

Osana tutkimusta on lisäksi selvitettävä ne alustavat toimenpiteet, jotka mahdollistavat pistepilvidatan merkkaamisen. Näitä toimenpiteitä ovat pistepilvidatan lukeminen Unityyn, pisteiden näyttäminen kuvaruudulla ja suorituskyvyn ylläpitäminen tarpeeksi nopeana, jotta toteutusta voidaan käyttää reaaliaikaisesti. Työssä keskitytään virtuaalitodellisuuden osalta sen ohjelmistotekniseen puoleen. Tämä tarkoittaa, että tarkastellaan esimerkiksi sellaisia asioita, kuten virtuaalitodellisuuden käyttöönoton vaikutuksia suorituskykyvaatimuksiin ja virtuaalitodellisuuden mahdollistamia rajapintoja ohjauksen kannalta.

2 UNITY VIRTUAALITODELLISUUDEN RAJAPINTANA

2.1 Unity-pelimoottori ohjelman ytimenä

Pelimoottorin tarkoitus on eristää yleiset pelin tuottamisessa käytettävät osat kuten pelin renderöinti, fysiikka ja ohjaus niin, että pelin kehittäjä voi keskittyä niihin yksityiskohtiin, joilla pelistä saadaan tehtyä ainutlaatuinen (Ward 2008). Pelimoottori on tarkoitettu pelin tuottamiseen, mutta pelimoottoreita on mahdollista käyttää muissakin tarkoituksissa, esimerkiksi sovelluksissa, joissa halutaan hyödyntää pelimäisiä reaaliaikaisia renderöintitekniikoita.

Reaaliaikaisella renderöinnillä tarkoitetaan sellaista sykliä, jossa tietokone näyttää kuvan ruudulla nopeasti, minkä jälkeen käyttäjä reagoi ruudulla näkyvään informaation vaikuttaen siihen, mitä seuraavaksi piirtyy kuvaruudulle. Tämä sykli tapahtuu niin nopeasti, ettei käyttäjä erota yksittäisiä kuvia. Reaaliaikaisella renderöinnillä tarkoitetaan yleensä kolmiulotteisen kuvan piirtämistä ruudulle käyttäen grafiikkakiihdytinlaitteistoa, poiketen perinteisistä kaksiulotteisista tietokoneohjelmista, jotka myös reagoivat käyttäjän komentoihin. (Akehine-Möller, Haines & Hoffman 2008, 1.)

Unity on kaupallinen Unity Technologiesin kehittämä ja ylläpitämä monen laitealustan pelimoottori. Sen ominaisuuksiin kuuluvat esimerkiksi reaaliaikainen renderöintimoottori ja tuki monen eri laitealustan grafiikkarajapinnalle, kuten DirectX, OpenGL ja Vulkan (Unity 2018b; Unity 2018m). Koska nämä ominaisuudet ovat jo valmiina Unityssä, on perusteltua käyttää sitä muuhunkin tarkoitukseen kuin pelituotantoon, jos näistä Unityn erityisominaisuuksista on hyötyä ohjelmiston kehityksen kannalta. Näin voidaan vähentää tarvittavaa ohjelmiston kehitysaikaa, kun pohjaominaisuudet ohjelmistolle ovat jo valmiiksi käyttövalmiina.

Reaaliaikaisen renderöinnin lisäksi Unity sisältää muitakin yleishyödyllisiä ominaisuuksia perinteisen ohjelmistojen tuottamisen kannalta. Unity tukee esimerkiksi ohjelmalogiikan kirjoittamiseen C# 6 -ohjelmointikieltä ja .NET 4.6 -luokkakirjastoja (Unity 2018k). Unityn toiminallisuutta pystytään myös laajentamaan liitännäisillä, jotka on toteutettu käyttäen .NET-kirjastoja tai kohdelaitteen natiiviohjelmointikieltä (Unity 2018i). Tämä

mahdollistaa sen, että monet C#-ohjelmointikielellä kirjoitetut liitännäiset pystytään lataamaan osaksi Unityä, vaikka niitä ei olisi suunniteltu käytettäväksi sen osana. Unity tukee myös XR:ää eli lisättyä, sekoitettua ja virtuaalitodellisuutta, jota hyödyntävät sekä suuret pelistudiot kuten Ubisoft, että IT-yritykset kuten Google (Unity 2018m). Pistepilven merkkaaminen virtuaalitodellisuudessa edellyttää monia näistä Unityn sisältäviä ominaisuuksista, minkä takia se vaikuttaa järkevältä alustalta ohjelman kehitykselle.

2.2 Virtuaalitodellisuus tietokoneohjelmassa

PC Magazine -tietokonelehden sanaston mukaan virtuaalitodellisuus määritellään tietokoneen luomana todellisuutena, joka projektoi käyttäjän 3D-avaruuteen. Stereoskooppisten virtuaalitodellisuuslasien sekä käsiensä tai muun ohjainyksikön avulla käyttäjä voi liikennöidä virtuaalitodellisuusympäristössä. (PC Magazine n.d.) Virtuaalitodellisuuslasit sisältävät näytön, jonka tehtävä on näyttää virtuaalinen maailma käyttäjälle sekä linssit, jotka taivuttavat valoa muodostaen käyttäjälle suuremman näkökentän (Thompson 2018). Tuotetun ohjelman tarvitsee vain piirtää kuva ruudulle, jonka sitten linssit muuntavat käyttäjän silmiin sopivaksi lopputulokseksi.

Käytettäessä virtuaalitodellisuutta tukevaa kehitysalustaa, ohjelmoijan ei tarvitse keskittyä siihen, että lopullinen kuva on yhteensopiva virtuaalitodellisuuslasien kanssa vaan se muunnetaan kehittäjän puolesta. Srinivasiahn (2017) mukaan Unityn tehtävänä on luoda kaksi näkymää, yksi kummallekin silmälle. Unity tukee eri tapoja tämän lopullisen kuvan renderöimiseen, jotka vaikuttavat osaltaan ohjelman suorituskykyyn, ohjelman tuettuihin laitealustoihin sekä varjostimien toimivuuteen (Srinivasiah 2017; Unity 2018l). On järkevää valita sellainen renderöintitapa, joka parhaiten palvelee ohjelman vaatimuksia yhteensopivuuden tai suorituskyvyn suhteen. Koska tämän virtuaalitodellisuusrenderöintitavan voi vaihtaa tarvittaessa Unity asetuksista, tätä valintaa ei tarvitse päättää kehityksen alussa, vaan sen voi tehdä myöhemmin, kun lopulliset tietokoneohjelman vaatimukset on selvitetty.

Virtuaalitodellisuuden renderöintitapa ei ole ainoa asia, joka vaikuttaa virtuaalitodellisuusohjelman suorituskykyyn. Tietokoneen kanssa käytettävillä virtuaalitodellisuuslaseilla resoluutio yksittäiselle silmälle vaihtelee alkaen 1080x1200 pikselistä aina

1440x1600 pikseliin asti (Greenwald 2018). Tämän perusteella tietokoneella virtuaalitodellisuutta käytettäessä kokonaisresoluutio olisi pienimmillään 2160x2400 pikseliä. Verrattuna perinteisiin näyttöihin esimerkiksi Steam-pelialustan käyttäjistä suurin osa käytetyistä näytöistä on resoluutioltaan 1920x1080 pikseliä (Steam 2018). Tästä voidaan päätellä, että tietokoneen tarvitsee prosessoida enemmän pikseleitä käytettäessä virtuaalitodellisuutta, verrattuna ohjelmaan, jota ajetaan 1920x1080-resoluutioisella näytöllä.

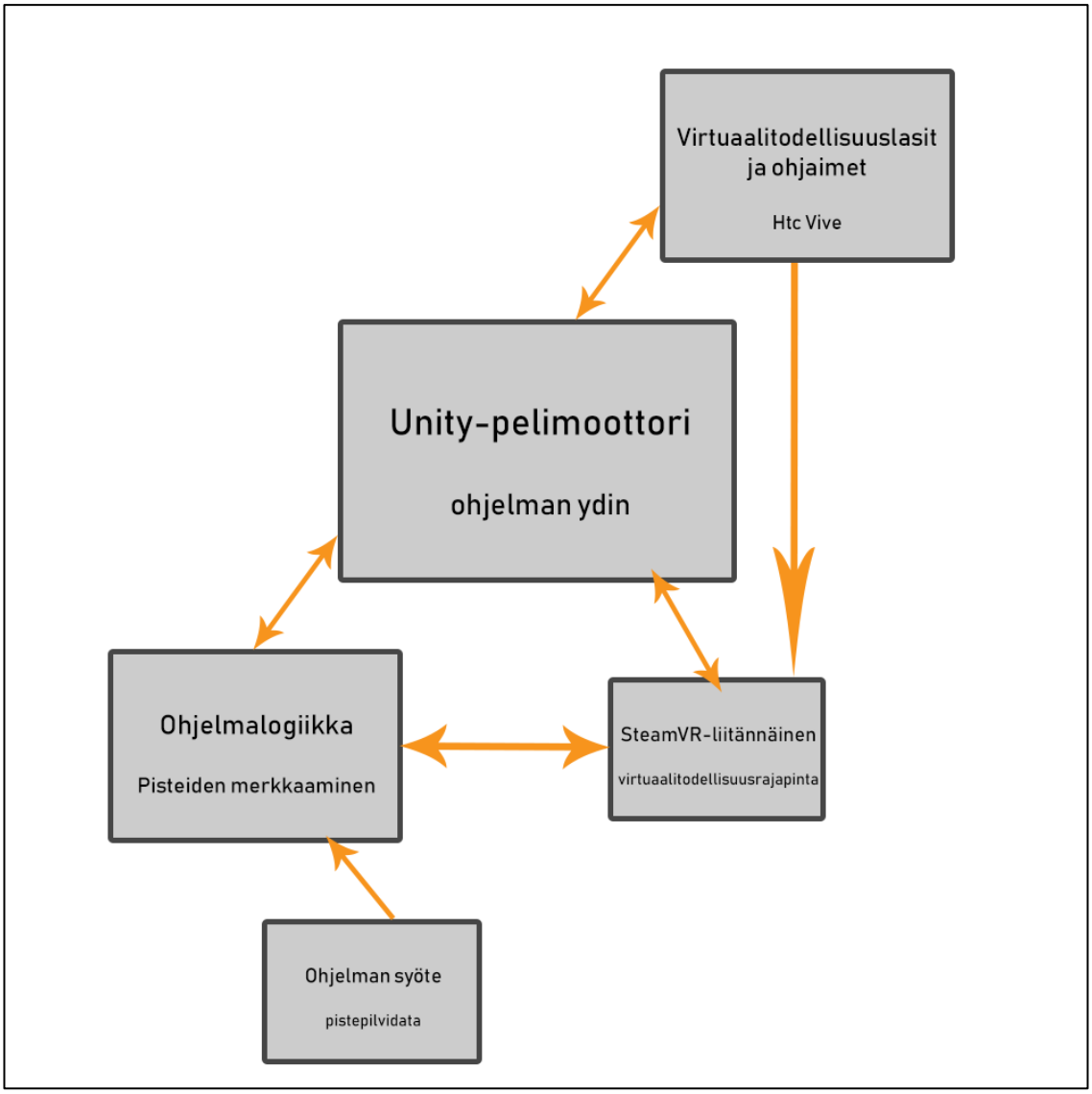
Virtuaalitodellisuudessa pahoinvoinnin välttämiseksi ihanteellinen viive näyttää liike näytöllä on alle 20 millisekuntia, minkä vuoksi laseissa on käytössä nopealla virkistystaajuudella olevat näytöt (McCaffrey 2017, 208). Virkistystaajuudella tarkoitetaan sitä nopeutta hertseinä, jolla kuva näytöllä virkistetään. Suurimassa osassa grafiikkajärjestelmiä tämä tapahtuu laitetasolla ja tämän vuoksi laitteiden virkistystaajuudet ovat vakioita. (Luekbe ym. 2003.) Myytävänä olevissa tietokoneeseen liitettävissä virtuaalitodellisuuslaseissa virkistystaajuus on 90 hertsiä (Greenwald 2018). Tämän perusteella pelimoottorin pitäisi lähettää laseille kuva 90 kertaa sekunnissa, jotta lasien virkistystaajuus saataisiin hyödynnettyä täysin. Verrattuna yleisiin 60 hertsin monitoreihin, kuvia pitäisi siis lähettää kolmasosan verran enemmän, mikä tarkoittaa, että ohjelma vaatii virkistystaajuudesta johtuen enemmän suoritustehoa kuin perinteinen reaaliaikainen ohjelma.

2.3 SteamVR Unityssä

SteamVR-liitännäinen on Valven kehittämä laajennus Unityyn, joka tarjoaa kehittäjille merkittävien virtuaalitodellisuuslasimallien kanssa yhteensopivan yksittäisen ohjelmointirajapinnan. Lisäksi se mahdollistaa ohjelmoijalle pääsyn Unitystä liiketunnisteisten ohjainten hallintaan ja tarjoaa virtuaaliset 3D-mallit tunnistamastaan laitteistosta. (Valve Corporation 2018). Käytännössä tämän lisäosan asentaminen Unityyn tarkoittaa sitä, että ohjelmiston käyttäessä SteamVR:n tarjoamaa rajapintaa sama ohjelma toimii muidenkin valmistajien laitteiden kanssa ilman erillistä ohjelmointityötä jokaiselle virtuaalitodellisuuslasi- tai ohjainmallille.

Opinnäytetyön toteutuksen pohjana toimivat siis Unity ja sen SteamVR-liitännäinen, mikä mahdollistaa sen, että työssä pystytään keskittymään suoraan pisteiden merkkaamisen ohjelmistotekniseen toteutukseen ja virtuaalitodellisuusinteraktiot suoritetaan yhtei-

siä rajapintoja käyttäen. Laitteistona työn kehityksessä käytetään HTC Vive -virtuaalitodellisuuslaseja. Kuviossa 1 kuvataan yleisellä tasolla työn relaatiot projektikokonaisuuden eri osa-alueisiin.



KUVIO 1. Virtuaalitodellisuusohjelman osien relaatiot toisiinsa

3 PISTEPILVIDATA

3.1 Pistepilven määrittäminen

Pistepilvi on tietorakenne, jota käytetään esittämään ryhmää moniulotteisia pisteitä. Sen yleinen käyttötarkoitus on esittää kolmiulotteista tietoa. Pisteitä käsiteltäessä tässä yhteydessä tarkoitetaan skannatun pinnan geometrisiä X, Y ja Z koordinaatteja. Pistepilvi voi sisältää myös väri-informaatiota, jolloin pilvi määritellään neliulotteiseksi. Pistepilvidataa voidaan hankkia eri tavoin kuten esimerkiksi stereokameroilla, time-of-flight kameroilla, 3D-skannereilla tai keinotekoisesti tietokoneohjelmalla luomalla. (Point cloud library n.d. a.) Opinnäytetyön raportissa esitetty pistepilvidata on peräisin julkisista, tutkimuskäytössä vapaasti käytettävistä lähteistä. Riippuen skannauksen koosta, yksi pistepilvi voi sisältää hyvin suuria määriä pisteitä. Esimerkiksi skannaus osasta Technische Universität München -rakennuksen sisustaa sisältää 59,7 miljoonaa pistettä (Huitl ym. 2012).

Pistepilvidatan esittämiseen ja tallentamiseen on olemassa monia eri tiedostomuotoja. Useiden tiedostomuotojen tarkasteluun voidaan käyttää avoimen lähdekoodin ohjelmaa nimeltä CloudCompare (CloudCompare n.d.). Opinnäytetyössä keskitytään tarkastelemaan Point Cloud Libraryn Point Cloud Data -tiedostomuodon pistepilvidataa.

3.2 PCD -tiedostomuoto

Point Cloud Data eli PCD on Point Cloud Libraryssä käytettävä pistepilvidatatiedostomuoto (Point Cloud Library n.d. c). Puolestaan Point Cloud Library eli PCL on avoin ohjelmistokirjastoprojekti kaksi- ja kolmiulotteiselle kuva- sekä pistepilvidatan prosessoinnille. PCL on maksuton käyttää sekä tutkimukselliseen että kaupalliseen käyttöön. (Point Cloud Library n.d. a.) Koska tarkoitus on lukea PCD-muotoinen data Unityyn, muita Point Cloud Libraryn ominaisuuksia kuin sen tiedostomuotoa ei työn tapauksessa käytetä.

PCD tiedosto koostuu kahdesta osasta: ASCII muotoisesta otsakkeesta ja varsinaisesta pistepilvidatasta. PCD-tiedoston otsake sisältää versiossa 0.7 seuraavat kentät:

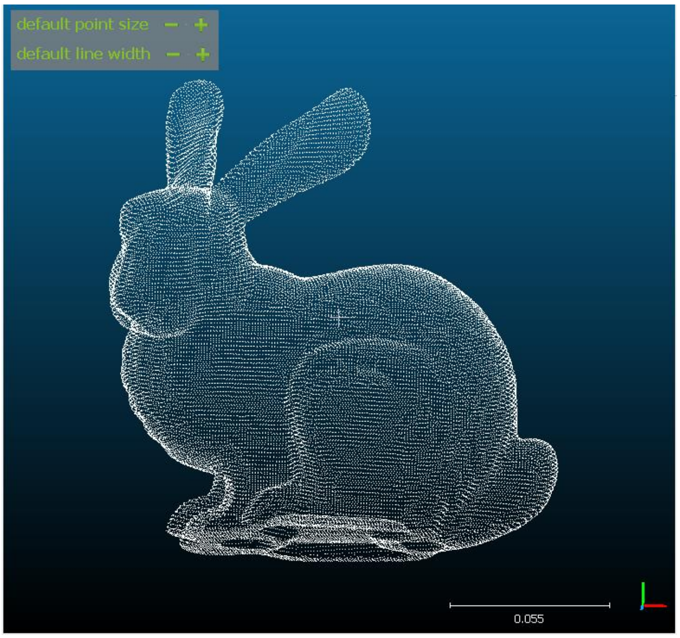
- VERSION – PCD-tiedoston versio
- FIELDS – pisteen ulottuvuuskenttien nimet
- SIZE – pisteen ulottuvuuden suuruus tavuissa eriteltynä jokaiselle ulottuvuudelle
- TYPE – pisteen ulottuvuuden tietotyyppi eriteltynä jokaiselle ulottuvuudelle
- COUNT – yhden ulottuvuuden alielementtien lukumäärä ulottuvuuskohtaisesti
- WIDTH – kokonaismäärä pisteitä yhdessä rivissä järjestetyssä pistepilvessä
- HEIGHT – kokonaismäärä pisterivejä järjestetyssä pistepilvessä
- VIEWPOINT – katselukulma pistepilven suhteen siirtymänä ja kiertona
- POINTS – pisteiden kokonaislukumäärä
- DATA – määrittää onko pistedata tallennettu binääri- vai ASCII-muodossa (Point Cloud Library n.d. c.)

Tarkemmin määriteltynä TYPE-kentän tietotyyppi kertoo, onko kyseessä joko etumerkitön kokonaisluku, etumerkillinen kokonaisluku vai liukuluku. Pistepilvessä, jossa pisteitä ei ole järjestetty on WIDTH-kenttä sama kuin kokonaispistemäärä ja HEIGHT-kenttä on tällöin vakiona yksi. Lisäksi otsakkeen kentät esiintyvät aina tässä luettelon määrittelemässä järjestyksessä. (Point Cloud Library n.d c.)

3.3 Pistepilven lukeminen Unityyn

Tämä kappale käsittelee pistepilvidatan lukemista Unityyn ilman kolmannen osapuolen pistepilvikirjastoja käyttäen toteutukseen vain Unityn ja .NET-luokkakirjaston ominaisuuksia. Tämä riippumattomuus näihin kirjastoihin pitää projektirakenteen siistinä käyttämättömästä koodista ja pitää projektin koon pienempänä. Lisäksi, kun pisteiden luku on ohjelmoitu juuri Unitylle, voidaan lataamisen aikana siirtää pisteet suoraan tiedostosta Unityn tietorakenteisiin.

Pisteiden lukemisen havainnollistamiseksi referenssipistepilvidatana käytettiin tässä raportissa Turkin ja Levoyn (1994) Stanford Bunny -mallia. Malli muutettiin PCD muotoon CloudCompare-ohjelmalla. Kuvassa 1 näytetään, millaiselta tämä pistepilvi näyttää CloudCompare-ohjelmassa.



KUVA 1. Turkin ja Levoyn (1994) Stanford Bunny avattuna CloudCompare-ohjelmassa ilman värejä ja pintoja

PCD-tiedoston voi avata tekstinkäsittelyohjelmassa, jonka avulla on mahdollista tarkastella tiedoston ASCII-muotoista otsaketta. Myös pisteiden ulottuvuuksien arvoja on mahdollista lukea, jos pistedata on tallennettu tekstimuotoon. Kuva 2 näyttää osan Stanford Bunny -PCD-tiedostoa avattuna tekstinkäsittelyohjelmassa. Kuvassa rivit yhdestä yhteentoista sisältävät tekstimuotoisen ASCII-otsakkeen, jonka jälkeen riviltä kaksitoista lähtien alkaa tekstinkäsittelyohjelmalle epäsopiva binäärimuotoinen pistepilvidata. Jotta pistepilvidata voidaan lukea Unityssä oikein, pitää otsakkeen tiedot saada myös välitettyä sille. PCD-tiedoston lukemiseksi tärkeät kentät otsakkeesta ovat FIELDS, SIZE, TYPE, COUNT, POINTS ja DATA. Jos tiedosto tai sen osa on tarkoitus myöhemmin tallentaa Unitystä takaisin kovalevylle, kannattaa tällöin lukea koko otsake Unitylle.

```
1   # .PCD v0.7 - Point Cloud Data file format
2   VERSION 0.7
3   FIELDS x y z _
4   SIZE 4 4 4 1
5   TYPE F F F U
6   COUNT 1 1 1 4
7   WIDTH 35947
8   HEIGHT 1
9   VIEWPOINT 0 0 0 1 0 0 0
10  POINTS 35947
11  DATA binary
12  Qó¼´STXETX>A ';NULNUL€?›j7½ôúETX>0°ù:NUL
```

KUVA 2. Muunnetun Stanford bunny -PCD-tiedoston alkuosa avattuna tekstinkäsittelyohjemassa (Pistepilvidata: Turk & Levoy 1994)

Microsoftin .NET-luokkakirjasto tarjoaa valmiita luokkia tiedostojen lukemiseen ja kirjoittamiseen. Näistä StreamReader-luokkaa käytetään lukemaan kirjoitusmerkkejä ja BinaryReader-luokkaa käytetään lukemaan binäärimuotoista tietoa. (Wenzel ym. 2017.) PCD-tiedostojen lukemiseen käytetään tekstimuotoisen otsakkeeseen StreamReader-luokkaa ja varsinaisen pistepilvidataan joko BinaryReader-luokkaa tai StreamReader-luokkaa riippuen siitä, onko se koodattu tiedostoon ASCII- vai binäärimuotoisena. Opinnäytetyössä keskitytään binäärimuotoisen pistepilvidatan lukemiseen, mutta ASCII-muotoisen tiedon lukemiseen toimivat samat funktiot, joilla otsake tulkitaan.

3.3.1 Otsakkeen lukeminen ja tulkitseminen

StreamReader luodaan antamalla sille parametrina luettava tekstitiedosto. Tämän jälkeen tiedostoa voidaan lukea yksi tekstirivi kerrallaan. Lukeminen voidaan lopettaa DATA-sanan sisältävän rivin jälkeen, koska PCD-tiedostomuodon otsakkeessa kentät ovat aina samassa järjestyksessä. Kuvassa 3 näytetään koodiesimerkki otsakkeen lukemisesta Unitylle rivi kerrallaan, niin että jokainen rivi otsakkeesta tulostetaan Unityn syötteeseen.

```
private void ParseHeader(string filePath)
{
    using (StreamReader reader = new StreamReader(filePath))
    {
        string currentLine = "";

        while (currentLine != null && !currentLine.Contains("DATA"))
        {
            currentLine = reader.ReadLine();
            Debug.Log(currentLine);
        }
    }
}
```

KUVA 3. Esimerkki PCD-tiedoston otsakkeen tulostavasta lähdekoodista, jossa ”filepath”-muuttuja sisältää tiedostopolun PCD-tiedostoon

Pelkällä otsakkeen kokonaisten rivien lukemisella ei saada välitetyksi tarpeellista tietoa binääridatan tulkitsemiseksi, vaan riveistä täytyy tulkata välimerkeillä erotetut arvot. Valmiita funktioita tekstin tarkasteluun ja pilkkomiseen on osana .NET-viitekehyksen String-

luokkaa (Microsoft n.d. f). On suositeltavaa, että otsaketta luettaessa koodissa varmistetaan, että rivi sisältää sen tiedon mitä sen oletetaan kenttien järjestyksen perusteella sisältävän. Jos rivi vaikuttaa oikeanlaiselta, sen arvot voidaan tulkata ja tallentaa myöhempää käyttöä varten. Otsakerivien tunnisteina toimivat nimet asetettiin vakiomerkkijonoiksi PcdFieldNames-luokkaan. Näin otsakkeen tulkitsemiseksi tarvittavat avainsanat pysyvät lähdekoodissa yhdessä paikassa, josta tunnisteita voi verrata luettavan tiedoston riveihin. Esimerkeissä otsakkeesta tulkitut arvot tallennettiin omaan HeaderInfo-nimiseen tietorakenneluokkaan, josta tarvittavat tiedot binääridatan lukemiseksi haetaan. Kuvan 4 esimerkissä näytetään pisteiden kokonaismäärän lukeminen kokonaislukumuuttujaan. Kun löytyy kohta, mikä täsmää POINTS-sanan kanssa, tekstirivi jaetaan kahtia ja pisteiden lukumäärä talletetaan ”headerInfo.TotalPoints” luokkaan myöhempää käyttöä varten.

```
currentLine = reader.ReadLine();

if (currentLine != null && currentLine.Contains(PcdFieldNames.Points))
{
    var splitLine = currentLine.Split(' ');

    headerInfo.TotalPoints = int.Parse(splitLine[1]);
}
```

KUVA 4. Esimerkki PCD-tiedoston otsakkeen kokonaispistemäärän tallettamisesta muuttujaan C#-ohjelmointikielessä

Vaikka monen otsakkeen kentän voi tallentaa samalla tavalla muuttujaan kuin esimerkissä käytetyn POINTS-arvon, täytyy erityistä huomiota kiinnittää kenttiin FIELDS, SIZE ja COUNT. Näiden arvojen pohjalta voidaan Unityssä kerätä binääridatasta varsinaisen pisteiden tiedot ja tallettaa ne muuttujiin. Käytännössä yhden ulottuvuuden tiedostokoko tavuissa saadaan kertomalla SIZE ja COUNT keskenään. Esimerkkitiedostossa tämän perusteella x-koordinaatin lopullinen koko olisi 4 tavua.

Tavallisten x, y ja z koordinaattien lisäksi FIELDS-kenttä voi sisältää muita lisäulottuvuuksia, kuten esimerkiksi pisteiden normaalit (Point cloud library n.d. c). Esimerkkitiedossa on yksi tällainen lisäkenttä, joka on merkattu alaviivalla. Tätä lisäkenttää ei ole suoraan määritelty ja se voi mahdollisesti olla PCD-muunnoksesta jäljelle jäänyt ylimääräinen kenttä. Vaikka tällaista kenttää ei tarvittaisi pistedatan koordinaattien siirtämiseen

Unityyn, on sen tavupituus silti huomioitava, jotta muu binääridata saadaan luettua oikein.

Kuvassa 5 näytetään esimerkki siitä, kuinka otsakkeesta voi laskea sijainnin pisteen ulottuvuuksille yhdessä ”binääririvissä”. Binääririvillä tarkoitetaan tässä tapauksessa sellaista osaa binääridatasta, joka sisältää kaikki pisteen ulottuvuudet yhden kerran. Esimerkissä käydään läpi jokainen FIELDS-kentän ulottuvuus, jolle lasketaan suuruus tavuissa SIZE- ja COUNT-kenttien kertolaskusta. Tämä lasketaan yhteen edellisten ulottuvuuksien tavukokojen kanssa, jolloin saadaan laskettua ulottuvuudelle paikka yhdessä binääririvissä. Lopuksi tämä sijainti annetaan Dictionary-tietorakenteelle, jonka avaimena toimii ulottuvuuden nimi. Tämän tietorakenteen sekä viimeiseksi yhteenlasketun koko ”binääririvin” pituuden eli ByteRowLength-muuttujan avulla voidaan lukea varsinainen binäärimuotoinen tieto Unityyn.

```
Dictionary<string, int> fieldOffsetDictionary = new Dictionary<string, int>();

int offset = 0;

for (int i = 0; i < fields.Length; i++)
{
    int byteLength = int.Parse(sizes[i]) * int.Parse(counts[i]);

    fieldOffsetDictionary.Add(fields[i], offset);

    offset += byteLength;
}

headerInfo.ByteRowLength = offset;
```

KUVA 5. PCD-tiedoston pisteen ulottuvuuksien binääripaikkojen laskeminen yhdelle binääririville

Otsakkeen tulkkia suunniteltaessa on järkevää selvittää, kuinka geneerinen sen tarvitsee toteutuksen kannalta olla. Lähdetiedostosta kannattaa kopioida Unitylle vain se informaatio, mikä on välttämätöntä pisteiden näyttämisen kannalta. Esimerkiksi monia kymmeniä miljoonia pisteitä sisältävästä pistepilvestä ei ole järkevää kopioida ohjelman muistiin ylimääräistä tietoa, jos sitä ei ole tarkoitusta näyttää käyttäjälle tai muuten hyödyntää pelimoottorissa.

3.3.2 Binääridatan siirtäminen Unityn muuttujille

Kun PCD-tiedoston otsakkeen tiedot on tarkistettu, voidaan pistepilvidata siirtää Unityyn BinaryReaderin avulla. Microsoftin (n.d. a) dokumentaation mukaan BinaryReader sisältää funktion ReadBytes, joka hyväksyy parametrina kokonaisluvun. Funktio lukee kokonaisluvun osoittaman määrän tavuja ja palauttaa lukemansa tavut sisältävän taulukon. Lisäksi palautettu taulukko on kooltaan pyydettyä pienempi, jos tavuja ei ollut tiedostossa jäljellä niin montaa kuin funktiolta pyydettiin. (Microsoft n.d. a.) Pistepilvitiedoston lukeminen voidaan siis lopettaa, kun tämä funktio palauttaa nollapituisen taulukon.

Ennen pisteiden arvojen siirtämistä on määriteltävä, mistä kohtaa tiedostoa pisteiden informaatio alkaa. Koska tekstimuotoinen otsake on osa samaa tiedostoa, täytyy binäärin lukeminen kelata kohtaan, missä tekstiosa päättyy. Koska PCD-tiedostomuodossa otsakkeen rivien järjestys on aina sama, voidaan tiedostosta lukea viimeisenä esiintyvä sana eli ”binary” ja siirtää lukija tämän jälkeisen rivivaihdon jälkeiseen tavuun. Kuvassa 6 on toteutus aloituspaikan haulle, jossa käytetään Queue-tietorakennetta ja tavuista tekstiksi muunnosta etsimään binääridatan aloituskohta. Koska otsake on jo tarkistettu aiemmassa vaiheessa, pitäisi tämän aliohjelman löytää aina aloituskohta normaalin ajon aikana. Jokin hakuraja on testauksen aikana kuitenkin järkevä olla, jotta Unity Editor ei jäädy testatessa loputtomaan silmukkaan ja pakota kehitysympäristön uudelleenkäynnistystä.

```
private void SeekStartingOffset(BinaryReader reader)
{
    int timeout = 10000;
    var byteQueue = new Queue<byte>();

    for (int i = 0; i < timeout; i++)
    {
        byte b = reader.ReadByte();

        byteQueue.Enqueue(b);

        if (byteQueue.Count == 6)
        {
            string s = System.Text.Encoding.ASCII.GetString(byteQueue.ToArray());

            if (s.ToLower() == "binary")
            {
                reader.ReadByte();
                return;
            }

            byteQueue.Dequeue();
        }
    }

    Debug.LogError("Timeout reached, offset search failed");
}
```

KUVA 6. PCD-tiedoston binääridatan alkamiskohdan hakeminen

Kun BinaryReader on siirretty oikeaan aloituskohtaan, voidaan binääridatan kopioiminen Unityn muuttujille aloittaa. Unity käyttää 3D-vektoreiden ja pisteiden esittämiseen Vector3-tietorakennetta (Unity 2018n). Tämän takia on luonnollista siirtää pistepilven pisteiden koordinaatit tähän Unityn omaan tietorakenteeseen. Microsoftin (n.d. c) .NET-luokkakirjaston BitConverter-luokan avulla voidaan muuntaa tavu-tietotyyppi toiseksi perustietotyypiksi, kuten liukuluvuksi.

Kuva 7 sisältää esimerkin, jossa ensimmäiseltä ”binääririviltä” haetaan koordinaattien liukuluvut ja näistä muodostetaan BitConverterin avulla Vector3 tietorakenteen piste. Kaikki pisteiden koordinaatit voidaan hakea lukemalla aina tiedostosta uusi ”binääririvi” ja tallentamalla tavuista sijainnit Vector3-tietorakenteeseen, kunnes lukija palauttaa tyhjän rivin. Jos tiedon otsakkeen TYPE-kenttä olisi määritellyt koordinaatit joksikin muuksi kuin liukuluvuiksi, pitäisi tavumuunnokseen käyttää toiselle tietotyypille vastaavaa muuntofunktiota. Tätä funktiota hyödyntämällä pystytään lukemaan myös muut pisteen ulottuvuudet, kuten värit tai muu lisätty skalaari-informaatio.

Kuvasta 7 voi myös huomata, että pisteen Z- ja Y-koordinaatit on luettu Unityyn päinvastoin verrattuna alkuperäisdataan. Tämä johtuu siitä, että Unityn koordinaattijärjestelmässä Y-akseli on saman suuntainen lähdetiedoston Z-akseli ja toisinpäin. Vaihtamalla ZY koordinaatit toisin päin, saadaan pistepilvi orientoitua niin, että pistepilven yläsuunta on myös Unityn yläsuunta.

```
SeekStartingOffset(binaryReader);

byte[] byteData = binaryReader.ReadBytes(headerInfo.RowByteLenght);

Vector3 point = new Vector3(
    BitConverter.ToSingle(byteData, headerInfo.FieldOffsetDictionary[FieldType.x]),
    BitConverter.ToSingle(byteData, headerInfo.FieldOffsetDictionary[FieldType.z]),
    BitConverter.ToSingle(byteData, headerInfo.FieldOffsetDictionary[FieldType.y])
);
```

KUVA 7. Esimerkki liukulukukoordinaattien siirtämisestä Unityn Vector3-tietorakenteeseen

Vaikka esimerkkitiedostossa ei käytetty pisteiden väriarvoja, on järkevää kuitenkin mahdollistaa niiden lukeminen. Point Cloud Libraryn (2018) dokumentaation mukaan histo-

riallisista syistä RGB-väriarvo pakataan yhteen integer-kokonaislukuarvoon, jonka muuttujatyyppi vaihdetaan lopuksi float-liukulukuarvoksi. Tämän perusteella voidaan päätellä, että jos luettava tiedosto sisältää otsakkeen mukaan väriarvona vain yhden liukuluku kentän on se silloin todennäköisesti pakattu tähän muotoon. Dokumentaatiossa on lisäksi annettu kaava C++-ohjelmointikielellä värien purkamiseksi tästä muodosta (Point Cloud Library 2018). Kuvassa 8 värien purkaminen kokonaisluvusta on muutettu C#-ohjelmointikielelle sopivaan muotoon ja värit talletettu Unityn väritietorakenteeseen. Uudemmissa Point Cloud Libraryn versiossa väriarvoja on mahdollista käyttää suoraan (Point Cloud Library 2018). Jos väriarvot on tallennettu suoraan tavuina, ne voidaan lukea binääririviltä ja tallettaa Unityn tietorakenteeseen ilman muuntamista.

```
byte r = (byte)((rgb >> 16) & 0x0000ff);
byte g = (byte)((rgb >> 8) & 0x0000ff);
byte b = (byte)(rgb & 0x0000ff);

color = new Color32(r, g, b, 0);
```

KUVA 8. Väritavujen tallennus Unityn rakenteeseen C#:lla, jossa rgb on kaikki värit sisältävä kokonaislukumuuttuja (Point Cloud Library 2018, muokattu)

4 PISTEPILVEN NÄYTTÄMINEN UNITYSSÄ

4.1 Pistepilven renderöiminen Unityssä

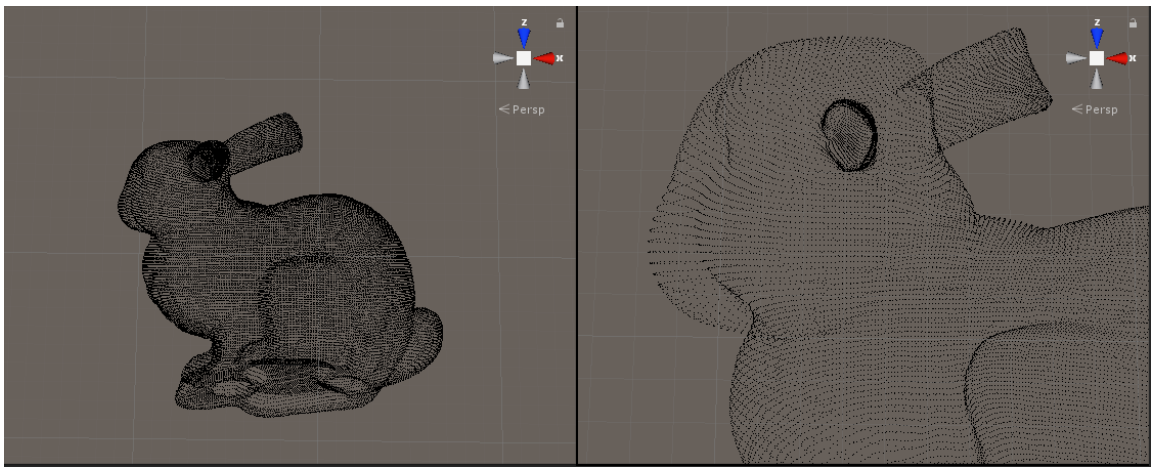
Pisteiden tallentaminen Unityn tietorakenteisiin ei vielä riitä pisteiden näyttämiseksi kuvaruudulla. Unityssä MeshRenderer-komponentti renderöi MeshFilter-komponentin asettamat 3D-verkot (Unity 2018g). Dimitrovin (2014) mukaan Unityssä voi renderöidä pistepilven tätä MeshRenderer-komponenttia käyttämällä. Hänen tekniikassaan jokaiselle pisteelle lasketaan juokseva kokonaislukuindeksi ja annetaan väri. Tämän jälkeen nämä tiedot voidaan siirtää MeshFilter-komponentissa olevalle Mesh-tietorakenteelle, jonka jälkeen Unity osaa piirtää tuloksen näytölle. Lisäksi hän huomauttaa, että yhden 3D-verkon rajana on 65 tuhatta pistettä. (Dimitrov 2014.) Nykyään tämän rajoituksen voi poistaa ja silloin Unity tukee suurempia verkkoja, suurimmillaan neljän miljardin pisteen kokoisia. Tämä voidaan sallia laittamalla Mesh-luokan indeksointimuoto 32 bittiseksi. (Unity 2018d.)

Kuvassa 9 on koodiesimerkki Vector3-muotoisten pisteiden siirtämisestä 3D-verkkoon. Esimerkissä toteutuksessa haetaan Vector3-muotoiset pisteet tietorakenteesta, johon on voitu tallentaa sijainnin lisäksi PCD-tiedoston lisäinformaatiota, kuten pisteiden värit tai intensiteettiarvot. Lopuksi 3d verkko merkitään dynaamiseksi MarkDynamic-metodilla, mikä takaa paremman suorituskyvyn ohjelmalle, jos verkkoa muutetaan useasti (Unity 2018e). Kolmiulotteista verkkoa päivitetään useasti pisteiden merkkaamisen yhteydessä, jonka takia ominaisuus laitetaan päälle.

```
private Mesh CreateMesh(Queue<PointItem> points)
{
    Mesh mesh = new Mesh();
    mesh.indexFormat = UnityEngine.Rendering.IndexFormat.UInt32;
    int count = points.Count;
    Vector3[] vertices = new Vector3[count];
    int[] indices = new int[count];
    Color[] colors = new Color[count];
    Vector2[] intensities = new Vector2[count];

    for (int i = 0; i < count; ++i)
    {
        var p = points.Dequeue();
        vertices[i] = p.Position;
        indices[i] = i;
        colors[i] =  p.Color;
        intensities[i] = new Vector2(p.Intensity,0);
    }

    mesh.vertices = vertices;
    mesh.colors = colors;
    mesh.uv2 = intensities;

    mesh.SetIndices(indices, MeshTopology.Points, 0);
    mesh.MarkDynamic();

    return mesh;
}
```

KUVA 9. Esimerkki 3D-verkkon luomisesta Vector3-pisteitä käyttämällä (Dimitrov 2014, muokattu)

Kun 3D-verkko on luotu, piirtyy pistepilvi Unityn näkymään. Koska PCD-lähdetiedostossa ei oltu esimerkissä määritelty värejä, latautuu pilvi oletusarvoisesti mustan värisenä. Ladatusta pistepilvestä voi huomata sen, että pisteet pysyvät aina yhden pikselin kokoisina riippumatta kameran etäisyydestä pisteisiin (Kuva 10). Tämä on epäkäytännöllistä virtuaalitodellisuuskäytön kannalta, koska pisteitä ei näe juuri lainkaan läheltä katsottuna. Varsinkin pisteiden merkkaus ohjaimia käyttämällä menee hankalaksi, jos pisteen tarkkaa sijaintia ei voi hahmottaa. Lisäksi pisteet näkyvät Unityssä oletusvarjostimella mustan värisinä riippumatta siitä, oliko 3D-verkkoon asetettu PCD-tiedostossa määritettyjä värejä vai ei.

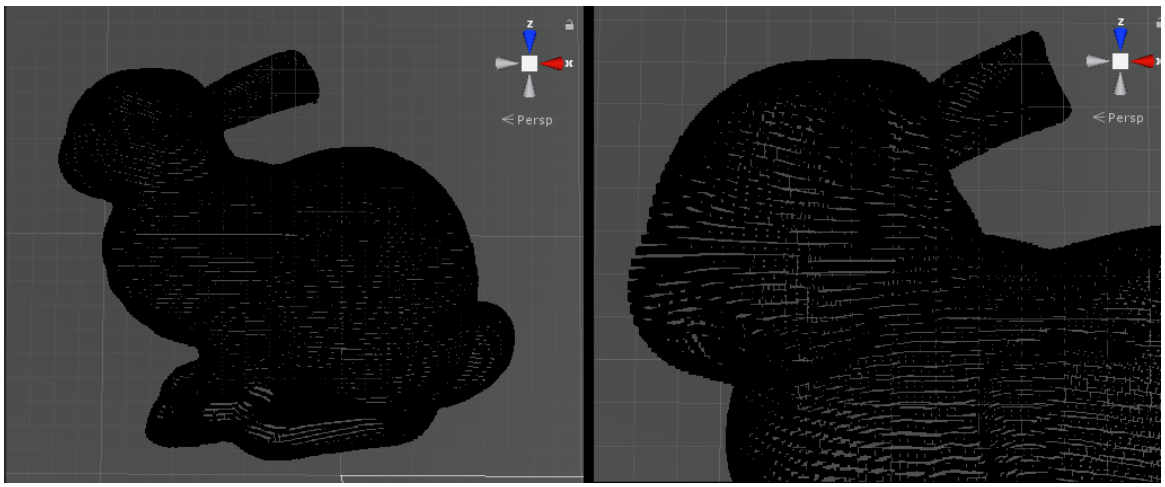


KUVA 10. Stanford kani pistepilvi piirrettynä Unityssä (Pistepilvidata: Turk & Levoy 1994)

4.2 Varjostimet pistepilviin

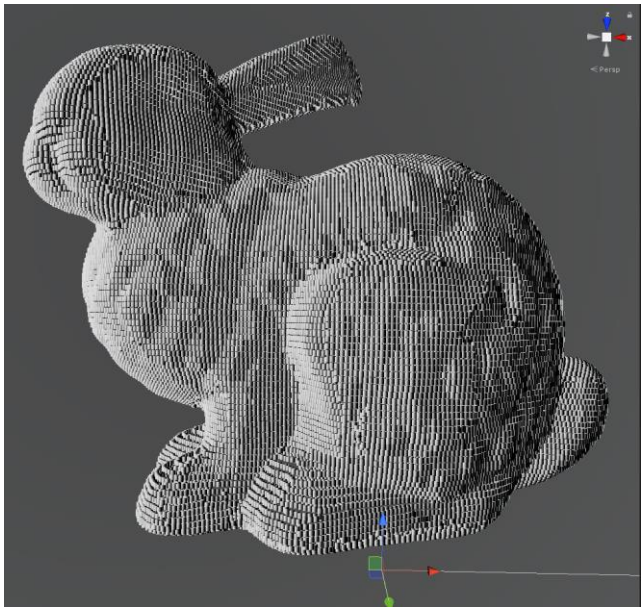
Jotta pisteistä saadaan virtuaalitodellisuusystävällisempiä, voidaan niiden ulkonäköön vaikuttaa varjostimilla. Varjostin on käyttäjän määrittelemä ohjelma, joka ajetaan grafiikkasuorittimella (Khronos 2015). Pelikehittäjien Alan Zucconin ja Kenneth Lammersin (2014, 95) mukaan termi shader eli varjostin sai alkuperänsä siitä, että sitä pääasiassa käytettiin simuloimaan realistista valaistusta ja varjoja 3D-malleissa. Nykyään niillä voi määritellä sekä objektien graafisen ulkonäön että niiden geometriset muodot (Zucconi & Lammers 2014, 95). Opinnäytetyössä varjostimia käytetään pistepilvien pisteiden koon suurentamiseen ja pisteiden värien säilyttämiseen. Pistepilvikäyttöön on olemassa erilaisia valmiita varjostimia, joita voi ostaa esimerkiksi Unityn Asset Store -kaupasta, mutta tässä raportissa keskitytään julkisesti saatavilla oleviin varjostimiin.

Pisteet voidaan näyttää 3D-avaruudessa värillisinä nelikulmioina geometriavarjostinta käyttämällä (smb02dunnal ym. 2018). Geometriavarjostin vastaanottaa yhden primitiivin, kuten pisteen, kolmion tai viivan ja voi palauttaa saman verran, enemmän tai vähemmän primitiivejä kuin annettaessa (Khronos 2018). Käytetty geometriavarjostin siis luo aina kameraan päin osoittavan nelikulmion jokaisen pisteen kohdalle. Lisäksi varjostin värjää nelikulmiot 3D-verkolle lähetettyjen pisteväriarvojen mukaisesti. (smb02dunnal ym. 2018). Kuvassa 11 esimerkkipistepilvi on piirretty geometriavarjostimella, jonka takia pisteet kasvavat perspektiivin mukaan oikein kameran siirtyessä niitä lähemmäksi.



KUVA 11. Pistepilvi geometriavarjostimen kanssa näytettynä (Varjostin: smb02dunnal ym. 2018; Pistepilvidata: Turk & Levoy 1994)

Kuvasta 11 käy ilmi, että pisteiden koon kasvattaminen tekee hankalammaksi hahmottaa muotoja yksivärisessä pistepilvessä. Käytettyyn neliögeometriavarjostimeen voi asettaa tekstuurin, joka näkyy jokaisessa piirretyssä neliössä (smb02dunnal ym. 2018). Näin mustaan pistepilveen saadaan värivaihtelua lieventämään tätä haittavaikutusta. Kuvassa 12 on käytetty mustavalkoista liukuväriä varjostimen tekstuurina havainnollistamaan tätä mahdollisuutta. Käytetyssä geometriavarjostimessa neliöiden tekstuurien värit kerrotaan 3D-verkon pisteiden väriarvoilla (smb02dunnal ym. 2018). Koska oletusvärin eli mustan RGBA-arvot ovat nollia, ei tekstuuri kertolaskun tuloksena näkyisi lainkaan. Tämän takia kuvassa 12 pisteiden värit on muutettu valkoisisi, jotta tekstuuri näkyisi.



KUVA 12. Mustavalkoinen liukuväri varjostimen tekstuurina (Varjostin: smb02dunnal ym. 2018; Pistepilvidata: Turk & Levoy 1994)

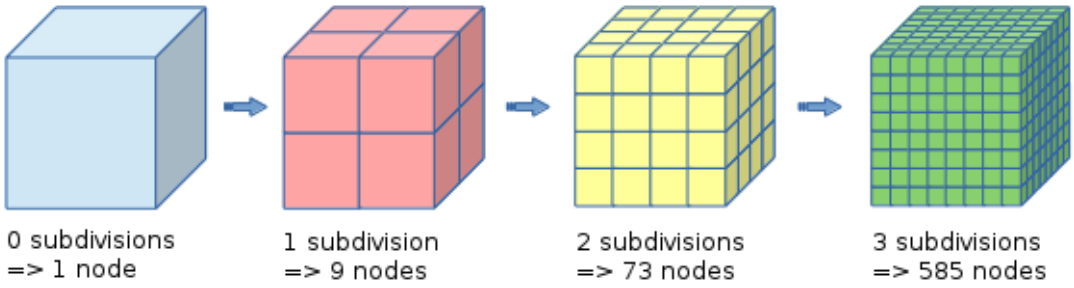
4.3 Pistepilven näyttämisen nopeuttaminen

Stanford bunnyä käytettiin esimerkkinä sen helpon hahmotettavuuden takia. Pienen pistemäärän takia se ei kuitenkaan anna todellista kuvaa sovelluksen reaaliaikaisesta suorituskyvystä. Tähän tarkoituksen käytetään 3d-skannausta osasta Technische Universität München -rakennuksen sisustaa, joka sisältää 59.7 miljoonaa pistettä (Huitl ym. 2012). Kohdelaitteina käytettiin suorituskykyisiä kuluttajaluokan pöytätietokoneita. Esimerkkikokoonpano sisältää Intel i7 4770k prosessorin, Nvidia RTX 2080Ti -näytönohjaimen ja 16 gigatavua keskusmuistia. Lataamalla tämän lähes 60 miljoonaa pistettä sisältävän pistepilven Unityyn saatiin reaaliaikaisesti renderöityä alle 30 kuvaa sekunnissa. Tämä johtuu siitä, että pisteitä näytetään liian paljon kerralla, jonka takia näytönohjain ei kykene piirtämään niitä sulavalla kuvien määrällä. Tällainen FPS on vielä liian vähän soveltuakseen virtuaalitodellisuuskäyttöön, minkä takia näytettävien pisteiden määrää on optimoitava.

Reaaliaikaisen 3D-grafiikan tekniikkaa, jossa monimutkaista objektia esitetään eri resoluutioilla ja joista sopivin esitystapa valitaan kompromissina kuvan tarkkuuden ja FPS:n väliltä, kutsutaan nimellä ”Level of Detail” eli LOD (Luekbe ym. 2003). Jotta pistepilven renderöinnin kuvataajuus voidaan ylläpitää tarpeeksi nopeana virtuaalitodellisuuskäyttöä varten, on järkevää käyttää tällaista tekniikkaa. Koska pistepilvidata koostuu vain irrallisista kolmiulotteisista pisteistä, voidaan näytettävien pisteiden määrää helposti vähentää jakamalla pisteet useammalle eri MeshRenderer-komponentin omaaville GameObject-peliobjekteille. Tällä tavalla pistepilven osia sisältäviä peliobjekteja voidaan helposti laittaa päälle ja pois Unityn C#-koodin sisältä. Tällaisen käyttäytymisen toteutukseksi tarvitaan taustalle algoritmi, jonka perusteella päätetään, mitkä objektit näytetään tai jätetään näyttämättä.

4.3.1 Octree algoritmina

Octree on yksinkertainen hierarkkinen tilanjakamisrakenne, jossa objektia ympäröivä laatikko jaetaan rekursiivisesti kahdeksaan yhtä suureen oktanttiin. Nämä oktantit voidaan puolestaan jakaa vielä pienemmiksi oktanteiksi. (Luekbe ym. 2003.) Kuviossa 2 havainnollistetaan, miltä octree-rakenne näyttää, kun sitä jaetaan pienempiin oktantteihin.



KUVIO 2. Octree rakenteen rekursiivinen jako (Kuvio: Geier 2014)

Octree-algoritmia käytetään usein yhdessä pistepilvien kanssa. Esimerkiksi tehokkuutensa takia CloudCompare-ohjelmassa sitä käytetään spatiaalisia laskentoja tekevissä funktiossa (CloudCompare 2015). Myös PCL sisältää octree-algoritmin ominaisuuksia pistepilven spatiaaliseen osittamiseen ja pisteiden hakemiseen (Point Cloud Library n.d b). Wimmer ja Scheiblauer (2006) käyttivät sisäkkäisiä octree-rakenteita kehittämässään LOD-algoritmissa suurien pistepilvien nopeaan renderöintiin. Opinnäytetyön toteutuksessa päädyttiin käyttämään octree-rakennetta renderöinnin nopeuttamiseen sen yleisyyden takia.

4.3.2 Octree Unityssä

Unitylle on olemassa valmiita toteutuksia octree-algoritmista eri käyttötarkoituksiin. Yksi sellaisista on Nitionin (2018) alun perin peliä varten kehittämä UnityOctree, jonka rakenteeseen voi tallettaa C#-objekteja ja niihin liittyvän reunus- tai sijainti-informaation. Rakenteesta voi hakea läheisiä rakenteeseen laitettuja objekteja antamalla hakukohdan sijainnin tai reunat (Nition 2018). Koska tämän toteutuksen julkinen rajapinta ottaa vastaan kokonaisia objekteja, olisi sen käyttäminen pistepilvikäytössä epäkäytännöllistä. Tämä johtuu siitä, että jos yhtä pistettä käsiteltäisiin yhtenä rakenteeseen talletettavana peliobjektina, muodostuisi suurien pistepilvien tapauksissa Unitylle useita miljoonia peliobjekteja käsiteltäväksi. Toisaalta, jos pisteet talletettaisiin tälle rakenteelle valmiiksi ositettuina kokonaisuuksina, olisi muun rakenteen pitänyt jo osittaa ne kertaalleen. Koska ositus on tarkoitus tehdä käyttämällä yhtä rakennetta, ei työn kannalta ollut järkevää hyödyntää tätä ratkaisua.

Toinen octree-rakennetta LOD:na hyödyntävä toteutus Unitylle on Simon Fraissin (2017) kehittämä suurten pistepilvien renderöintitekniikka. Hänen toteutuksessaan pistepilvet ladataan valmiista octree-rakenteen sisältävästä tiedostohierarkiasta Unityyn renderöintiä varten. Unityssä hän määrittelee kameradatan perusteella, mitkä LOD-tasot pistepilvestä näytetään. (Fraiss 2017). Koska pistepilvet on tarkoitus lukea ja osittaa PCD-muotoisesta järjestämättömästä tiedostosta laitteen muistiin, ei Fraissin Unity-toteutusta voida soveltaa sellaisenaan tässä tarkoituksessa erilaisen lähdetiedostorakenteen takia.

Pisteitä on tarkoitus näyttämisen lisäksi pystyä myös merkkaamaan ja kyseisessä toimenpiteessä pisteiden sijainnit on saatava tarkasteltavaksi. Tämän takia sen rakenteen toiminta, jonne pisteet talletetaan, on syytä ymmärtää täysin. Koska Unitylle tehdyt valmiit ratkaisut eivät suoraan soveltuneet PCD-muotoisen pistepilven osittamiseen, päädyttiin toteuttamaan pistepilvidatalle sopiva octree-rakenne, jonne pisteet voidaan sijoittaa suoraan PCD-tiedoston lukemisen yhteydessä.

Gobbetti ja Marton (2004) käyttivät pistepilviin LOD tekniikkaa, jossa pistepilvidata oli ryhmitetty ja yksityiskohtaisemmat tasot tarkensivat perusryhmän pistepilveä. Lisäksi Wimmer ja Scheiblauer (2006) käyttivät sisäkkäisessä octree-rakenteessaan samanhenkistä mallin tarkentamista. Fraissin (2017) suurten pistepilvien renderöimisessä Unityssä hyödynnettiin Markus Schuetzin (2016) kehittämää Potree-tiedostorakennetta, jossa myös pisteinformaatiota tarkennetaan lisäämällä pisteitä syvemmillä octree-tasoilla. Koska pistepilvidatan tarkentamista on käytetty aiemminkin LOD-tasojen luontiin, käytetään opinnäytetyössä samankaltaista tekniikkaa.

Opinnäytetyössä käytetyn tekniikan päämäärä on sisällyttää oktantteihin niiden alueelle sijoittuvat PCD-tiedostolta luetut pisteet. Pisteet on tarkoitus tallentaa rakenteeseen luomalla pistedataa, kuten koordinaatteja ja värejä, sisältävä objekti jokaiseen oktanttiin, jonka sisälle pisteitä sijoittuu. Suurin eli juurioktantti sisältäisi karkeasti pisteitä koko PCD-tiedostosta. Syvemmät oktantit puolestaan sisältävät lisää pisteitä omalta alueeltaan. Tällä tavoin pistepilvi tarkentuu, mitä enemmän syvyystasoja octree-rakenteesta näytetään kerralla samalla tavalla kuin Wimmerin ja Scheiblauerin (2006) tekniikassa, mutta käyttämällä vain yhtä octree-rakenetta. Opinnäytetyön octree toteutuksessa ei luoda tiedostohierarkiaa pisteille vaan octree-rakenne ja pisteet sisältävät 3D-verkot pidetään ajon aikana tietokoneen keskusmuistissa.

Toteutuksessa laskettiin aluksi annetun pistepilven suurimmat ja pienimmät koordinaatit, jotta voidaan päätellä tarvittavan octree-rakenteen koko ja keskikohta. Koska PCD-tiedosto ei sisällä otsakkeessaan näitä tietoja, on se esiluettava, jotta saadaan pistepilven suurimmat ja pienimmät koordinaattiarvot. Toimenpiteen nopeuttamiseksi esilukuvaiheessa ei ole kuitenkaan välttämätöntä lukea tiedostosta jokaista pistettä, mikäli rakenteen koon tarkkuus ei ole ehdotonta projektin kannalta. Rakenteesta mahdollisesti yli menevät pisteet voidaan määritellä kuuluvaksi octree-rakenteen juurioktantille, jonka takia täydellinen tarkkuus koon määrittelemisessä ei ole tarpeen toteutuksessa. Kun pistepilven suurimmat ja pienimmät koordinaatit on määritetty, voidaan rakeenteen keskikohta sijoittaa näiden kahden pisteen väliin ja juurioktantin koko laskea pisteiden etäisyydestä.

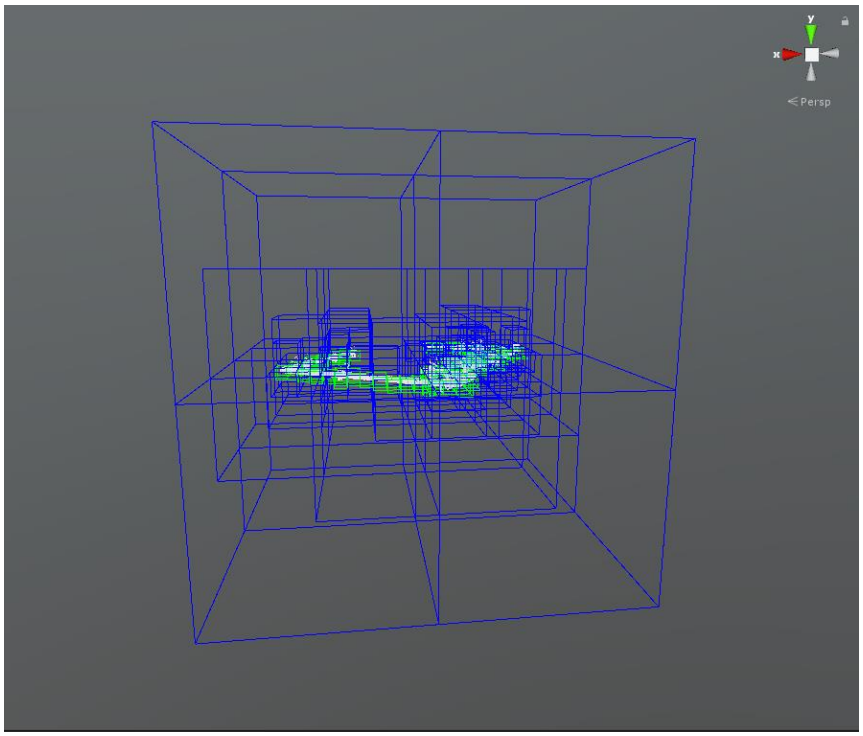
Rakenteen koon ja sijainnin lisäksi on myös määriteltävä, kuinka syvä rakenteesta tehdään. Octreen syvyydellä tarkoitetaan sitä, kuinka monta kertaa maksimissaan oktantit jaetaan uudestaan pienempiin oktantteihin. Wimmerin ja Scheiblauerin (2006) nopeassa tavassa renderöidä pistepilviä käytettiin käyttäjän määrittelemää maksimisyvyysarvoa sekä sisäisille että ulkoisille rakenteille. He suosittelivat, että syvyysarvon ei tulisi olla liian pieni, jotta heidän algoritmissaan käytettäviä sisäkkäisiä octree-rakenteita ei syntyisi liian montaa. Lisäksi syvyys ei saisi olla liian suuri, jotta kuvakartion ulkopuolelle jäävien osien näyttämättä jättäminen eli ”frustrum culling” toimisi tehokkaasti. (Wimmer & Scheiblauer 2006).

Vaikka opinnäytetyössä käytetään vain yhtä octree-rakennetta, vaikuttaa syvyyden valinta samankaltaisesti pisteiden näyttämisen suorituskykyyn. Mikäli syvyyttä kasvatetaan, tarvittavien pistedataobjektien määrä lisääntyy kahdeksankertaisella määrällä jokaisella tasolla verrattuna edelliseen. Jos taas syvyys pidetään liian pienenä, ei ositettuja pisteryhmiä ole tarpeeksi sujuvan FPS:n saavuttamiseksi. Tämän syvyyden voi jättää käyttäjän määriteltäväksi tai sille voidaan laskea sopiva arvo perustuen pisteiden tiheyteen koko octree rakenteessa.

4.3.3 Oktanttien näyttäminen sijainnin perusteella

Kun octree-rakenteen syvyys on määritetty, voidaan PCD-pisteet suodattaa rakenteeseen. Suodatus tapahtuu niin, että osa pisteistä jätetään ylemmille suuremmille oktanteille ja

loput sijoitetaan alemmille tarkemmille oktanteille. Suodatuksen jälkeen jokaisen oktantin pisteryhmäobjektista voidaan muodostaa Dimitrovin (2014) tavalla 3D-verkko ja asettaa se MeshRenderer-komponentin sisältävään peliobjektiin. Koska Unity (2018d) mahdollistaa 3D-verkon indeksointimuotoa vaihtamalla neljän miljardin pisteen kokoiset 3D-verkot, riittää toteutuksessa aina yksi 3D-verkko oktanttia kohden. Näitä voidaan laittaa helposti päälle ja pois tarvittaessa, esimerkiksi riippuen kameran sijainnista. Niille oktanteille, joille ei suodatu yhtään pistettä, ei suoritustehon ja muistin ylläpitämiseksi luoda objektia lainkaan. Pistepilvestä näkyy kauempana kamerasta epätarkempi versio, mutta lähestyttäessä mielivaltaista osiota pistepilvestä lisää peliobjekteja laitetaan näkyväksi ja pistepilvi tarkentuu. Kuvassa 13 näytetään pistepilvi Unityssä opinnäytetyön octree-rakenteeseen sijoitettuna, jossa siniset laatikot kuvastavat luotuja oktantteja ja vihreät tarkimman tason oktantteja.

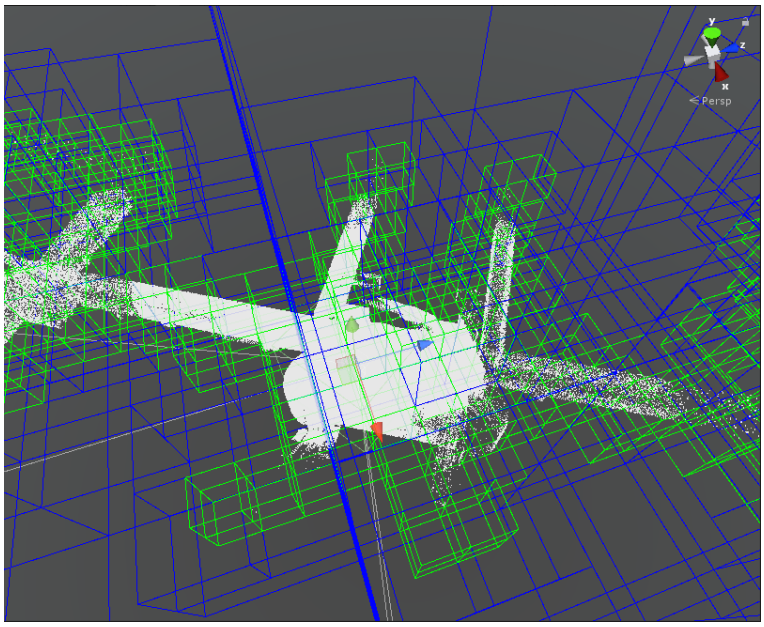


KUVA 13. Pistepilvi ladattuna octree-rakenteeseen Unityssä (Pistepilvidata: Huitl ym. 2012)

Pisteiden näyttämiseen octreestä kameran sijainnin perusteella käytettiin Unityn Bounds-luokkaa. Unityn (2018a) dokumentaation mukaan Bounds-luokka on esitys koordinaattiakselien suuntaisesta reunalaatikosta eli AABB:stä, joka ympäröi mielivaltaista objektia. Koska laatikkoa ei koskaan kierretä koordinaattiakselien suhteen, se voidaan määritellä pelkällä keskikohdalla ja laajuudella (Unity 2018a).

Toteutuksessa jokaiselle oktantille luotiin niiden kokoa vastaava Bounds-laatikko. Lisäksi kameran ympärille määriteltiin erikokoisia Bounds-alueita rakenteen leikkaustarkasteluja varten. Oktantit, joiden kanssa leikkaus tarkastetaan, haetaan rakenteesta rekursiivisesti vertaamalla niitä kameran reunalaatikoihin. Käytännössä tämä tehdään tarkastelemalla juuritasolta lähtien oktantteja, jotka leikkaavat kameran AABB:n kanssa ja tekemällä tarkastelut pienempiin oktantteihin, mikäli leikkaus tapahtui suuremmalla tasolla. Kameran reunalaatikkoa leikkaavien oktanttien sisältävät peliobjektit voidaan laittaa päälle, kun taas ne ryhmät, jotka eivät leikkaa voidaan laittaa pois päältä. Kameran ympärille luotiin kolme erikokoista AABB:tä, jotta octree-rakenteesta voidaan hakea karkeampaa informaatiota kauempana kamerasta ja tarkkaa informaatiota lähellä kameraa. Rekursiivisille hauille voidaan määrittää maksimisyvyys, jolloin leikkaustarkasteluja ei tarvitse suorittaa yhtä paljoa kameran kaukaisille laatikoille kuin läheisille.

Koska näkyvyystarkasteluissa käytetään AABB-laatikoita, ei tarkastettava alue muutu, vaikka kameraa käännetään ympäriinsä. Tämän ansiosta aktiiviset pisteet pysyvät samoina, mikäli käyttäjä vain katselee ympäriinsä virtuaalitodellisuudessa muuttamatta sijaintiaan. Nopea kameran kääntäminen voisi aiheuttaa suuren määrän pisteiden latautumista kerralla, mikäli tarkastusalue olisi rajattu vain kameran edessä olevalle alueelle. Pahoinvoinnin välttämiseksi koettiin, että on parempi pitää alue vakiosuuruisena kameran kaikilta puolilta. Kuva 14 havainnollistaa, kuinka pisteet näkyvät tiheämpänä kuvan keskellä, johon Unityn päänäkymän kamera on sijoitettu.

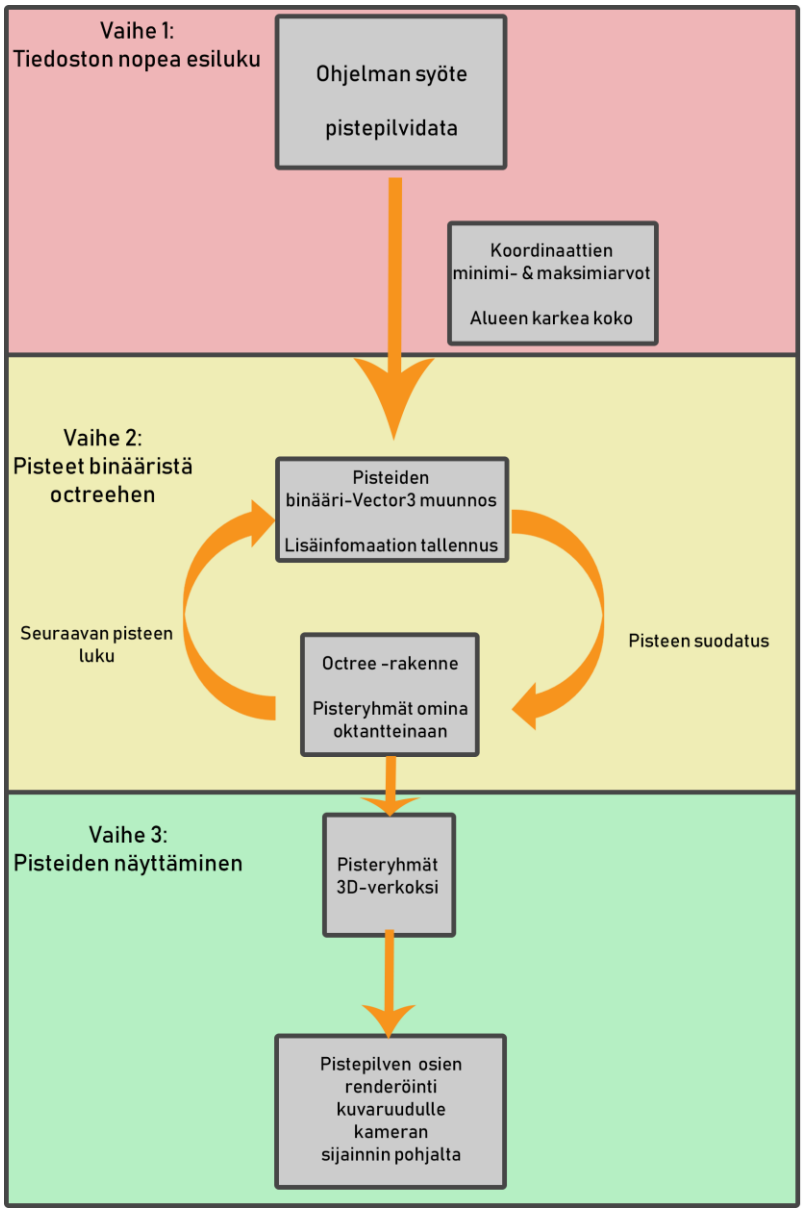


KUVA 14. Pistepilvi näkyy tarkempana nuolilla merkityn kameran kohdalta (Pistepilvidata: Huitl ym. 2012)

4.3.4 Octreen tulokset toteutuksessa

Ohjelman FPS parani huomattavasti, koska pistepilvestä ei näytetä enää jokaista pistettä, vaan vain ne, jotka on tarpeellista näyttää kameran sijainnin perusteella. Ottamalla käyttöön octree, saatiin pistepilvi näkymään virtuaalitodellisuuden edellyttämällä kuvataajuudella.

Kuvio 3 esittää kaaviona pistedatan etenemisen PCD-tiedostolta octree-rakenteen kautta ruudulle piirrettäväksi. Pisteiden lukeminen ja suodatus octree-rakenteeseen vaatii Unityltä kohtuullisesti latausaikaa, mutta edut sulavan kuvataajuuden suhteen ovat toteutuksen kannalta tärkeämmät kuin tiedostoon prosessointiin kulunut aika.



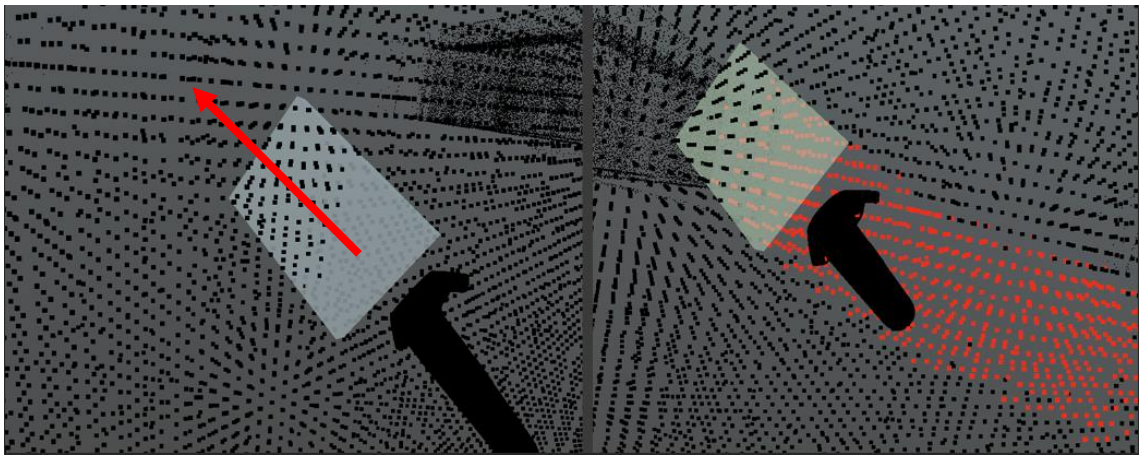
KUVIO 3. Pistepilvidatan kulkeminen tiedostosta ruudulle piirrettäväksi

5 PISTEPILVEN MERKKAAMINEN PISTETASOLLA

5.1 Pisteiden merkkaus virtuaalitodellisuudessa

Pistepilven piirtyminen kuvaruudulle sulavalla FPS:llä tarkoittaa, että alustavat toimenpiteet pisteiden merkkausta varten ovat valmiit. Merkkaamisella tarkoitettiin siis osan pistepilvidatan erottamista tästä ladatusta pistepilvidatasta. Verrattuna tavallisiin tietokonenäyttöihin virtuaalitodellisuus helpottaa pistepilvidatan hahmottamista sen linsseistä johtuvan paremman syvyysvaikutelman ansiosta. Lisäksi virtuaalitodellisuuslaitteistojen, kuten HTC Viven, ohjaimet voi orientoida 3D-avaruudessa vapaasti, minkä takia pisteiden valitseminen on tarkempaa kuin esimerkiksi kahdessa ulottuvuudessa toimivalla tietokonehiirellä.

Pisteiden merkitseminen toteutettiin asettamalla suorakulmaisen särmiön muotoinen kappale seuraamaan ohjaimen sijainti- ja kiertoarvoja. Näin käyttäjä voi vapaasti valita merkattavan kohdan pistepilvestä asettamalla särmiön niin, että sen sisässä on pistepilven pisteitä. Tämän jälkeen ohjainta liikutettaessa särmiön sisälle jäävät uudet pisteet tulevat myös merkatuksi. Käyttäjälle tämän valinnan voi esittää esimerkiksi värjäämällä merkatut pisteet eriväriksi. Kuvassa 15 näytetään miten särmiötä leikanneet pisteet tulevat merkatuksi HTC Viven ohjainta liikutettaessa.



KUVA 15. Pistepilven pisteiden merkkaus ohjaimen avulla (Pistepilvidata: Huitl ym. 2012)

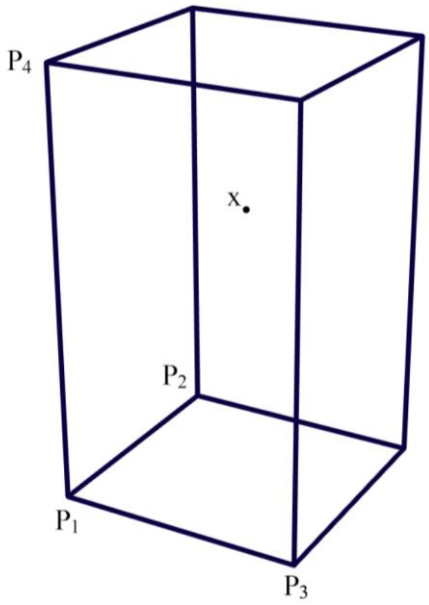
5.2 Pisteiden rajaus Unityssä

Merkkauksessa tarkasteltavat yksiköt ovat pisteitä kolmiulotteisessa koordinaatistossa ja niitä voi olla useita miljoonia kappaleita. Pisteiden XYZ-koordinaatit on siirretty oktanttien mukaisiin 3D-verkkoihin lisäinformaatioineen. Näille 3D-verkossa oleville pisteille täytyy suorittaa tarkastus, jossa katsotaan, sijaitseeko pisteitä ohjaimen särmiön kokoisella alueella.

Unityn fysiikkamoottorin valmiit tarkastelufunktiot, kuten Physics.Raycast tai Physics.OverlapBox käyttävät tarkasteluissaan Collider-komponentteja (Unity 2018h). Pistepilvistä muodostetuilla 3D-verkoilla ei kuitenkaan ole lainkaan fysiikkainformaatiota, minkä takia Unityn valmiit päällekkäisyystarkastelufunktiot eivät sovellu tähän tarkoitukseen. Vaikka 3D-verkoille loisi omat MeshCollider-fysiikkakomponentit, saisi päällekkäistarkasteluista tiedon leikatuista 3D-verkoista, mutta ei yksittäisistä pisteistä. Puolestaan Unityn Bounds-luokka sisältää funktion tarkistukselle, jossa katsotaan, kuuluuko piste määriteltyyn XYZ-akselien mukaiseen alueeseen (Unity 2018a). Tätäkään ei voida toteutuksessa käyttää särmiön sisäkkäisyystarkasteluun sen akselisuuntaisuuden takia, koska valitsimen pitää pystyä liikkumaan ja kääntymään vapaasti. Näillä funktioilla ei siis suoraan pysty tarkastamaan onko yksittäinen piste mielivaltaisen särmiön sisällä. Tämän takia toteutettiin sisältyvyystarkastus ilman Unityn fysiikkafunktioita lukuun ottamatta valitsinsärmiön lähtöarvojen hakua BoxCollider-komponentista.

5.2.1 Rajaus suorakulmaisen särmiön alueelle

Tarkastus siitä, ovatko pisteet suorakulmaisen särmiön sisällä, voidaan tehdä matematiikan avulla. Kuvio 4 havainnollistaa tilannetta, kun keskitytään tarkastelemaan yhtä pistettä suorakulmaisessa särmiössä, jonka kulmakoordinaatit tunnetaan. (Empy2, Jealie, Faas 2016). Toteutuksessa Unityssä tämä tarkistus toistetaan kaikille octree-rakenteelta saaduille 3D-verkon pisteille. Kuvion 4 särmiö vastaa ohjaimeen kiinnitettyä särmiötä, joka voi olla 3D-avaruudessa mielivaltaisessa sijainnissa ja asennossa.



KUVIO 4. Suorakulmainen särmiö, jossa *x* on särmiön sisällä oleva piste. (Empy2, ym. 2016, muokattu)

Sisäkkäisyystarkistusta varten pitää ensin laskea suorakulmaisen särmiön kohtisuorat reunat *u, v ja w*

$$u = P_1 - P_2, \tag{1}$$

$$v = P_1 - P_3 \text{ ja} \tag{2}$$

$$w = P_1 - P_4, \tag{3}$$

jossa P_1 on särmiön kulma, jossa kohtisuorat vektorit leikkaavat. P_2, P_4 sekä P_4 ovat tämän viereiset kulmat (Kuvio 4). Piste *x* on särmiön sisällä, jos seuraavat pistetulolausekkeet

$$u \cdot P_1 < u \cdot x < u \cdot P_2 \text{ tai } u \cdot P_2 < u \cdot x < u \cdot P_1, \tag{4}$$

$$v \cdot P_1 < v \cdot x < v \cdot P_3 \text{ tai } v \cdot P_3 < v \cdot x < v \cdot P_1 \text{ ja} \tag{5}$$

$$w \cdot P_1 < w \cdot x < w \cdot P_4 \text{ tai } w \cdot P_4 < w \cdot x < w \cdot P_1 \tag{6}$$

ovat tosia. Lausekkeissa (4), (5) ja (6) muuttujat u, v ja w ovat yhtälöiden (1), (2) ja (3) tulosvektorit ja P_1, P_2, P_4 sekä P_4 ovat yhtälöiden (1), (2) ja (3) särmiön kulmapisteet. (Empy2 ym. 2016).

Ennen kuin nämä ehdot voi siirtää Unityyn, on pelimoottorista saatava lausekkeiden lähtöarvot eli tarkistettavan särmiön kohtisuorat kulmat ja tarkastettavat pisteet. Ensin on järkevää laskea kaikkien tarkasteltavien pisteiden kannalta yleiset arvot (Kuva 16). Näin säästyy suoritustehoa, kun ainoastaan yhdelle pisteelle ainutlaatuiset arvot lasketaan vertausta tehtäessä. Särmiön kulmat voidaan laskea lisäämällä särmiöön BoxCollider-komponentti. BoxColliderin keskikohdan ja koon avulla voidaan laskea särmiön kulmapisteet maailmakoordinaatistossa. Kohtisuorat suuntavektorit voidaan näiden avulla laskea ja tallentaa myöhempiä vertauksia varten. Pisteen vertaamiseen käytetyt pelkän särmiön arvoista selville saatavat pistetulot voidaan laskea ennen pistepilvidatan vertaamista näihin arvoihin.

```
// BoxCollider edge points
vertices[0] = box.transform.TransformPoint(
    box.center + new Vector3(box.size.x, box.size.y, box.size.z) * 0.5f);

vertices[1] = box.transform.TransformPoint(
    box.center + new Vector3(-box.size.x, box.size.y, box.size.z) * 0.5f);

vertices[2] = box.transform.TransformPoint(
    box.center + new Vector3(box.size.x, -box.size.y, box.size.z) * 0.5f);

vertices[3] = box.transform.TransformPoint(
    box.center + new Vector3(box.size.x, box.size.y, -box.size.z) * 0.5f);

// Calculate u, v, w & save them to array
directions[0] = vertices[0] - vertices[1];
directions[1] = vertices[0] - vertices[2];
directions[2] = vertices[0] - vertices[3];

// Calculate Dot products for comparing point values
float uP1 = Vector3.Dot(directions[0], vertices[0]);
float uP2 = Vector3.Dot(directions[0], vertices[1]);

ComparisonDotProducts[0] = Mathf.Min(uP1, uP2);
ComparisonDotProducts[1] = Mathf.Max(uP1, uP2);

float vP1 = Vector3.Dot(directions[1], vertices[0]);
float vP3 = Vector3.Dot(directions[1], vertices[2]);

ComparisonDotProducts[2] = Mathf.Min(vP1, vP3);
ComparisonDotProducts[3] = Mathf.Max(vP1, vP3);

float wP1 = Vector3.Dot(directions[2], vertices[0]);
float wP4 = Vector3.Dot(directions[2], vertices[3]);

ComparisonDotProducts[4] = Mathf.Min(wP1, wP4);
ComparisonDotProducts[5] = Mathf.Max(wP1, wP4);
```

KUVA 16. Suorakulmaisen särmiön sisäkkäisyystarkastelun pisteiden kannalta yleisten arvojen laskeminen C#-ohjelmointikielellä

3D-verkot, joissa pisteet sijaitsevat, voidaan hakea octree-rakenteesta samalla funktiolla mitä käytetään pisteryhmien hakemissa kuvaruudulle piirtämistä varten. Unityn Mesh-luokka sisältää vertices-ominaisuuden, jolla 3D verkon pisteet saadaan kopioitua 3D-ver-

kosta Vector3-taulukkoon maailmakoordinaatistoon (Unity 2018f). Mikäli tässä rakenteessa oleva piste täyttää yhtälöiden (4), (5) ja (6) ehdot, voidaan todeta, että piste on määritellyn BoxCollider-laatikon sisällä (Kuva 17). Merkattujen pisteiden Vector3-sijainnit voidaan tallettaa tietorakenteeseen myöhempää käyttöä varten.

```
public static bool ContainsPoint(Vector3 point, Vector3[] directions, float[] comparisonDotProducts)
{
    float ux = directions[0].x * point.x + directions[0].y * point.y + directions[0].z * point.z;
    float vx = directions[1].x * point.x + directions[1].y * point.y + directions[1].z * point.z;
    float wx = directions[2].x * point.x + directions[2].y * point.y + directions[2].z * point.z;


    if (ux > comparisonDotProducts[0] && ux < comparisonDotProducts[1] &&
        vx > comparisonDotProducts[2] && vx < comparisonDotProducts[3] &&
         wx > comparisonDotProducts[4] && wx < comparisonDotProducts[5])
    {
        return true;
    }

    return false;
}
```

KUVA 17. Yhden pisteen sisäkkäisyystarkastelu suorakulmaisen särmiön suhteen C# ohjelmointikielessä

5.2.2 Rajauksen näyttäminen käyttäjälle visuaalisesti

Sisäkkäisyystarkastelun jälkeen käyttäjälle on ilmoitettava, mitkä pisteet merkattiin. Korostamalla merkattujen pisteiden värejä käyttäjä voi helposti erottaa nämä pisteet muista. Pistekohtaiset värit saadaan kopioitua muokattavaksi pisteryhmän Mesh-objektilta colors32-ominaisuudella, joka palauttaa taulukon tavumuodossa olevia värejä (Unity 2018c). Koska värejä on 3D-verkossa yhtä monta kuin pisteitä, särmiön sisällä olevien pisteiden indeksejä voi käyttää määrittelemään mitkä värit muutetaan väritaulukossa. Kun värit on vaihdettu, ne voidaan asettaa takaisin Mesh-objektiin, jonka jälkeen käytetty varjostin vastaa siitä, miten värit kuvaruudulla näytetään. Aiemmissa esimerkeissä pisteiden piirtoon käytetty neliögeometriavarjostin kertoi pistekohtaisen värin varjostimen tekstuurin värillä tuottaen lopullisen värin (smb02dunnal ym. 2018).

Pisteiden 3D-verkon värien alfakanavaa eli läpinäkyvyysarvoa ei hyödynnetä toteutuksessa pisteiden piirtämisessä, joten sitä voidaan käyttää säilyttämään muuta tietoa. Siihen voidaan esimerkiksi kirjata, mikäli piste on jo merkattu, jolloin matemaattisia tarkastuksia ei tarvitse myöhemmin tehdä samalle pisteelle. Tämä myös takaa sen, että jos käyttäjä haluaa tehdä kaksi erillistä merkkausta, voidaan alfakanavalla estää samojen pisteiden

merkkaaminen kahteen eri osioon. Varjostimen värin määritys tarvitsee kuitenkin ohjelmoida niin, että varjostin ei käytä alfakanavaa pisteiden läpinäkyvyyden määrittämiseen. Kaiken lisäksi oletusarvoisesti värittömät eli mustat pisteet piilottavat tekstuurit värien kertolaskun tuloksena. Oletusvärin vaihtamisen sijasta voidaan näytettävän värin laskennassa hyödyntää pisteen alfakanavaa, kun käsitellään yksiväristä pistepilveä.

Kuvassa 18 on esitelty kolme eri tapaa määritellä pisteiden värit varjostinlogiikan pikselivarjostimessa. Ensimmäisessä vaihtoehdossa pisteistä näytetään vain värit jättäen tekstuuriominaisuus kokonaan pois. Tätä vaihtoehtoa on hyvä käyttää, jos pistepilvi on värillinen ja vain todelliset värit halutaan näyttää. Toisessa lasketaan tekstuurien ja pisteiden värien keskiarvo. Tämä tapa toimii hyvin värillisiin pistepilviin, jos myös tekstuuri halutaan näkyviin. Viimeinen vaihtoehto säilyttää tekstuurin oletusvärit värittömässä eli mustassa pistepilvessä, mutta näyttää merkatut värit riippuen 3d-verkon alfakanavan arvosta. Tekstuurin voi näyttää oletusarvoisena, kun alfakanavan arvo on nolla eli pistettä ei ole merkattu ja värin voi kertoa merkkausvärillä, kun piste on merkattu ja alfakanava on yksi. Tämä tapa toimii hyvin värittömän pistepilvien kanssa, kun tarkoitus on saada tekstuuri näkymään muuttumattomana merkkaamattomissa pisteissä ja samalla säilyttää tietorakenteissa pisteiden oletusväri.

Pisteiden värin määritys ilman tekstuuria

```
input.color.a = 1;
return input.color;
```

Pisteiden värin määritys tekstuurin värin ja pisteen värin keskiarvona

```
float4 finalColor = _PointTexture.Sample(sampler_PointTexture, input.tex0);
finalColor = (finalColor + input.color) / 2;
finalColor.a = 1;
return finalColor;
```

Pisteiden värin määritys musta väri näytettäessä tekstuurin oletusväreillä

```
float4 finalColor = _PointTexture.Sample(sampler_PointTexture, input.tex0);
finalColor = finalColor * (input.color * _Strenght + 1 - input.color.a);
finalColor.a = 1;
return finalColor;
```

KUVA 18. Pikselivarjostimen eri värinnäyttötapoja, joissa lopullinen läpinäkyvyysarvo on riippumaton pisteiden alfakanavasta

Yhdistämällä matemaattiset sisäkkäisyystarkastelut ja pisteiden värjäys virtuaalitodellisuusrajapinnan avulla ohjaimiin, voidaan pistepilvestä kätevästi merkata alueita. Merkkaamisen väriksi voi valita haluamansa värin, jonka ansiosta samasta pistepilvestä voi merkata useita osia eri väreillä (Kuva 19). Koska pisteiden värjäys merkkausvärillä poistaa varjostimesta alkuperäisen värin, kannattaa värillisen pistepilvidatan alkuperäiset värit tallettaa merkattujen pisteiden kanssa samaan tietorakenteeseen. Alkuperäisiä värejä voidaan käyttää, jos ajon aikana merkkaus päätetään poistaa tai merkattu osa tallettaa levylle alkuperäisillä väreillä.

KUVA 19. Useampia merkattuja alueita pistepilvestä (Pistepilvidata: Huitl ym. 2012)

5.3 Pistepilven merkkaamisen optimointi säikeistyksellä

Kun pisteiden rajauslogiikka ja värjääminen toimivat, osoittautuu ohjelman rajoittavaksi tekijäksi sen suorituskyky. Ohjelman kuvataajuus vähenee huomattavasti, kun särmiön sisäkkäisyystarkasteluja suoritetaan ohjelman ajon aikana. Tämä johtuu siitä, että merkatessa pistetarkastukset täytyy suorittaa joka kerta kun ohjainta liikutetaan mahdollisten

uusien pisteiden vuoksi. Vaikka pisteet nyt värjäytyvätkin oikein, virtuaalitodellisuuskäyttöön ohjelma ei vielä sovellu tämän suorituskykyrajoitteen takia. Ongelman ratkaisemiseksi pitäisi lukuisat sisäkkäisyystarkastelut suorittaa niin, että ohjelma ei vähennä uusien kuvien piirtonopeutta pisteiden merkkauksen aikana.

Säikeistyksellä tarkoitetaan sellaista ohjelmointia, joka hyödyntää prosessorin kykyä suorittaa monia säikeitä samaan aikaan käyttäen useita suorittiminen ytimiä. Sen sijaan, että tehtävät tai komennot suoritetaan toistensa jälkeen, säikeistyksen avulla niitä voidaan ajaa samanaikaisesti. (Unity 2018o). Oletusarvoisesti C#-ohjelmalla on yksi säie käytössään ja lisäsäikeitä voidaan luoda tarvittaessa. Yleinen käyttötapa säikeille on käyttää niitä suorittamaan aikaa vieviä tai aikakriittisiä tehtäviä, jotka eivät tarvitse useaa muiden säikeiden käyttämää resurssia. (Kulikov ym. 2015). Pistepilvien merkkaustarkastelu on aikaa vaativa toimenpide, jonka takia suorituskyvystä johtuvalle ongelmalle haettiin ratkaisua säikeistämällä nämä tehtävät. Lisäksi säikeistys mahdollistaa useamman 3D-verkon pisteitten sisäkkäisyystarkastelun samanaikaisesti, joka myös nopeuttaa tarkastusoperaatiota moniytimisellä suorittimella.

Säikeistetyn koodin kirjoittamista pidetään monimutkaisena ja haastavana johtuen siitä, että monet oletukset mitkä pätevät yksisäikeisisiin ohjelmiin eivät enää ole totta monisäikeisissä ohjelmissa. Samaa objektia yhtä aikaa käsittelevien kahden säikeen suoritusjärjestyksestä ei ole varmuutta. Tämä voi aiheuttaa sen, että toinen säie lukee objektin arvon samalla kun toinen säie on vasta osittain muuttanut sen arvoja. Tilannetta, jossa ohjelman toiminta riippuu siitä, miten käyttöjärjestelmä on ajoittanut säikeet, kutsutaan nimellä ”race condition” eli kilpailutilanne. Kilpailutilanteita sisältävä ohjelma voi toimia oikein suurimman osan ajasta ja väärin vain joskus, mikä hankaloittaa säikeistettyä ohjelmointia. Ohjelmakoodia, joka toimii oikein säikeistetyssä ohjelmassa, kutsutaan säieturvalliseksi. (Michaelis & Lippert 2016, 734, 738-740). Pistepilviä merkatessa pitää siis varmistaa, että säikeistetyt merkkaustarkastelut käyttävät jaettuja tietorakenteita, kuten pisteiden sijainteja, säieturvallisesti ilman, että mahdollisia kilpailutilanteita syntyy säikeiden välille.

5.3.1 Säikeistys Unityssä

Myös Unityssä voidaan luoda säikeitä. Unityn ohjelmointirajapinta ei kuitenkaan ole säieturvallinen, jonka takia kutsuja Unityn rajapintaan ei pitäisi tehdä muualta kuin Unityn pääsäikeestä. (Bonastre 2016). Toinen tapa säikeistää asioita on käyttää Unityn kehittämää *Unity C# Job System* -ominaisuutta. Unityn kehittäjän Tim Johanssonin (2018) mukaan Job System jakaa pelin logiikan pieniin työyksikköihin, joita yksi säie suoritinydintä kohden suorittaa jonotietorakenteesta. Hänen mukaansa Unityn toteutus estää kilpailutilanteiden ilmaantumisen. Lisäksi Job Systemin työyksikköjä on mahdollista integroida toimimaan yhteen joidenkin Unityn järjestelmien kanssa. Tiedon siirtämiseen työyksikköihin käytetään Unityn NativeContainer-tietorakenneluokkia perinteisten C#-tietorakenteiden sijaan. (Johansson 2018).

Job Systemin käytön toteutuksessa esti se, että ajoitetut Job Systemin työt pitää saada suoritettua neljän kuvan aikana töiden Schedule-ajoittamismetodin kutsumisesta johtuen Unityn sisäisistä allokaatioista. Tämä rajoite on suunniteltu korjattavaksi tulevaisuudessa, mutta nykyisillä Unity versiolla ohjelman syötteeseen tulostuu varoituksia, mikäli töiden ajoituksesta on kulunut yli neljä kuvaa. (Johansson & Ante 2018). Pistepilvessä verrattavia pisteitä voi olla suuressa tai tiheässä octree-rakenteessa hyvin paljon riippuen merkattavan alueen koosta. Lisäksi aina särmiötä liikutettaessa täytyy suorittaa uusia tarkasteluja, mikä kasvattaa töiden määrää. Tämä tarkoittaa sitä, että jos töiden lukumäärä on hetkellisesti suurempi kuin säikeiden määrä, työn ajoittamien ei välttämättä heti aloita työn suorittamista. Koska neljän kuvan rajoite lasketaan ajoitusmetodin kutsumisesta, voi se vaikuttaa sellaisten pisteryhmienkin tarkastuksiin, joiden suorittamista ei vielä ole aloitettu.

Virtuaalitodellisuuskäytössä pääsäikeen eli kuvan renderöinnistä vastaavan säikeen pysäyttäminen ei ole suositeltavaa, jotta merkkaustarkistukset ehtisivät valmiiksi neljän kuvan aikana. Pienessä viiveessä värjättyjen pisteiden näkymisessä ei ole yhtä paljon haittaa käyttäjälle kuin koko kuvan pysähtymisestä, mikäli merkkaustarkistukset eivät ehdi ajoissa valmiiksi. Työsäikeiden ajoitusta lomittamalla ja tarkistettavien pisteverkkojen suuruuksia jakamalla olisi mahdollisesti voitu saada ajastetut työsäikeet suoritettua alle neljässä kuvassa. Ohjelmoinnin selkeyden takia päädyttiin kuitenkin käyttämään Job Systemin sijasta tavallisia säikeitä, jotka ovat riippumattomia renderöityjen kuvien määrästä.

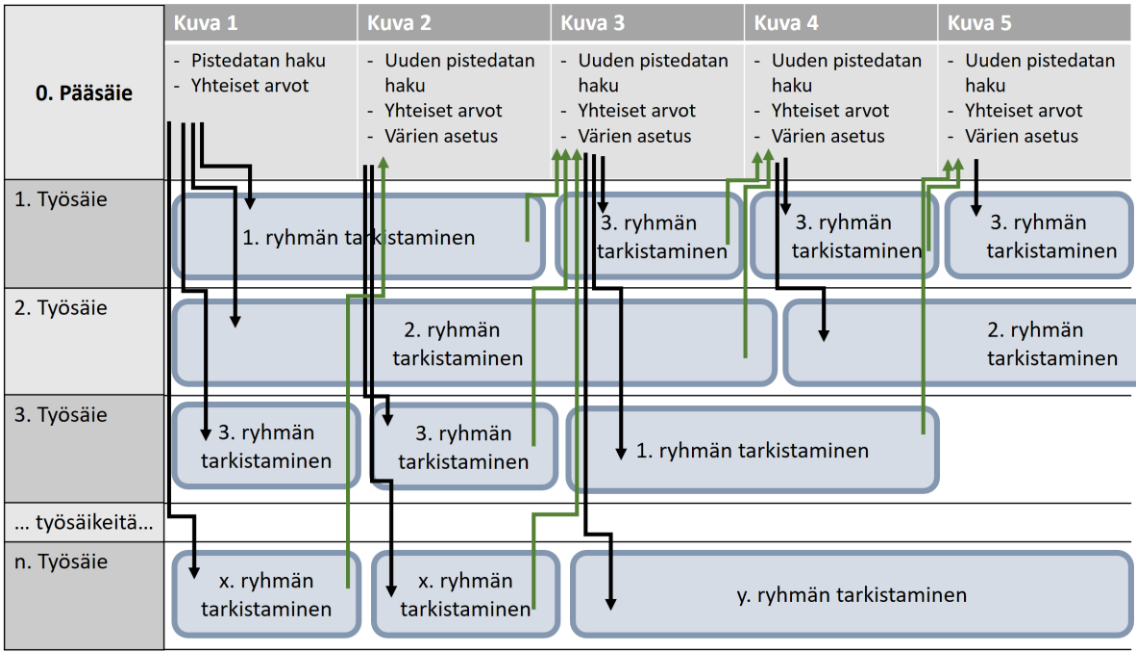
Tällä tavoin yhden verkon pistevertailut voitiin suorittaa omana tehtävänään pilkkomatta niitä enempää pääsäikeellä.

5.3.2 Rajauksen tehokkuuden parantaminen monisäikeistyksellä

Koska Job Systemin tapa jakaa tehtävät eri säikeille vaikutti käytännölliseltä lähestymistavalta merkkaustarkastelujen muodostamiseksi, otettiin samankaltainen malli käyttöön tavallisille säikeille. Esimerkki tällaisen työmallin mahdollistamasta viitekehyksestä Unitylle on MIT-lisenssillä vapaasti käytettävissä oleva Job Queue (Bunny83 2016).

Rajauksen nopeuttamiseksi kaikki pistekohtaiset tarkastelut siirrettiin pois Unityn pääsäikeeltä, jotta laskelmat eivät vaikuta Unityn FPS:ään. Säieturvallisuuden takia pääsäikeen suoritettavaksi täytyi jättää pistedatan hakeminen 3D-verkoilta. Tarkasteltavasta särmiöalueesta riippuvaiset kuusi pistetuloa voidaan laskea ennakkoon pääsäikeellä. Kun nämä tiedot on laskettu kerran renderöityä kuvaa kohti, ne voidaan välittää työsäikeille pisteiden sijaintien kanssa tarkistuksia varten. Näin työsäikeiden ei tarvitse tehdä koko laskutoimitusta yksittäistä pistettä kohti, vaan tarkastusten kannalta yleiset tulokset on jo kerran laskettu valmiiksi. Käytettävien säikeiden määrän voi Job Queuessa määrittää vapaasti (Bunny83 2016). Unityn (2018o) dokumentaatiossa huomautetaan, että jos säikeitä on käytössä enemmän kuin suoritinytimiä, joutuvat säikeet kilpailemaan suorittimen resursseista keskenään. Tämän takia käytettyjen säikeiden määrä asetettiin toteutuksessa vastaamaan suorittimessa olevia säikeitä, joita testitietokoneella oli kahdeksan.

Kuviossa 5 havainnollistetaan pisteinformaation siirtymistä pää- ja työsäikeiden välillä ohjelman ajon aikana, kun lisäsäikeitä hyödynnetään vain sisäkkäisyystarkasteluissa. Työsäikeille lähetetään aluksi työn kannalta olennaiset arvot yhden 3D-verkon pisteryhmästä, jonka jälkeen pääsäie ei vaikuta työsäikeen etenemiseen. Kun säie on suorittanut tarkastuksensa loppuun, voidaan merkattujen pisteiden värit asettaa 3D-verkolle pääsäikeessä, jonka jälkeen muutetut värit näkyvät sovelluksessa. Merkattuun alueeseen liittyvät tiedot, kuten pisteiden indeksit ja alkuperäiset värit voidaan tallentaa säikeessä pisteryhmäkohtaisesti. Nämä pisteryhmät kuuluvat puolestaan tietorakenteeseen, joka muodostaa koko merkatun alueen. Tällainen työnjako mahdollistaa octree-rakenteelta saatujen eri 3D-verkkojen tarkistamisen samanaikaisesti.



KUVIO 5. Tiedon siirtäminen pääsäikeeltä työsäikeille ohjelman ajon aikana

Pistepilven samojen oktettien tarkastaminen yhtäaikaisesti aiheuttaisi kuitenkin kuvion 5 käytetyssä mallissa kilpailutilanteen, jossa yksi tarkistussäie voi kirjoittaa arvonsa toisen säikeen tulosten päälle. Octreen oktetin eli yhden pisteryhmän tarkasteltava alue muuttuu valitsinta liikutettaessa ajon aikaan. Tällöin kahdella peräkkäisellä kuvalla valitsimen alueeseen kuuluvat pisteet eivät ole täysin samat. Jos nämä tarkistukset suoritetaan samaan aikaan voi yksi säie kumota toisen säikeen tallentamat värimuutokset, vaikka molemmat alueet pitäisi huomioida. Kilpailutilanteen välttämiseksi samassa osiossa tehtävien tarkastusten alkua voi viivästyttää, kunnes aiempi tarkistus ryhmään on suoritettu. Näin ohjelman FPS pysyy sulavana ja kilpailutilanteet on vältetty.

Vaikka ohjelma on virtuaalitodellisuuden kannalta käyttökelpoinen tällä tekniikalla, merkkaukset saattavat näkyä käyttäjälle viiveellä. Mitä enemmän saman pisteryhmän tarkistuksia ohjelma joutuu suorittamaan lyhyessä ajassa, sitä enemmän se joutuu jonottamaan samaan pisteryhmään kohdistuvien tarkastustöiden alkua. Tämä ilmenee toteutuksessa, kun valitsinta liikutetaan pitkiä matkoja, jolloin tarkistettavia alueita kertyy paljon. Mikäli viiveen vähentäminen on ohjelman kannalta tärkeää, täytyy tiedonsiirtoa muuttaa. Se voidaan tehdä käyttämällä sisäkkäisyystarkasteluiden tulosten tallentamiseen sellaisia tietorakenteita, jotka mahdollistavat säieturvallisen tavan tallettaa samaan pisteryhmään koskevaa tietoa. Tällöin tarkastuksia ei tarvitse viivästyttää lainkaan vaan tarkistukset

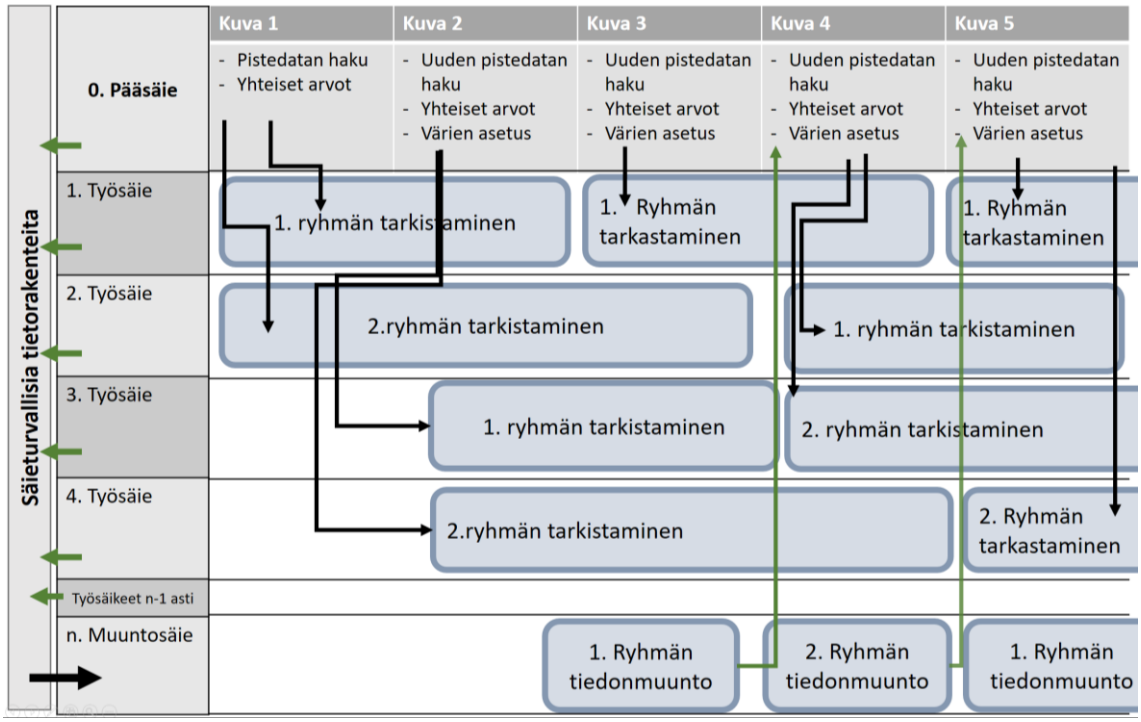
voidaan suorittaa heti. Toisaalta tällaisten tietorakenteiden käyttö hankaloittaa Unityn rajapinnalle lähetettävää tietoa, koska tieto pitää muuntaa sopivan muotoiseksi ennen kuin sitä voi käyttää rajapinnan kanssa. Esimerkiksi merkatuiden pisteiden värit pitää lähettää Unitylle array-taulukkomuodossa, jonka takia ne täytyy kopioida säieturvallisista rakenteista aiheuttaen lisää muistinkulutusta.

Säieturvallisia tietorakenteita, kuten ConcurrentDictionary tai ConcurrentQueue, on valmiiksi käytettävissä System.Collections.Concurrent-nimiavaruudessa. Näitä luokkia suositellaan käytettäviksi, kun useat säikeet käyttävät tietorakenteita samanaikaisesti (Microsoft n.d. g). Tietojen tallentamiseen käytettiin ConcurrentDictionary-rakenteita, jotka tarjoavat mahdollisuuden tallettaa avain-arvopareja säieturvallisesti (Microsoft n.d. d). ConcurrentDictionary-rakenteita luotiin jokaiselle octree-rakenteesta saadulle pisteryhmälle, jolloin niihin voitiin laittaa avaimeksi pisteen 3D-verkon indeksi ja arvoon tallettaa merkattuihin pisteisiin liittyvää informaatiota. Näin samaan aikaan suoritetut työsäikeet pystyvät lisäämään särmiön sisällä olevat pisteet rakenteeseen ilman kilpailutilanteiden syntymistä.

Vaikka pisteiden tarkistus samanaikaisesti helpottuu säieturvallisten tietorakeinteiden ansiosta, se samalla hankaloittaa Unityn-rajapinnalle lähettävien värien asettamista. Ilman saman 3D-verkon pisteiden yhtäaikaista käsittelyä pisteryhmän värit voitiin asettaa taulukkotietorakenteeseen säikeen suorittamisen lopussa ja tämä rakenne asettaa pääsäikeellä takaisin 3D-verkkoon Unityn Mesh.colors32-ominaisuudella. Laitteen resurssien käytön kannalta ei kuitenkaan ole järkevää luoda aina uutta kokonaista väritaulukkoa jokaisen samaa pisteryhmää käsittelevän säikeen lopuksi. Värirakenteen muuntaminen Unitylle sopivaan muotoon voidaan ajoittaa tapahtumaan omalla säikeellään niin, että samaan aikaan ajettujen tarkistussäikeiden tulokset muunnetaan sopivaan muotoon yhdellä kertaa.

Kuviossa 6 havainnollistetaan tiedon kulkua säikeissä kahden kuvitteellisen pisteryhmän tarkastuksella. Esimerkissä kuvion ensimmäisen pisteryhmän tarkastamiseen on määritelty kuluvan aikaa noin kahden kuvan verran ja toiselle suuremmalle pisteryhmälle lähes kolme kuvan verran. Vaikka tarkistus edelliseltä kuvalta on käynnissä, voidaan seuraavalla kuvalla aloittaa jo saman pisteryhmän tarkistaminen vapailla säikeillä. Kun jokin säie on valmis, voidaan välittää tieto, että merkkausvärit voidaan muuntaa Unitylle sopi-

vaan muotoon. Tämän jälkeen muuntosäikeellä luotu tehtävä noutaa säieturvallisesta rakenteesta merkattujen pisteiden arvot ja luo niistä uuden array-taulukkorakenteen, joka pääsäikeessä voidaan asettaa 3D-verkkoon Mesh.colors32-ominaisuuden kautta. Mikäli muita samaan verkkoon tarkastuksia suorittavia tehtäviä olisi valmistunut ennen muuntosäikeen tehtävän aloittamista, olisi niiden värimuutokset asetettu samalla kertaa. Sama asia tehdään toiselle sekä muille mahdollisille pisteryhmille. Toisin sanoen, kun pisteryhmälle suoritettu tarkistustehtävä on valmis, voidaan tämän perusteella aloittaa rakenteen muuntaminen käsiteltävälle pisteryhmälle muuntosäikeellä.



KUVIO 6. Tiedon siirtyminen säikeiden avulla säieturvallista tietorakennetta käytettäessä

Säieturvallisten rakenteiden käytön ansiosta merkatut pisteet näkyvät käyttäjälle nopeammin kuin viivyttämällä samaan pisteryhmään kohdistuvien tarkistuksia. Toteutuksessa huomattiin, että tiedonsiirto Unityn rajapinnan avulla 3D-verkosta ja sinne takaisin on hidas toiminto. Unityn dokumentaation (2018c; 2018f) mukaan sekä Mesh.vertices että Mesh.colors32 -ominaisuudet palauttavat kopion tiedoistaan pelkän viitteen sijasta, mikä selittää funktioden hitautta. Koska nämä käyttävät Unityn rajapintaa, ei niiden kutsuminen Unityn pääsäikeen ulkopuolelta ole säieturvallista. Tämän takia sujuvan kuvataajuuden ylläpitämiseksi on järkevää välttää käyttämästä aikaa vieviä toimintoja liian montaa kertaa lyhyessä ajassa.

Pisteinformaatiota koskien suorituskykyä voidaan säästää tallettamalla tärkeät tiedot merkkauksen ajaksi muistiin, ettei niitä tarvitse hakea rajapinnalta niin usein. Lisäksi pisteiden sijaintien ja värien kopiointia voi tarpeen mukaan jakaa tapahtumaan useamman kuvan ajalle, jolloin yhden kuvan aikainen kuormitus jää pienemmäksi. Tämä kuitenkin puolestaan lisää viivettä käyttäjälle, jonka takia toimintojen jakamista usean kuvan ajalle kannattaa käyttää vain sen verran että kuvataajuus pysyy ohjelman tavoiteluvuissa. Myös pisteiden tarkastuksien määrän vähentäminen edesauttaa sulavan suorituskyvyn ylläpitämistä ilman suuresti näkyvää viivettä. Tarkistusten määrää voidaan esimerkiksi vähentää tekemällä tarkistukset vain valitsimen liikuttua tarpeeksi edellisestä sijainnista, jolloin tarkastuksia ei turhaan tehdä samalle kohdalle useasti.

5.4 Merkatun datan tallentaminen PCD-muotoon

Merkatun pistepilvidatan voi tarvittaessa tallettaa kiintolevylle PCD-muodossa, mikä mahdollistaa sen tarkastelun muissa ohjelmissa, kuten CloudComparessa. Käytännössä tämä voidaan toteuttaa ottamalla alkuperäisen PCD-tiedoston otsake ja päivittämällä siitä tarvittavat kentät, kuten pisteiden lukumäärä, vastaamaan merkattua aluetta. Jos otsakkeen jokainen rivi on tiedoston lukuvaiheessa tallennettu muistiin, voidaan niistä tarvittavat osiot päivittää täsmäämään merkattua aluetta.

Mikäli pistedatan lukuvaiheessa on jätetty huomioimatta joitain FIELDS-rivin osia, kuten mahdollisesti pisteiden normaalit, täytyy ne poistaa otsakkeesta, jotta muut ohjelmat osaavat lukea tiedoston. Kun otsake vastaa merkattua osaa, se voidaan kirjoittaa tiedostoon tekstimuotoisena. Tämän jälkeen pisteet ja niiden lisäinformaatio kirjoitetaan tiedostoon binäärimuotoisena. BinaryWriter-luokka sisältää funktiota, joilla voidaan kirjoittaa primitiivisiä tietotyyppejä, kuten kokonais- ja liukulukuja (Microsoft n.d. b). Myös ASCII-muotoista tekstiä voi kirjoittaa BinaryWriter-luokalla muuntamalla sen tavuiksi Encoding.ASCII.GetBytes-funktion avulla (Microsoft n.d. e). Näin otsake voidaan kirjoittaa samalla luokalla kuin varsinainen binäärimuotoinen pistedata, jonka takia yksi kirjoitusluokka on tarpeeksi koko tiedoston luomiseen. Tiedosto kirjoittuu oikein, kun pistepilvidata kirjoitetaan FIELDS-kentän osoittamassa järjestyksessä tiedostoon ja informaation tavukoot sekä lukumäärät vastaavat otsakkeen määriä. Koska tiedoston kirjoittamiseen ei käytetä Unityn ohjelmointirajapintaa, voidaan tiedosto luoda taustalla toisella säikeellä ajon aikana.

6 JOHTOPÄÄTÖKSET JA POHDINTA

Opinnäytetyössä tutkittiin pisteiden merkkaamista Unityllä virtuaalitodellisuudessa ja niitä alustavia toimenpiteitä, jotka mahdollistavat pisteiden pistepilven lataamisen ja näyttämisen merkkaamista varten. Opinnäytetyön perusteella voidaan todeta, että Unityllä on mahdollista merkata pisteitä suurista yli 50 miljoonan pisteen pistepilvistä. Suurien pistemäärien kanssa työskennellessä toteutus on kuitenkin optimoitava niin, että ohjelman kuvataajuus pysyy tarpeeksi sulavana sekä pisteitä näytettäessä että niitä merkatessa. Opinnäytetyössä tämä taattiin käyttämällä octree-rakennetta pisteitä näytettäessä ja säikeistämällä matemaattiset tarkastelut pisteiden merkkausvaiheessa. Tämän perusteella opinnäytetyön lähtökohtaisissa tavoitteissa onnistuttiin.

Pistepilven yksittäisten pisteiden käsittelystä Unityn virtuaalitodellisuudessa saatiin opinnäytetyön seurauksena lisää tietoa. Pisteiden merkkaamisessa hyödynnettyä sisäkkäisyystarkastelutekniikkaa on mahdollista jatkokehittää osaksi muitakin toimintoja. Sitä voidaan hyödyntää sellaisissa ominaisuuksissa, joissa tavoitteena on päästä Unityn kautta käsittelemään pistepilvidataa reaaliaikaisesti, kuten esimerkiksi pisteiden siirtäminen tai poistaminen. Esimerkiksi pistepilven pisteiden poistamiseen tai niiden liikuttamiseen on mahdollista hyödyntää samoja tarkastelufunktiota kuin mitä pisteiden merkkaamisessa käytettiin. Verrattuna perinteisiin ohjaustapoihin pisteiden merkkausta oli sujuvaa käsitellä virtuaalitodellisuuslaitteiden ohjainten vapaan liikkuvuuden ansiosta. Tämän perusteella voisi olettaa muidenkin pisteiden käsittelyyn liittyvien toimenpiteiden toimivan hyvin virtuaalitodellisuudessa.

Pistepilvien näyttämistä ja suorituskyvyn ylläpitämistä koskevia aihealueita oli jo ennestään tutkittu paljon. Myös pistepilvien näyttämiseen Unityllä oli jo tehty aiempia ratkaisuja. Nämä tutkimukset antoivat suuntaa opinnäytetyön ongelmissa hyödynnetyille ratkaisulle. Opinnäytetyön tavoitteen eli pisteiden merkkaamisen saavuttaminen edellytti kuitenkin täyttä ymmärrystä koko prosessista käytetystä tiedostomuodosta lähtien pistepilven suorituskykyiseen näyttämiseen ja merkkaamiseen asti. Tämän takia oli perusteltua rakentaa opinnäytetyön toteutus vastaamaan suoraan työlle asetettuja tavoitteita.

Pisteiden lukemisen ja osittamisen ohjelmistoteknisen osan toteuttaminen itse mahdollisti sen, että pisteiden merkkaaminen hyödynsi samoja tietorakenteita ja funktioita kuin muut

ohjelman osat. Esimerkiksi merkkaustarkasteluihin pisteet voitiin suoraan hakea samasta octree-rakenteesta, johon ne oli lukemisen yhteydessä talletettu. Myös merkattujen pisteiden tallettaminen samaan tietomuotoon oli mahdollista sen ansiosta, että tiedostomuoto oli tutkittu sen lukijaa tehtäessä. Koska opinnäyteyön myötä koko prosessi pisteiden siirtämisestä selvitettiin, on tulevaisuudessa myös mahdollista laajentaa lukijaa toimimaan myös muihin tiedostomuotoihin ja hyödyntää niiden kanssa samoja ositus- ja näyttämistekniikoita.

Pisteiden merkkaamisessa käytetyssä tavassa rajoittavaksi tekijäksi osoittautui Unityn ohjelmointirajapinta, jota pystyi käyttämään vain pääsäikeeltä käsin. Koska 3D-verkossa olevien pisteiden arvoja käsiteltiin työssä näillä Unityn funktiolla, ei kaikkea voitu suorittaa pääsäikeen ulkopuolella. Vaikka pisteinformaatiosta voisi pitää kopion tavallisissa tietorakenteissa suurempaa keskusmuistin käyttöä vastaan, on silti vähintään merkatuiden pisteiden värit päivitettävä 3D-verkolle rajapinnan kautta. Koska Unityyn kehitetään paremman suorituskyvyn mahdollistavia ominaisuuksia, kuten C# Job System ja kokeellinen Entity Component System, ehkä myöhemmät Unity-versiot mahdollistavat 3D-verkon käsittelyn C#-ohjelmointikielellä nopeammin. Toinen mahdollisuus olisi suorittaa pisteiden tarkistus kohdelaitteen natiiviohjelmointikieltä käyttäen liitännäisellä, jonka avulla grafiikkarajapinnan käyttöön ei tarvitsi Unityn funktioita. Tällaisen toteutuksen tekeminen edellyttäisi kuitenkin natiiviohjelmointikielien, kuten C++:n käyttöä, mikä vaatisi erilaista osaamista kuin pisteiden merkkaamisen toteutus C#-ohjelmointikielellä.

Unityn ominaisuuspäivitysten lisäksi toteutuksen nopeutta voi olla mahdollista kehittää eteenpäin vähentämällä vaadittujen pisteryhmien merkkaustarkastelujen määrää. Niitä olisi mahdollista vähentää muuttamalla käytettyä ositusrakennetta pisteiden säilyttämiseen. Esimerkiksi, jos pisteet ositettaisiin ristikkomaiseen rakenteeseen ilman rekursiivisuutta, voitaisiin osa ylempien octree-tasojen pisteryhmien tarkasteluista välttää. Kuitenkin tällaisessa tapauksessa pistepilven hahmottamiseen hyödylliset karkeat ylätasot jouduttaisiin jättämään pois tai luomaan muulla perusteella kuin octree-rakenteen läpi suodattamalla. Rekursiivisten tasojen pois jättäminen tarkoittaisi myös sitä, että haettaessa pistepilviryhmiä joko näyttämistä tai merkkaamista varten jouduttaisiin tekemään enemmän vertauksia, jos kaikki mahdolliset pisteryhmät ovat samalla syvyys tasolla.

Toinen tapa muuttaa ositusrakennetta olisi kopioida rakenteen alimmille tasoille myös alueelle sijoittuvat ylemmän tason pisteet, jolloin pisteitä merkatessa voitaisiin tarkistaa

vain alimman tason pisteryhmät. Toisaalta tämä ratkaisu lisäisi osaltaan ohjelman muistinkulutusta, koska osa pisteistä esiintyisi usealla tasolla kopiona. Lisäksi pisteiden kopiointi usealle tasolle vaikeuttaisi värillä korostettujen pisteiden näyttämistä, koska olisi määriteltävä tapa, jolla merkatut osiot näkyisivät myös karkeammilla osioilla. Nämä muut ositustavat sisältävät siis omat haasteensa ja suorituskyky opinnäytetyössä käytetyn octree-rakenteen kanssa saavutti jo virtuaalitodellisuuskäytön edellyttämän kuvataajuuden. Tämän perusteella opinnäytetyössä käytetty rakenne soveltuu pisteiden merkkaamiseen suuresta tarkastettavien ryhmien määrästä huolimatta.

Opinnäytetyössä pistepilvidata pidettiin koko tarkastelun ajan tietokoneen keskusmuistissa ja 3D-verkot näytönohjaimen muistissa lähdetiedoston luvun jälkeen. Näin PCD-muotoinen tiedosto oli mahdollista avata toteutuksessa suoraan tallettamatta siitä mitään välissä kovalevylle. Myös pisteiden sisäkkäisyystarkasteluja tehtäessä voitiin tämän takia olettaa, että kaikki tarkasteltavat pisteet ovat ohjelman muistissa riippumatta tarkasteltavan alueen sijainnista tai koosta. Toisaalta käytetty ratkaisu rajoittaa ladattavan pistepilven suuruutta tietokoneen keskusmuistin mukaan. Tämä tarkoittaa, että hyvin suurten pistepilvien lataamiseen tarvitaan enemmän keskusmuistia tietokoneelta. Tästä rajoituksesta huolimatta opinnäytetyössä käytettyihin lähdetiedostoihin ratkaisu riitti hyvin, eikä keskusmuistinkulutus ylittynyt.

Opinnäytetyön tarkoitus, eli se osa tietokoneohjelmasta, jolla pistepilviä merkataan, tuotettiin asiakkaalle. Toteutettu käytännön työ toimi oikein ohjelmistolle asetettujen tavoitteiden mukaisesti. Opinnäyteyön toimeksiantaja Intopalo Digital Oy ja asiakas Sandvik Mining and Construction Oy olivat tyytyväisiä opinnäytetyön tuloksiin.

Yhteenvetona voi todeta, että Unity-pelimoottori osoittautui sopivaksi alustaksi pistepilvien merkkaamiselle. Työssä hyödynnettiin videopelien kehityksessä käytettyjä ominaisuuksia kuten virtuaalitodellisuutta ja reaaliaikaista renderöintiä perinteiseen ohjelmistokehitykseen. Tämän seurauksena painotettavat asiat ohjelmoinnissa erosivat tavallisesta pelinkehityksestä. Työ mahdollisti pelimoottorilla kehittämisen tietotaidon sisällyttämisen ei pelilliseen ohjelmistoon, jonka seurauksena toteutus pystyttiin valmistamaan nopeammin kuin jos koko sovellus olisi pitänyt tehdä kokonaan alusta asti.

LÄHTEET

Akehine-Möller T., Haines E. & Hoffman N. 2008. Real-Time Rendering. 3. Painos. Yhdysvallat: CRC Press.

Bonastre J. Why should I use Threads instead of Coroutines? Unityn tukiartikkeli. Luettu 20.10.2018. https://support.unity3d.com/hc/en-us/articles/208707516-Why-should-I-use-Threads-instead-of-Coroutines-

Bunny83. 2016. JobQueue. Avoin lähdekoodi säikeistetylle työjonolle Unityyn. Luettu 25.10.2018. http://wiki.unity3d.com/index.php/JobQueue

CloudCompare. 20.2.2015. CloudCompare octree. CloudCompare -ohjelman dokumentaatio. Luettu 11.10.2018. https://www.cloudcompare.org/doc/wiki/index.php?title=CloudCompare_octree

CloudCompare. N. d. CloudCompare-ohjelman kotisivut. Luettu 10.9.2018. https://www.cloudcompare.org/

Dimitrov K. Rendering a Point Cloud inside Unity. 27.04.2014 Blogikirjoitus. Luettu 9.10.2018. http://www.kamend.com/2014/05/rendering-a-point-cloud-inside-unity/

Empy2, Jealie & Faas. 21.10.2016. Check if a point is inside a rectangular shaped area (3D)? Matemaattinen kaava. Luettu 17.10.2018. https://math.stackexchange.com/questions/1472049/check-if-a-point-is-inside-a-rectangular-shaped-area-3d

Fraiss S. 2017. Rendering Large Point Clouds in Unity. Bachelor of Science in Media Informatics and Visual Computing. TU Wien. Kandidaatin opinnäytetyö.

Geier D. 18.8.2014. Advanced Octrees 2: node representations. Blogikirjoitus. Luettu 29.10.2018. https://geidav.wordpress.com/2014/08/18/advanced-octrees-2-node-representations/

Gobbetti E. & Marton F. 2004. Layered Point Clouds. Eurographics Symposium on Point-Based Graphics, CRS4 Visual Computing Group

Greenwald W. 18.9.2018 The Best VR (Virtual Reality) Headsets of 2018. PC Magazine UKn verkkolehtiartikkeli. Luettu 5.10.2018. https://uk.pcmag.com/consumer-electronics-reviews-ratings/75926/guide/the-best-vr-virtual-reality-headsets-of-2018

Huitl R., Schroth G., Hilsenbeck S., Schweiger F. & Steinbach E. 9.2012. TUMindoor: an extensive image and point cloud dataset for visual indoor localization and mapping. IEEE International Conference on Image Processing. Julkaistun artikkelin pistepilvitiedostojen lataussivu. Luettu 6.10.2018. http://www.navvis.lmt.ei.tum.de/dataset/

Johansson T. 30.3.2018. Unity at GDC - Job System & Entity Component System. Video. Katsottu 25.10.2018. https://www.youtube.com/watch?v=kwnb9Clh2Is&t=1s

Johansson T. & Ante J. 17.2.2018. JobTempAlloc has allocations that are more than 4 frames old. Foorumikeskustelu. Luettu 25.10.2018 https://forum.unity.com/threads/jobs-lags-jobtempalloc-has-allocations-that-are-more-than-4-frames-old.513124/

Khronos. 6.4.2015 Shader. OpenGL-dokumentaatio. Luettu 10.10.2018. https://www.khronos.org/opengl/wiki/Shader

Khronos. 3.1.2018. Geometry Shader. OpenGL-dokumentaatio. Luettu 10.10.2018. https://www.khronos.org/opengl/wiki/Geometry_Shader

Kulikov P., Wenzel M., Blome M., Latham L. & tompratt-AQ. 20.7.2015. Threading (C#). Microsoftin ohjelmointiopas C#-kielelle. Luettu 20.10.2018. https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/threading/
Arkistoitu: https://github.com/dotnet/docs/blob/886e5c9e7267ffd07f460c8688d104ea639854c1/docs/csharp/programming-guide/concepts/threading/index.md

Luekbe D., Reddy M., Cohen J., Varshney A., Watson B & Huebner R. 2003. Level of Detail For 3D Graphics. Yhdysvallat: Morgan Kaufmann Publishers.

McCaffrey M. 2017. Unreal Engine VR Cookbook. 1. Painos. Yhdysvallat: Addison-Wesley.

Michaelis M. & Lippert E. 2016. Essential C# 6.0. 1. Painos. Yhdysvallat: Addison-Wesley.

Microsoft N.d. a. BinaryReader.ReadBytes(Int32) Method. C#-kielen tekninen dokumentaatio. Luettu 7.10.2018 https://docs.microsoft.com/en-us/dotnet/api/system.io.binaryreader.readbytes?view=netframework-4.7.2#System_IO_BinaryReader_ReadBytes_System_Int32_

Microsoft N.d. b. BinaryWriter Class. C#-kielen tekninen dokumentaatio. Luettu 29.10.2018 https://docs.microsoft.com/en-us/dotnet/api/system.io.binarywriter?view=netframework-4.7.2

Microsoft N.d. c. BitConverter Class. C#-kielen tekninen dokumentaatio. Luettu 9.10.2018 https://docs.microsoft.com/en-us/dotnet/api/system.bitconverter?view=netframework-4.7.2

Microsoft N.d. d. ConcurrentDictionary<TKey,TValue> Class. C#-kielen tekninen dokumentaatio. Luettu 28.10.2018 https://docs.microsoft.com/en-us/dotnet/api/system.collections.concurrent.concurrentdictionary-2?view=netframework-4.7.2

Microsoft N.d. e. Encoding Class. C#-kielen tekninen dokumentaatio. Luettu 29.10.2018 https://docs.microsoft.com/en-us/dotnet/api/system.text.encoding?view=netframework-4.7.2

Microsoft N.d. f. String Class. C#-kielen tekninen dokumentaatio. Luettu 7.10.2018 https://docs.microsoft.com/en-us/dotnet/api/system.string?view=netframework-4.7.2

Microsoft N.d. g. System.Collections.Concurrent Namespace. C#-kielen tekninen dokumentaatio. Luettu 28.10.2018 https://docs.microsoft.com/en-us/dotnet/api/system.collections.concurrent?view=netframework-4.7.2

Nition. 2018. Unity Octree. Valmis octree toteutus Unitylle. Luettu 11.10.2018. https://github.com/Nition/UnityOctree

PCMagazine. N. d. Definition of virtual reality. Verkkolehden sanakirja. Luettu 4.10.2018. https://www.pcmag.com/encyclopedia/term/53945/virtual-reality

Point Cloud Library. 18.11.2018. pcl::PointXYZRGB Struct Reference. Teknillinen dokumentaatio. Luettu 20.11.2018
http://docs.pointclouds.org/trunk/structpcl_1_1_point_x_y_z_r_g_b.html

Point Cloud Library. N.d. a. About. Kuvaus Point Cloud Librarystä. Luettu 6.10.2018 http://pointclouds.org/about/

Point Cloud Library. N.d. b. Spatial Partitioning and Search Operations with Octrees. Tekninen dokumentaatio. Luettu 11.10.2018 http://pointclouds.org/documentation/tutorials/octree.php

Point Cloud Library. N.d. c. The PCD (Point Cloud Data) file format. Tekninen dokumentaatio. Luettu 8.10.2018 http://pointclouds.org/documentation/tutorials/pcd_file_format.php

Schuetz M. 2016. Potree: Rendering Large Point Clouds in Web Browsers. Visual Computing. TU Wien. Diplomi-insinööri opinnäytetyö.

smb02dunnal, Ixpk, Nothke & peteyhayman. 2018. Billboard Geometry Shader. Unity käyttäjien foorumit. Luettu 10.10.2018. https://forum.unity.com/threads/billboard-geometry-shader.169415/

Srinivasiah R. 21.11.2017 How to maximize AR and VR performance with advanced stereo rendering. Unityn blogikirjoitus. Luettu 5.10.2018
https://blogs.unity3d.com/2017/11/21/how-to-maximize-ar-and-vr-performance-with-advanced-stereo-rendering/

Steam. 9.2018. Steam Hardware & Software Survey: September 2018. Valven tekemän laitteistokyselyn tulokset. Luettu 5.10.2018. https://store.steampowered.com/hwsurvey/ Arkistoitu: https://web.archive.org/web/20181005012107/https://store.steampowered.com/hwsurvey/

Thompson S. 1.1.2018. VR Lens Basics: Present And Future. Uutisartikkeli. Luettu 5.10.2018 https://www.tomshardware.com/news/virtual-reality-lens-basicsvr,36182.html

Turk G., Levoy M. 1994. Stanford Bunny, The Stanford 3D Scanning Repository. Stanford Computer Graphics Laboratoryn 3D-mallien lataussivu. Päivitetty 19.8.2014. Luettu 6.10.2018 http://graphics.stanford.edu/data/3Dscanrep/

Unity 2018a. Bounds. Teknillinen dokumentaatio. Luettu 12.10.2018 https://docs.unity3d.com/ScriptReference/Bounds.html

Unity 2018b. Graphics API. Teknillinen dokumentaatio. Luettu 4.10.2018 https://docs.unity3d.com/Manual/GraphicsAPIs.html

Unity 2018c. Mesh.colors32. Teknillinen dokumentaatio. Luettu 18.10.2018 https://docs.unity3d.com/ScriptReference/Mesh-colors32.html

Unity 2018d. Mesh.indexFormat. Teknillinen dokumentaatio. Luettu 9.10.2018 https://docs.unity3d.com/ScriptReference/Mesh-indexFormat.html

Unity 2018e. Mesh.MarkDynamic. Teknillinen dokumentaatio. Luettu 9.10.2018 https://docs.unity3d.com/ScriptReference/Mesh.MarkDynamic.html

Unity 2018f. Mesh.vertices. Teknillinen dokumentaatio. Luettu 17.10.2018 https://docs.unity3d.com/ScriptReference/Mesh-vertices.html

Unity 2018g. MeshRenderer. Teknillinen dokumentaatio. Luettu 9.10.2018 https://docs.unity3d.com/ScriptReference/MeshRenderer.html

Unity 2018h. Physics. Teknillinen dokumentaatio. Luettu 18.10.2018 https://docs.unity3d.com/ScriptReference/Physics.html

Unity 2018i. Plugins. Käyttöohje. Luettu 5.10.2018 https://docs.unity3d.com/Manual/Plugins.html

Unity. 2018j. Real-time solutions. Endless opportunities. Luettu 3.10.2018. https://unity.com/solutions

Unity 2018k. Scripting Runtime Upgrade. Teknillinen dokumentaatio. Luettu 4.10.2018 https://docs.unity3d.com/Manual/ScriptingRuntimeUpgrade.html

Unity 2018l. Single Pass Stereo Rendering. Teknillinen dokumentaatio. Luettu 5.10.2018 https://docs.unity3d.com/Manual/SinglePassStereoRendering.html

Unity. 2018m. The world's leading real-time engine. Luettu 4.10.2018. https://unity3d.com/unity

Unity 2018n Vector3. Teknillinen dokumentaatio. Luettu 9.10.2018 https://docs.unity3d.com/ScriptReference/Vector3.html

Unity 2018o. What is multithreading? Käyttöohje. Luettu 18.10.2018 https://docs.unity3d.com/Manual/JobSystemMultithreading.html

Valve Corporation. 2018. SteamVR Plugin. Unity Asset Store -kaupan sivut. Luettu 5.10.2018. https://assetstore.unity.com/packages/tools/integration/steamvr-plugin-32647

Ward J. 29.4.2008. What is a Game Engine? Artikkeli. Luettu 4.10.2018. http://www.gamecareerguide.com/features/529/what_is_a_game_.php

Wenzel M., Latham L., Petrusha R., tompratt-AQ, Kulikov P., Jones M., Blome M., xaviex. & Maddock C. File and Stream I/O. Microsoftin C#-kielen tekninen dokumentaatio. Luettu 7.10.2018 https://docs.microsoft.com/en-us/dotnet/standard/io/

Wimmer M. & Scheiblauer C. 7.2006. Instant Points: Fast Rendering of Unprocessed PointClouds. Eurographics Symposium on Point-Based Graphics, 129 –136.

Zucconi, A. & Lammers K. 2016. Unity 5.x Shaders and Effects Cookbook. 1.Painos Iso-Britannia, Birmingham: Packt Publishing ltd.