

Pistepilvien visualisointi laitossuunnitteluohjelmistossa

Pro Gradu
Turun yliopisto
Tulevaisuuden teknologioiden laitos
Tietojenkäsittelytiede
2018
Timo Heinonen
Tarkastajat:
P.P.
H.H.

Turun yliopiston laatujärjestelmän mukaisesti tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck-järjestelmällä

TURUN YLIOPISTO

Tulevaisuden teknologioiden laitos

Timo Heinonen Pistepilvien visualisointi laitossuunnitteluoohjelmistossa

Pro Gradu, 53 s., 3 liites.

Tietojenkäsittelytiede

5. tammikuuta 2020

Turun yliopiston laatujärjestelmän mukaisesti tämän julkaisun alkuperäisyys on tarkastettu Turnitin Originality Check -järjestelmällä.

Pistepilvet ovat useimmiten laserkeilaimilla mitattuja malleja jostakin tosimailman esineestä tai maisemasta. Monimutkaisen geometrian sijaan kohdetta kuvataan valtavalla määrellä pisteitä, jotka visualisoimalla saadaan kuva yhtenäisitä pinnoista. Laserkeilaimella taltioituja pistepilviä käytetään esimerkiksi arkkitehtuurissa, rakentamisessa ja kulttuajutteiden suunnittelussa. Tässä tutkielmassa keskitytään pistepilvien käyttöön laitossuunnitteluoohjelmistoissa, jossa käyttäjä haluaa katsella suurista laitoksista keilattuja pistepilviä ja mallintaa geometriaa niiden avulla.

Pistepilvien massiivinen koko aiheuttaa ongelmia niitä käsitteleville ja visualisoiville ohjelmistoille. Suuret pistepilvet eivät mahdu kerralla tietokoneen keskusmuistiin ja niiden visualisoiminen kestää kauan. Usein pistepilvi tallennetaan hierarkiseen tietorakenteeseen, joka mahdollistaa sen asteittaisen lataamisen kiintolevyltä ja pisteiden määrä vähentäviä tarkkuustasoja, joilla saadaan aikaan interaktiivinen ruuudunpäivitystaajuus.

Tässä tutkielmassa perehdytään pistepilvien visualisoinnissa käytettyihin hierarkisiin tietorakenteisiin ja todetaan niin kutsuttujen sisäkkäispistepuiden soveltuvan laitossuunnitteluoohjelmistossa käytettäväksi. Lisäksi esitellään niin sisäkkäispistepuille yksinkertainen kompressiotekniikka ja renderöintialgoritmi.

Sisältö

1 Johdanto	1
1.1 Pistepilvet	1
1.2 Laserkeilaimet	4
1.3 Pistepilvidatan käsittey	6
1.3.1 Panoramakuvat	7
1.3.2 Rekisteröinti	7
1.3.3 Pintojen rekonstruointi	9
1.3.4 Normaalivektorien löytäminen	10
1.4 Pistepilvien hyödyntäminen tietokoneavusteisessa suunnittelussa	11
1.5 Tutkielman tavoitteet ja rakenne	12
2 Pistepilvien visualisointi	14
2.1 Pistepilvien visualisoinnin haasteet	14
2.2 Hierarkiset tietorakenteet	15
2.2.1 Qsplat	15
2.2.2 Peräkkäispistepuut	18
2.2.3 Sisäkkäispistepuut	19
2.2.4 kd-puut	23
2.3 Ei-hierarkiset tekniikat	23
3 Laitossuunnitteluohjelmistoon optimoitu tietorakenne	25
3.1 Käyttötapaukset ja vaatimukset tietorakenteelle	25
3.1.1 Mallintaminen	25
3.1.2 Mittaaminen	26
3.1.3 Katselu	26
3.2 Tietorakenteen valinta	27
3.3 Pistedatan esitysmuoto	28

3.4	Tietorakenteen rakentaminen	32
3.5	Renderöinti	36
3.6	Pisteiden valitseminen	40
4	Tietorakenteen arviointi	42
4.1	Tietorakenteen rakentaminen ja lataaminen	42
4.2	Renderöinti	43
4.3	Pohdintaa	48
5	Yhteenvetö	50
	Viitteet	50

1 Johdanto

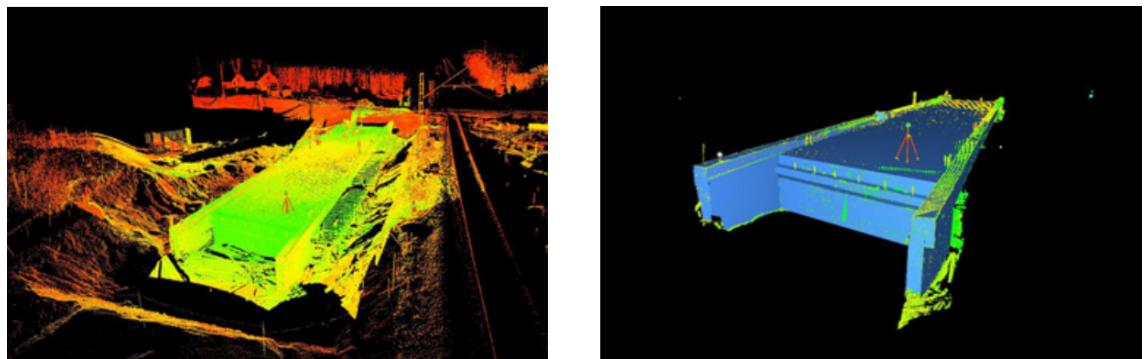
Tässä luvussa määritellään pistepilven käsite, tutustutaan laserkeilaimiin, joilla pistepilviä taltioidaan ja mainitaan muutamia yleisiä pistepilvien käsittelyn työvaiheita. Lisäksi esitellään muutama esimerkki pistepilvien käyttökohteista. Lopuksi määritellään tämän tutkielman tavoitteet, toteutustapa ja rakenne.

1.1 Pistepilvet

Pistepilveksi kutsutaan jotakin objektiota tai maisemaa kuvaavaa suurta joukkoa pisteitä kolmiulotteisessa avaruudessa. Pistepilvi tuotetaan yleensä laserkeilaimella (engl. *laser scanner*), joka ampuu ympärilleen laserpurskeita ja mittaa etäisyksiä pisteisiin, joista purske heijastuu takaisin. Pistepilviä voidaan tuottaa myös synteettisesti ottamalla näytepisteitä mistä tahansa 3d-mallista, mutta tässä tutkielmassa keskitytään laserkeilaimilla tuottuihin pistepilviin.¹

Pistepilville on useita käyttökohteita, joista arkkitehtuuri ja rakentaminen on yksi tärkeimmistä. Pistepilvien hyödyntäminen voidaan aloittaa rakennusprojektiin hyvin varhaisessa vaiheessa. Rakennettavan tontin ympäristöstä voidaan ottaa laserkeilauska, jotta suunniteltavan rakennuksen sopimista tontille voidaan helposti arvioida. Rakennusprojektiin aikana säädöllisillä laserkeilauskilla voidaan seurata tarkasti projektin etenemistä ja havaita mahdollisia ongelmia ajoissa. Pistepilvillä on myös tärkeä rooli rakennuksen valmistumisen jälkeen. Muutostöitä tehtäessä halutaan rakennuksesta saada ajantasalla oleva 3d-malli. Manuaalinen mallintaminen olisi hyvin suuri työ verrattuna muutaman kymmenen pistepilven luomiseen laserkeilaimella, mikä voidaan tehdä päivässä. [2]

¹1980-luvulta lähtien pisteitä on ehdotettu yleisiksi renderöintiprimitiiveiksi kuvaamaan mitä tahansa geometriaa [1]. Ajatus on nykyään vielä ajankohtaisempi, sillä renderöitäävät mallit monimutkaistuvat jatkuvasti ja usein yksittäiset polygonit projisoituvat kuvaruudulle alle pikselin kokoisina. Tällöin polygonien kärkipisteiden sijaan olisi tehokkaampaa säilyttää muistissa vain yhtä pistettä. Grafiikkakirjastot ja näytönohjaimet on kuitenkin vielä optimoitu kolmioiden käsittelyyn.



Kuva 1: Silta Joroisten ja Varkauden välissä Kuvasintiellä. Vasemmalla pistepilvi, oikealla pistepilvestä muodostetua geometriaa suunnitteluoohjelmassa. Kuva: [3]

Eräs mielenkiintoinen sovelluuskohde laserkeilaukselle on siltojen rakentaminen. Sillanrakennuksessa haasteita tuottavat tien ja maaston geometrian yhtenosovittamisen lisäksi projektin pitkä kesto. Silta on rakennettava osissa ja lopputuloksen on oltava suora, minkä takia siltatyömaalla suoritetaan usein tarkistusmittauksia. Älykäs silta -projektissa tutkittiin tapoja hyödyntää tietotekniikkaa sillanrakentamisessa ja -korjaamisessa, ja havaittiin laserkeilauksen olevan hyvä tapa suorittaa tarkkuusmittauksia. Laserkeilauksella muodostettiin pistepilviä sillan kannesta ja rakenteista, jonka jälkeen niistä muodostettiin pintoja 3d-suunnitteluoohjelmaan. Pistepilvi ja siitä muodostettu geometria on esitetty kuvassa 1. [3]

Pistepilviä käytetään hyväksi myös arkeologiassa. Roomalaiset perustivat vuoden 40 tienoilla Tonavan varrelle nykyisen Itävallan alueelle sotilasleirin, josta kasvoi myöhemmin Carnuntumin kaupunki. Kaupungin muurien ulkopuolelle rakennettiin amfiteatteri, johon mahtui 13000 katsojaa. Vuonna 2007 Ala-Itävallan osavaltion hallitus aloitti arkeologiset kaivaukset amfiteatterin alueella, joiden yhteydessä alueesta muodostettiin kattava pistepilvi noin kahdella sadalla laserkeilauksella. Ruudunkaappaus kysesestä pistepilvestä on esitetty kuvassa 2. [5]

Pistepilviä käytetään historiallisen kulttuuriperinnön säilyttämiseen myös pienemmäsä mittakaavassa. 1800-luvulla tehdyissä Cerveterin arkeologisissa kaivauksissa Italiassa löydettiin 500-luvulla terrakottasavesta tehty etruskilainen sarkofagi, joka kuvasi avio-

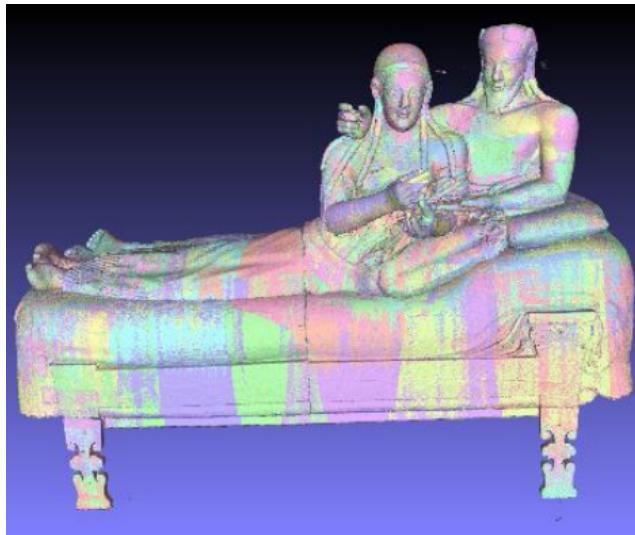


Kuva 2: Carnuntumin kaupungin amfiteatteri taltioituna laserkeilausella. Kuva: [4]

paria rentoutumassa tuonpuoleisessa. Satoihin palasiin hajonnut sarkofagi restauroitiin vuonna 1893 ja se digitoitiin käyttämällä laserkeilausta ja fotogrammetriaa vuonna 2013 taidenäytelyä varten. Sarkofagista keilattu pistepilvi on esitetty kuvassa 3. [6]

Eräs vaativa pistepilvien sovelluskohde on itseohjautuvat kulkuneuvot. Voidakseen navigoida liikenteessä itseohjautuva auto tarvitsee kameroiden ja ultraäänisensoreiden lisäksi katollen laserkeilaimen, jolla voidaan tarkkailla auton etäisyyttä muihin tienväyläjiin ja esteisiin. Itseohjautuvat autot ovat merkittävä tutkimuskohde myös pistepilvien käsitelyn kannalta. Auton katolle asennettavan laserkeilaimen tulisi olla tarkka, nopea ja edullinen, ja sen tuottamaa pistedataa täytyy voida käsitellä reaalialjassa. [7]

Pistepilviä voidaan käyttää myös huomattavasti suuremmassa mittakaavassa. Maanmittauslaitos on kerännyt ilmasta käsin pistepilvidataa lähes koko Suomen maaperästä. Lentokoneesta keilattu pistepilvi on melko harva - puoli pistettä neliömetriä kohden -, mutta keilattavan kohteen laajuus tekee pilvistä valtavia. Pistepilviä on käytetty lähinnä metsävarojen kartoittamiseen, mutta Maanmittauslaitos aikoo ryhtyä keräämään pistedataa tarkemmillä laserkeilamilla myös rakennuksista. [8]



Kuva 3: Etruskilaisesta sarkofagista muodostettu pistepilvi. Kuva: [6]

Laserkeilauksen kohteena ei ole aina maasto tai eloton rakennelma, vaan myös ihmisiä voidaan tehdä pistepilviä. Esimerkiksi moottoripyöräkypärien suunnittelija voi käyttää hyväkseen ihmisen päättä kuvaavia pistepilviä selvittääkseen sopivan muodon kypärän sisukselle. Kypärän ulkokuori voidaan sen jälkeen mallintaa suunnitteluoohjelmistolla niin, että se täytyy tarvittavat turvallisuusvaatimukset. Pistepilvillä on paikkansa myös lääketieteessä. Proteesin valmistaja voi esimerkiksi käyttää potilaan terveestä jalasta mitattua pistepilveä suunnitellessaan proteesia, jotta se muistuttaisi mahdollisimman paljon amputoitua jalkaa. Myös kirurgit voivat käyttää leikkauksen suunnittelussa hyväkseen elimistä laserkeilauksella muodostettuja 3d-malleja. [9]

1.2 Laserkeilaimet

Perinteinen pistepilven mittaamiseen käytetty laserkeilain on jalustalla seisova laite, joka pyörii pystyakselinsa ympäri ampuen ympärilleen miljoonia laserpurskeita. Laserkeilain mittaa etäisyksiä pisteisiin, joista laserpurske heijastuu takaisin keilaimeen ja muodostaa näistä pisteistä pistepilven. Heijastuksen voimakkuutta käytetään usein määräämään pisteeelle väri. Yleensä tarkasteltavaa kohdetta täytyy keilata useista eri suunnista, jotta saatasiin tarpeeksi kattava joukko pistepilviä. Kohteesta riippuen voidaan tarvita jopa satoja



Kuva 4: Kulkuaikatekniikkaan perustuva Leica RTC360 -laserkeilain. Kuva: [10]

keilauksia. Pistepilvien sovittamista yhteen tarkastellaan luvussa 1.3.

Nykyaikaisella laserkeilaimella saadaan muodostettua hyvin tarkka ja tiheä pistepilvi nopeasti. Esimerkiksi kuvassa 4 näkyvä Leica Geosystems RTC360 -keilaimen luvataan mittaamaavan jopa kaksi miljoonaa pistettä sekunnissa ja kiertävän täyden ympyrän alle kahdessa minuutissa. Keilaimen lisäksi laitteessa on kamera, jolla saadaan määritettyä pisteille oikeat värit. [11]

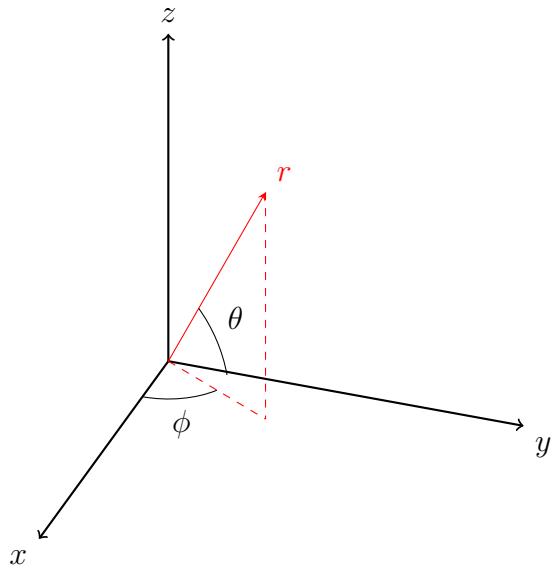
Laserkeilaimien toimintaperiaatteissa on eroja. Kaksi yleisintä toimintaperiaatteita ovat kulkuaikatekniikka (engl. *Time-Of-Flight*) ja vaihesiirtotekniikka (engl. *phase shift*). Kulkuaikatekniikkassa pisteen etäisyys keilaimesta selviää ajasta, joka kuluu laserpurskeen lähetettämisenstä sen heijastuksen vastaanottamiseen. Pisten etäisyys keilaimesta on yksinkertaisesti

$$d = \frac{c \cdot t}{2}, \quad (1)$$

missä c on valonnopeus ja t on mitattu aika. [12] Vaihesiirtotekniikka perustuu keilaimesta lähtevän signaalin vaiheen vertaamista palaavan signaalin vaiheeseen. Pisten etäisyys keilaimesta saadaan laskemalla

$$d = n \cdot \lambda + \frac{\Phi \cdot \lambda}{2 \cdot \pi}, \quad (2)$$

missä n on havainnon täysien aaltojen määrä, λ on signaalin aallonpituuus ja Φ on lähtevän



Kuva 5: Pallokoordinaatisto. Pisteen sijainti avaruudessa ilmaistaan korotuskulmalla θ , atsimuuttikulmalla ϕ ja säteellä r .

ja palaavan signaalin vaihe-ero. [12]

Laserkeilain tallentaa mittaamansa pisteen pallokoordinaateissa. Pisteen siirtäminen pallokoordinaateista karteesiseen koordinaatistoon onnistuu laskemalla koordinaatit

$$\begin{aligned} x &= r \cdot \sin \theta \cdot \cos \phi, \\ y &= r \cdot \sin \theta \cdot \sin \phi, \\ z &= r \cdot \cos \theta, \end{aligned} \tag{3}$$

missä r on pallon säde, θ on korotuskulma ja ϕ atsimuuttikulma. Pallokoordinaatista on havainnollistettu kuvassa 5.

1.3 Pistepilvidatan käsittely

Laserkeilaussella tuotettuja pistepilviä ei usein käytetä sellaisenaan, vaan niitä täytyy ensin esikäsittellä. Yleisiä esikäsittelyvaiheita ovat panoramakuviien luominen, usean pistepilven rekisteröinti yhteen koordinaatistoon, muukalaispisteiden poistaminen ja pintojen rekonstruointi.



Kuva 6: Saksan Würzburgissa sijaitsevasta Randersackerin Neitsyt Marian kärsimysten kappelista keilatusta pistepilvestä muodostettu panoramakuva.

1.3.1 Panoramakuvat

Yksinkertaisin tapa visualisoida pistepilvi on muodostaa siitä kaksiulotteinen kuva. Pilvestä voidaan luoda panoramakuva projisoimalla laserkeilaimen ympäröimät pisteet kaksiulotteiselle kuvatasolle.² Kuvakoosta riippuen voidaan pistepilvestä karsia huomattava määrä pisteitä, joiden koko olisi liian pieni kuvatasolle projisoituna. Panoramakuvat toimivat siis myös pistedatan pakkausalgoritmina. Panoramakuvan pikseleille voidaan antaa värit joko sen perusteella, kuinka paljon valoa heijastuu takaisin niitä vastaavista pilven pisteistä, tai kuinka kaukana pisteet ovat keilaimesta. Panoramakuvaan on helppo soveltaa erilaisia kuvankäsittelyalgoritmeja, kuten piirteentunnistusta (engl. *feature detection*).

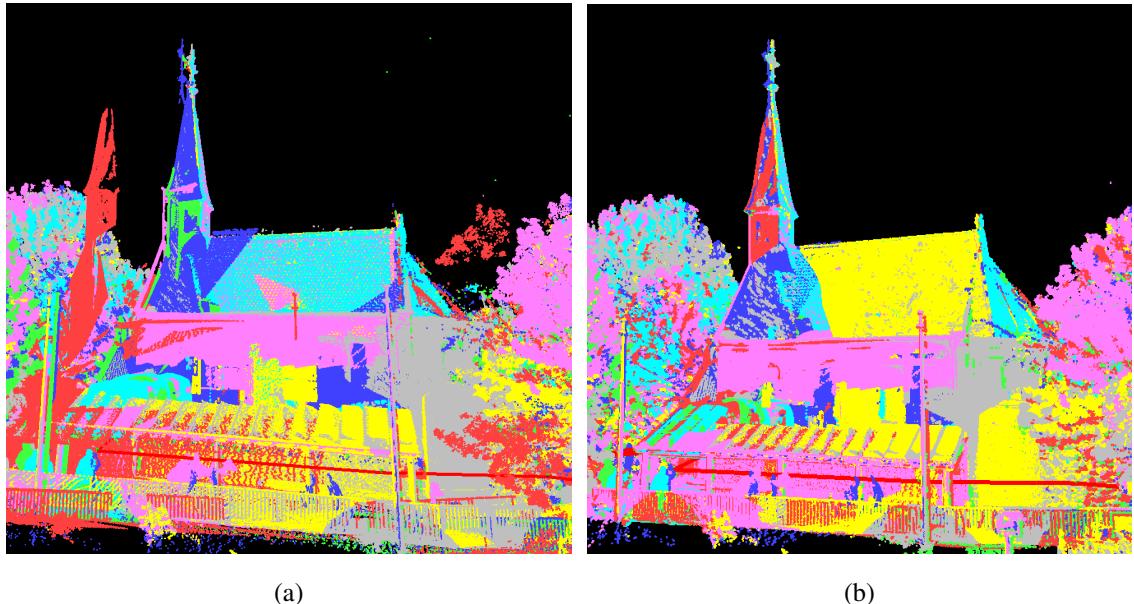
Kuvassa 6 on esimerkki panoramakuvasta.

1.3.2 Rekisteröinti

Laserkeilauksen tuottama pistepilvi sisältää joukon pisteitä koordinaatistossa, jonka origona on keilaimen sijainti. Usein keilauksen kohteesta otetaan kymmeniä tai jopa satoja keilauksia, jotka täytyy saada samaan koordinaatistoon. Tätä kutsutaan pistepilvien rekisteröinniksi.

Pistepilviä voidaan rekisteröidä usealla eri tavalla. Joskus keilattavaan kohteeseen ase-

²Projektiot vaikuttavat suuresti panoramakuvan laatuun. Hyvän yleiskatsauksen erilaisiin projektioihin antaa [13]. Kuvassa 6 on käytetty tasavälistä lieriöprojektiota (engl. *equirectangular projection*).



Kuva 7: Kuvan 6 kappelista keilaittuja pistepilviä (a) ennen rekisteröintiä ja (b) rekisteröinnin jälkeen. Kunkin keilaksen pisteet on piirretty eri väreillä.

tetaan erityisiä merkkikuvioita, jotka näkyvät useasta keilaimesta. Kun tiedetään merkkien etäisyys ja suunta kustakin keilaimesta, voidaan pistepilvet sovittaa helposti yhteen koordinaatistoon. Joissakin sovelluksissa käyttäjä merkitsee kahdesta pilvestä joukon pisteitä, joiden avulla pilvet saadaan rekisteröityä. Esimerkiksi 3DTK-kirjastolla käyttäjä voi valita kahdesta pilvestä samaa aluetta kuvaavia pisteitä, joiden avulla pilvien yhteensovitus onnistuu automaattisesti [14].

Pistepilviä voidaan rekisteroidä myös ilman merkkikuvioita tai käyttäjän apua. Iteratiivinen lähimmän pisteiden algoritmi (engl. *iterative closest point, ICP*) sovittaa pistepilven toiseen etsimällä rotaation ja translaation, jolla pilvien välinen virhe saadaan minimoitua. Virheen laskemista varten täytyy määrittää pilvistä toisiaan vastaavat pisteet. Yksinkertaisimillaan pistettä vastaavaksi pisteeksi valitaan toisesta pilvestä se, jonka euklidinen etäisyys edellä mainittuun on pienin. Iteratiivisen algoritmin kunkin transformaation hyväys lasketaan kaikkien vastaavien pisteiden välisten etäisyyksien muodostamasta virheestä. Algoritmin iteroiminen voidaan lopettaa, kun jokin ennaltamääritetty raja virheelle

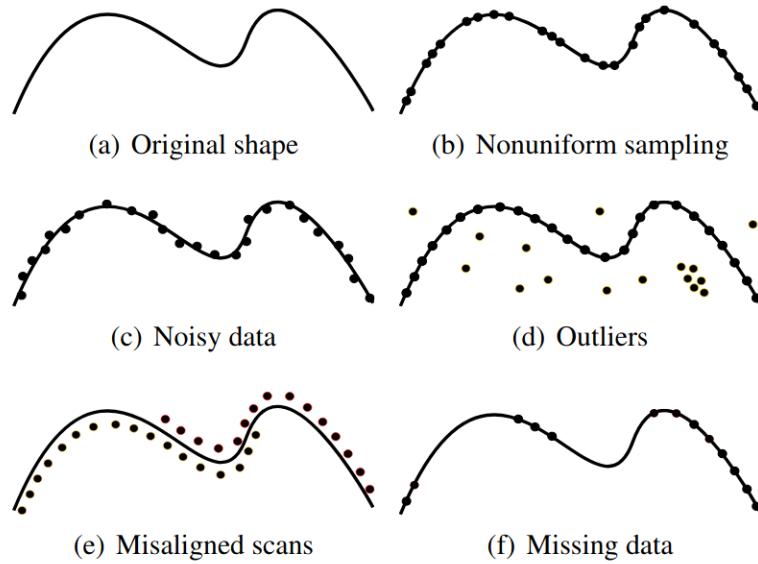
on alitettu. Kuvassa 7 on ICP-algoritmilla sovitettu yhteen kuusi pistepilveä. [15]

ICP-algoritmi tarvitsee kuitenkin käyttäjältä hyvän alkuarvauksen, jotta virhe suppeisi ja pilvien sovittaminen onnistuisi. Se on myös erittäin raskas suorittaa isoille pistepilville. Yleensä ICP-algoritmia käytetäänkin karkeamman rekisteröintialgoritmin jälkeen hienosäättämiseen. Karkeampi pilvien yhteensovittaminen tehdään yleensä etsimällä kahdesta pistepilvestä muodostetuista panoramakuivista toisiaan vastaavia avainpisteitä (engl. *keypoints*). Näiden avainpisteiden sijainneista saadaan harva joukko pisteitä, joiden yhteensovittaminen on paljon helpompaa kuin kokonaisten pistepilvien. Suosittuja teknikoita avainpisteiden löytämiseen ovat esimerkiksi Harrisin nurkat [16] sekä FAST ja SIFT -piirteet [17][18]. [19]

1.3.3 Pintojen rekonstruointi

Joissakin sovelluksissa halutaan luoda pistedataasta kohteen pintoja kuvaava polygoniverkko, jotta renderöinti olisi nopeampaa olemassaolevilla grafiikkakirjastoilla ja visuaalinen lopputulos parempi. Yksinkertainen tekniikka luoda tiivis kolmiointi pistepilvestä on Delaunayn kolmiointi. Delaunayn kolmiointi perustuu Voronoin diagrammiin (engl. *Voronoi diagram*), joka jakaa pisteitä sisältävän tason tai avaruuden konvekseihin Voronoin soluihin. Voronoin solu kattaa sen alueen, jossa etäisyys solua vastaavaan pisteeseen on pienempi kuin muihin pisteisiin. Kahden solun välillä on Voronoin jana, josta etäisyys kahteen pisteeseen on sama, ja kolmen janan leikkauspisteessä on Voronoin kärki, josta etäisyys kolmeen pisteeseen on yhtä suuri. Delaunayn kolmiointi on Voronoin diagrammin duaaligraafi, josta luodaan graafi siten, että pisteen välillä on kaari, mikäli niitä vastaavat Voronoin solut jakavat Voronoin janan. [20]

Oikeat, laserkeilaimella taltioidut pistepilvet sisältävät kuitenkin usein erilaisia virheitä, kuten kuvassa 8 esitellyt epätasainen keilaus, keilaimen häiriö, muukalaispisteet (engl. *outlier*), ongelmat rekisteröinnissä ja puuttuva data. Virheiden johdosta Delaunayn kolmiointi ei ole kovin tehokas algoritmi. Useissa pintojen rekonstruointialgoritmeissa teh-



Kuva 8: Kaarevasta pinnasta (a) keilatussa pistepilvissä esiintyviä mahdollisia virheitä: (b) epätasainen pistetihesys, (c) häiriötä pisteiden sijainneissa, (d) muukalaispisteitä, (e) epätarkka rekisteröinti ja (f) puuttuvaa dataa. Kuva: [21]

dään joitakin oletuksia pistepilven ominaisuuksista. Etenkin tietokoneavusteisessa suunnittelussa keilauksen kohteet muodostuvat yksinkertaisista geometrisista primitiiveistä, kuten tasoiista ja sylinteriistä. Esimerkiksi RANSAC-algoritmi [22] sovittelee primitiivejä satunnaistetusti pistepilveen ja arvioi niiden sopivuutta. [21]

1.3.4 Normaalivektorien löytäminen

Joskus on hyödyllistä tietää pistepilven esittämien pintojen normaalivektorit. Tasainen pinta on helppo sijoittaa pistepilven päälle, jos jollakin alueella on tiheästi pisteitä, joiden normaalivektorit osoittavat samaan suuntaan. Laserkeilaimen epätarkkuudesta tai vaikka pa tuulen heiluttamista puiden lehdistä johtuen on pistepilvissä usein muukalaispisteitä. Yksinkertainen tapa havaita ja poistaa muukalaispisteitä on vertailla pisteiden normaalivektoreita niiden naapuruston normaaleihin ja etsiä poikkeavuuksia.

Pisteen p_i normaalivektori voidaan selvittää pääkomponenttianalyysillä (engl. *principal component analysis, PCA*). Ensin on etsittävä pilvestä pisteen p_i k lähintä naapuria,

jonka jälkeen naapuriston pisteistä lasketaan ominaisarvot. Kahta suurinta ominaisarvoa vastaavaa ominaisvektoria voidaan käyttää kuvaamaan tasoa, joka sovitetaan naapuriston päälle. Jäljelle jäävä ominaisvektori kuvaaa pisteen p_i normaalialia. [23]

1.4 Pistepilvien hyödyntäminen tietokoneavusteisessa suunnittelussa

Aiemmin mainittiin erilaisia sovelluksia laserkeilainten tuottamille pistepilville. Tässä tutkielmassa keskitytään pistepilvien hyödyntämiseen tietokoneavusteisessa suunnittelussa (engl. *computer aided design, CAD*) ja erityisesti laitossuunnitteluoohjelmistoissa (engl. *plant design software*).

Tietokoneavusteisessa suunnittelussa pistepilviä käytetään olemassaolevien rakenteiden taltiointiin. Usein käytetty esimerkki on autoteollisuuden alalta: ryhmä suunnittelijoita kokeilee uutta korimallia rakentamalla prototyypiauton helposti muovattavasta materiaalista. Kun prototyppi on todettu aerodynaamiseksi ja miellyttävän näköiseksi, täytyy se saada digitoitua, jotta se voidaan siirtää massatuotantoon. Usein yksinkertaisin ja kustannustehokkain tapa on laserkeilata prototyppi ja jatkokäsitellä pistepilveä niin, että saadaan luotua haluttu 3d-malli.

Laitossuunnittelussa yleisempi ongelma on 3d-mallin vanhentuminen tai sen puuttuminen kokonaan. Vaikka laitosta alunperin suunniteltaessa siitä olisi tehty 3d-malli, laitteistojen sommittelua saatetaan muuttaa ilman, että samoja muutoksia tehdään 3d-malliin. Kun 3d-mallia halutaan taas hyödyntää, voi olla kustannustehokkaampaa luoda laitoksesta laserkeilaimella pistepilvi kuin mallintaa tehdyt muutokset suunnitteluoohjelmalla. [24]

Kuten sillanrakennuksessa, myös laitoksia rakentaessa on joskus syytä suorittaa tarkeusmittauksia ja verrata niitä alkuperäiseen 3d-malliin. Pistepilvet ovat tähänkin tarkeukseen sopivia. Nykyaikaisten laserkeilainten tuottamat pistepilvet ovat niin tarkkoja, että niistä voi havaita esimerkiksi putkien roikkumisen ja lämpölaajenemisen [24].

Pistepilviä voi hyödyntää monin tavoin laitossuunnittelussa. Jos laitokseen halutaan vaikkapa uusi vesiputki, voidaan sen mahtuminen varmistaa reitittämällä putki suunnit-

teluohjelmistolla ja tarkastamalla, osuko se pistepilveen. Jos taas vanhasta laitoksesta halutaan luoda ajantasalla oleva 3d-malli, voi suunnittelija mallintaa laitosta pistepilvien avulla. Pistepilven päälle on helppo sijoittaa suunnitteluoohjelman putkistoja ja laitteita oikeille paikoilleen. Markkinoilla on myös ohjelmistoja, joiden luvataan tuottavan pistepilvestä automaattisesti älykäs 3d-malli komponenttitietoineen ilman aikaavievää päälle-mallinnusta [25].

Tietokoneavusteisen suunnittelun ohjelmisto käsittelee pistepilviä eri tavoin riippuen suunnittelutyön luonteesta. Kappalemallinnuksessa visualisoidaan usein yksittäisiä osia, joita kuvaavat pistepilvet ovat hyvin tiiviitä ja usein kaikki pisteet ovat kerralla näkyvissä. Laitossuunnittelussa pistepilvet ovat taas hyvin laajoja ja voivat kuvata esimerkiksi koko-naisista tehdasalueita. Näissä sovelluksissa on tärkeää pistepilven nopea rajaaminen niin, että vain näkymän sisältävät pisteet renderöidään. Usein on tarpeen myös käyttää eri tarkkuksia pistepilven eri osille; kaukana katsojasta sijaitsevista pisteistä renderöidään vain osa [26]. Joitakin yleisiä käyttötapauksia pistepilvien kanssa työskentelystä laitossuunnitelussa ja niiden esittämiä vaatimuksia laitossuunnitteluoohjelmistolle on esitetty luvussa 3.1.

1.5 Tutkielman tavoitteet ja rakenne

Tämän tutkielman tarkoituksesta on tutustua pistepilvien käsittelyn ja visualisoinnin tuotamiin ongelmiin ja selvittää niihin ratkaisuvaihtoehtoja alan julkaisujen pohjalta. Selvitystyön pohjalta kehitetään tietorakenne erityisesti laitossuunnitteluoohjelmiston tarpeisiin. Lopuksi arvioidaan tietorakenteen suorituskykyä ja soveltuvuutta tehtävään käytäen verrokkina yksinkertaista ei-hierarkista tietorakennetta.

Tässä luvussa on perehdytty pistepilviin ja laserkeilaimiin yleisellä tasolla. Luvussa 2 tutustutaan alan kirjallisuuteen ja esitellään tietorakenteita, joiden ominaisuuksia voisi hyödyntää myös laitossuunnitteluoohjelmistossa. Luvussa 3 määritellään vaatimuksia, joita laitossuunnitteluoohjelmisto asettaa pistepilvien käsittelyyn ja visualisointiin käytetylle

tietorakenteelle. Tämän jälkeen ehdotetaan luvussa 2 esiteltyihin julkaisuihin perustuva, laitossuunnitteluoohjelmistoon soveltuva tietorakenne ja algoritmeja. Näiden algoritmien suorituskykyä arvioidaan luvussa luvussa 4.

Tämä tutkielma on tehty CADMATIC Oy:n toimeksiantona. CADMATIC on suomalainen ohjelmistoyritys, joka kehittää tuotteita laivojen ja laitosten tietokoneavustiseen suunnittelun. CADMATIC on toiminut alalla 1980-luvulta lähtien ja sillä on asiakkainaan yli tuhat organisaatiota yli viidestäkymmenestä maasta [27]. Yrityksen pääkonttori on Turussa. Tutkielmassa kehitettävää tietorakennetta testataan CADMATIC Plant Modeler -laitossuunnitteluoohjelmistossa, sekä eBrowser-mallinkatseluohjelmistossa.

2 Pistepilvien visualisointi

Tässä luvussa selvitetään, mitä haasteita pistepilvien ominaisuudet asettavat visualisoinnissa käytetylle tietorakenteille ja algoritmeille. Lisäksi tutustutaan alan julkaisuihin, joilla haasteisiin pyritään vastaamaan.

2.1 Pistepilvien visualisoinnin haasteet

Suurin haaste pistepilvien käsitteilyssä ja visualisoinnissa on niiden koko. Nykyainainen laserkeilain, kuten luvussa 1.2 esitety Leica Geosystemsin RTC360, tuottaa pistepilven, jossa on satoja miljoonia pisteitä. Kun tällaisella keilaimella tehdään useita keilauksia, on pisteiden määrä valtava. Oletetaan esimerkiksi, että suressa projektissa käytetään pistepilviä, joissa on yhteensä miljardi pistettä. Kun koordinaatit tallennetaan kolmella neli-tavuisella liukuluvulla ja värit RGB-muodossa kolmella tavulla ja lisätään perään vielä yksi täytetavu, voidaan yksi pilven piste esittää 16:lla tavulla. Miljardin pisteen pilvi olisi siis kooltaan 16 gigatavua mahdollisen metatatan lisäksi. Tämän kokoista pilveä ei haluta pitää kerralla keskusmuistissa, vaan pisteitä haetaan levyltä muistiin vain tarvittaessa.

Pisteiden tallentaminen levylle aiheuttaa kuitenkin ongelmia tiedonsiirron hitauden takia, sillä tiedonsiirtonopeus kiintolevyltä on jopa kakso kertaluokkaa hitaampi kuin keskusmuistista. Ongelman ratkaisemiseksi käytetään usein ulkoisen muiston algoritmeja (engl. *out-of-core algorithm*), jotka lataavat pisteitä levyltä muistiin isoissa palasissa, mikä on huomattavasti nopeampaa kuin yksittäisten pisteiden lataaminen. [28]

Pistepilvien koon vuoksi ei ole realistista olettaa, että kaikki pisteet voitaisiin renderöidä reaalialajassa.³ Tästä syystä on kehitetty erilaisia tekniikoita pistepilven harventamiseen ja osittaiseen renderöintiin. Yksinkertainen tapa harventaa pistepilveä on jakaa sen peittämä alue niin kutsuttuihin vokseleihin (engl. *volume element, voxel*), eli säädöllisiin kuutioihin, ja visualisoimalla vain yksi piste kustakin vokselista. Pilven harventaminen

³Miellyttävän käyttökokemuksen takaamiseksi pitäisi ruutu päivittää vähintään kymmenen kertaa sekunnissa.

kuitenkin aiheuttaa yksityiskohtien katoamista pilvestä, joten sitä täytyy käyttää sovelluskohteesta riippuen matallisesti.

Toinen keino vähentää visualisoitavien pisteen määrää on määrittää pistepilvelle tarkkuustasot (engl. *level-of-detail, LOD*). Tarkkuustasot mahdollistavat pistepilven astettaisen tarkentamisen karkeasta yleiskuvasta yksityiskohtiin. Usein interaktiivisissa ohjelmistoissa korkean ruudunpäivitystaajuuden ylläpitäminen vaatii sitä, että pistepilvi renderöidään matalimmalla tarkkuudella esimerkiksi käyttäjän pyörittäessä näkymää. Toisaalta kun kamera pysähtyy paikalleen, voidaan renderöintiin käyttää enemmän aikaa ja pistepilveä tarkentaa.

2.2 Hierarkiset tietorakenteet

Edellä mainittuihin haasteisiin on otettu kantaa laajalti alan julkaisuissa. Lupaavin teknikka tarkkuustasojen muodostamiseen, ulkoisen muiston käyttöön ja pistepilvien harventamiseen näyttää olevan pistepilven jakaminen hierarkiseen tietorakenteeseen. Ajatuksena on, ettei kaikkia pisteitä tarvitse käydä läpi jokaisella ruudunpäivityksellä, vaan hierarkian yläpäässä on karkein tarkkuustaso ja alaspäin kulkissa tarkkuus kasvaa.

Esitellään seuraavaksi muutama kiinnostava hierarkinen tietorakenne. Aihealueen pioneeriyöä pidetään Rusinkiewiczin ja Levoyn Qsplatia, joka onkin antanut vaikuttavia uudemmille tietorakenteille. Dachsbacher et al. esittelivät peräkkäispistepuit, jotka voidaan renderoida hyvin nopeasti näytönohjaimella. Viime vuosina pistepilvien visualisoinnin tutkimuksen kirkkainta kärkeä on edustanut Wienin teknillisen yliopiston tietokonegrafiikan tutkimusyksikkö. Tämän tutkielman päälähteinä käytetään Claus Scheiblauerin ja Markus Schützin julkaisuja sisäkkäispistepuista.

2.2.1 Qsplat

Yksi ensimmäisistä pistedatan visualointiin käytetyistä hierarkisista tietorakenteista on Rusinkiewiczin ja Levoyn esittämä QSplat, joka on kehitetty polygoniverkon visualisoin-

tiin pisteiden avulla. Tietorakenne muodostetaan kolmiodusta mallista, jossa kolmioiden normaalit tunnetaan, joten se ei suoraan sovella raa'an pistepilvidatan käsittelyyn.⁴ QSplatissa on käytetty kuitenkin monia kiinnostavia tekniikoita, joita voi hyödyntää pistepilvien käsittelyssä. [29]

QSplat perustuu puurakenteeseen, jonka solmuissa on rajauspalloja (engl. *bounding sphere*). Pallot jakavat avaruutta rekursiivisesti pienempiin osiin siten, että juuren pallo sisältää kaikki kolmiot ja jokainen sisäinen solmu jakaa avaruuden keskimäärin neljään osaan. Puun latva saavutetaan, kun avaruuden jakamisen seurauksena jäljelle jää yksi kolmio. Siitä muodostetaan lehtisolmu, jonka rajauspallo sisältää koko kolmion. Puun visualisointi onnistuu piirtämällä jokaisen pallon kohdalle sopivan kokoinen täplä (engl. *splat*). Puurakenne mahdollistaa myös tehokkaan pisteiden karsimisen. Jos solmun pallo ei ole näkökentässä, eivät sen lapsetkaan ole ja haaraa ei tarvitse käydä läpi. [29]

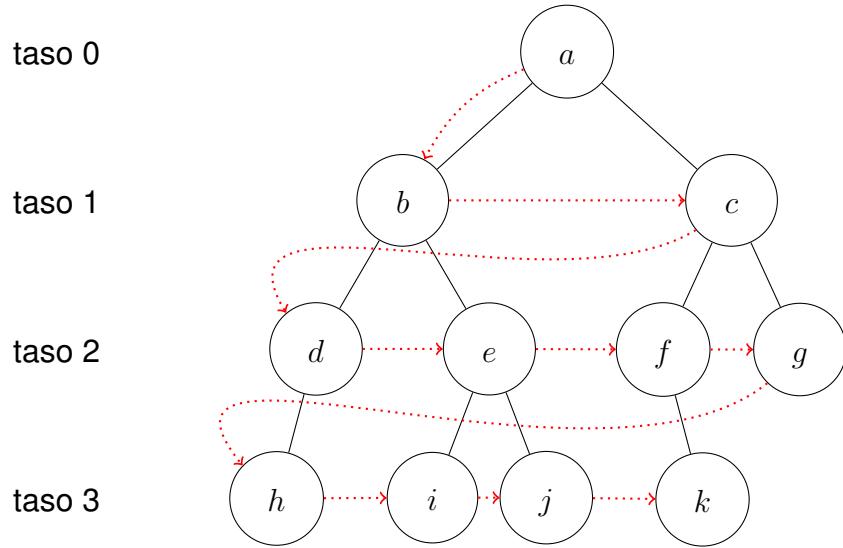
Puurakenne tallennetaan levylle leveysjärjestysessä (engl. *breadth-first*). Tämän ansiosta puun tasot muodostavat luonnolliset tarkkuustasot: juurisolmun pallo esittää koko mallia, ensimmäinen taso sisältää muutaman pienemmän pallon, ja niin edelleen. Kun tällainen tiedoston sisäinen rakenne yhdistetään ulkoisen muiston teknikoihin, voidaan täplien piirtäminen aloittaa heti, kun tarpeeksi puun solmuja on ladattu levyltä muistiin.⁵

Puun rakennetta on havainnollistettu kuvassa 9. [29]

Toinen hyödyllinen QSplatissa käytetty tekniikka on koordinaattien kvantisointi (engl. *quantization*). Kun tarkkuudesta voidaan tinkiä, solmujen pallojen absoluuttisia koordinaatteja ei tallenneta, vaan niiden sijainti ilmaistaan suhteessa vanhempiinsa. Pallon säteen ja keskipisteen suhteellisen poikkeaman ilmaisemiseen käytetään vain 13:a arvoa. Pallon säde r voi olla välillä $[\frac{1}{13}, \frac{13}{13}]$ ja samaten keskipisteen suhteellisen poikkeaman x, y ja z -koordinaatit ovat vanhemman pallon läpimitan kolmastoistaosan monikertoja. Kun

⁴Itse asiassa Rusinkiewicz ja Levoy käyttivät laserkeilatusta pistepilvestä muodostettua kolmioverkkoa, jonka tietorakenne esitti yksinkertaisempuna pistedatanana.

⁵Rusinkiewicz ja Levoy päättävät ruudulle projisoitujen täplien koon perusteella kuinka syvälle puussa tulee edetä.



Kuva 9: Puun läpikäyntijärjestys (punainen katkoviiva) muodostaa luonnolliset tarkkuus-tasot

vielä hylätään vanhemman pallon ulkopuolella olevat keskipisteet ja käytetään hakutau-lua, voidaan pallon sijainti esittää vain 13:lla bitillä, kun normaali liukulukuesitys vaatisi vähintään 16 tavua. [29]

QSplat onnistui renderöimään 1,5-2,5 miljoonaa pistettä sekunnissa, mikä on sen ai-kaisella laitteistolla erinomainen tulos [29]. Kuten sanottu, se ei sellaisenaan kuitenkaan sovelli laserkeilattujen pistepilvien käsitteilyyn. Pistepilvien pisteiden normaaleja ei yleen-sä tiedetä, joten ne pitäisi esiprosessointivaiheessa selvittää esimerkiksi luvussa 1.3 esite-tyllä tekniikalla.

Toisena ongelmana voidaan pitää tapaa, jolla mallin kolmioita esitetään palloina. Tie-torakenteeseen luodaan nimittäin uutta dataa, kun lehtisolmuihin tallennettujen, yhden kolmion sisältävien pallojen lisäksi ylemmillä tasolla on keinotekoisia, monia kolmioi-ta kuvaavia palloja. QSplat tarjoaa kuitenkin monia tekniikoita, joita pistepilviä käsitte-levässä tietorakenteessa voidaan hyödyntää, kuten hierarkinen rakenne ja koordinaattien suhteellinen esitystapa.

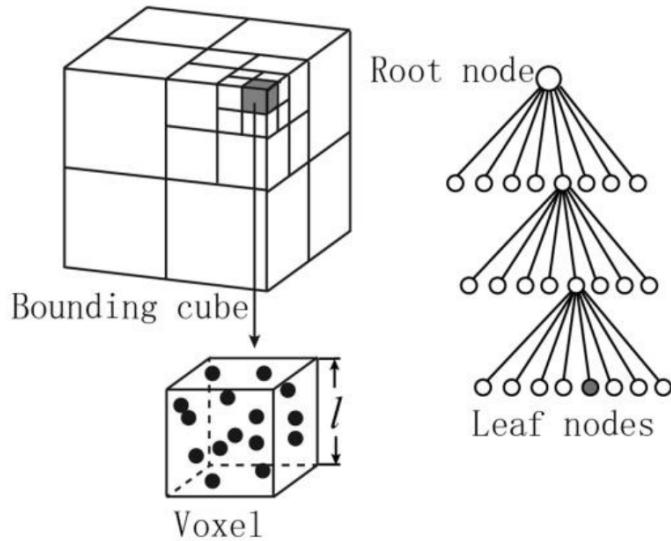
2.2.2 Peräkkäispistepuut

Dachsbacher et al. esittelevät niin kutstutun peräkkäispistepuun (engl. *sequential point tree*), jossa pisteet on aluksi järjestetty samaan tapaan pallopuuhun kuin QSplatissa. Pisteet sijaitsevat puun lehtisolmuissa ja sisäsolmuissa säilytetään täpliä, jotka juuri ja juuri peittävät solmun lasten rajauspallot. Täplien väri määräytyy lapsisolmujen värien keskiarvolla. [30]

Jokaiseen puun solmuun on virhelaskelmien perusteella lisätty rajat katseluetäisyydelle, jolla solmu valitaan visualisoitavaksi. Visualisointivaiheessa hierarkia litistetään taulukoksi, joka voidaan syöttää suoraan näytönohjaimelle. Näytönohjain käy taulukkoa läpi ja valikoi sopivat solmut, joiden perusteella täpliä piirretään ruudulle. [30]

Peräkkäispistepuut voidaan renderöidä nopeasti näytönohjaimen käytön ansiosta, mutta niissä on myös heikkouksia. Tietorakenteen vaatimuksena on, että kaikki data mahtuu näytönohjaimen muistiin. Näin on vain pienillä malleilla ja tilannetta pahentaa se, että peräkkäispistepuut eivät ole kovin säästäväisiä muistin suhteen. Hierarkian jokaisessa sisäsolmussa luodaan lisää dataa, kun lapsisolmujen unionia kuvataan täplän sijainnilla ja koolla, sekä keskimääräisellä värellä.

Wimmer ja Scheiblauer esittävät parannuksia peräkkäispistepuihin. Uusien täplien luomisen sijaan puun sisäsolmuissa valitaan lapsisolmuista edustaja, joka parhaiten kuvaa sisäolmun esittämää avaruuden osaa. Tämä on tärkeä huomio, sillä käsiteltäessä massiivisia pistepilviä tulisi välittää ylimääräisen datan luomista. Hierarkiasta muodostetaan tarkkuustasot siten, että alempat tarkkuustasot sisältyvät ylempiin tasoihin ja visualisoitaessa oikea tarkkuustaso valitaan täplien koon ja katseluetäisyyden perusteella. Solmua vastaavan täplän koko määräytyy siitä, kuinka syvällä hierarkiassa se on. Wimmer ja Scheiblauer kutsuvat tätä rakennetta muistioptimoiduksi peräkkäispistepuuksi. (engl. *memory optimized sequential point tree, MOSPT*) [31]



Kuva 10: Oktettipuun juurisolmu sisältää kaikki pisteet sisältävän rajauslaatikon. Jokainen sisäsolmu jakaa rajauslaatikkonsa kahdeksaan osaan. Kuva: [32]

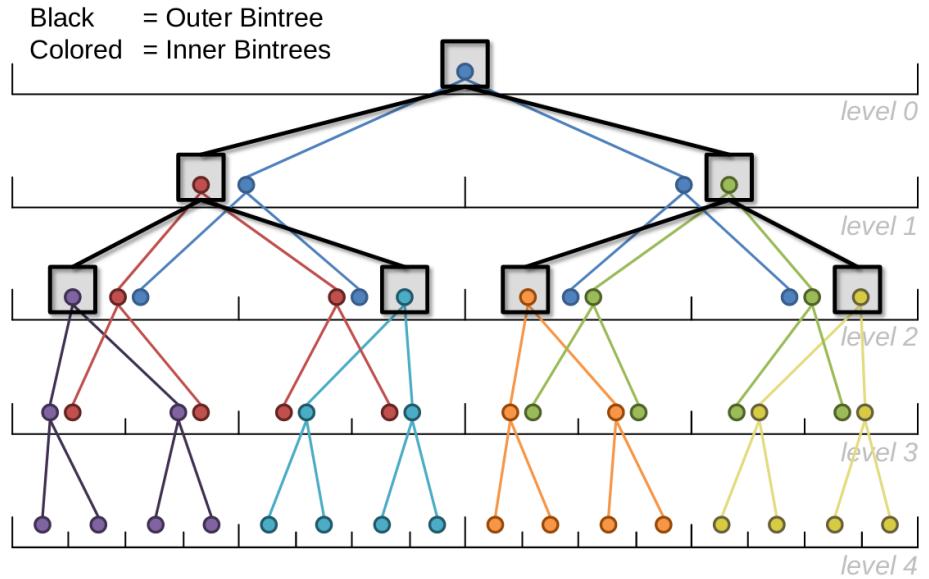
2.2.3 Sisäkkäispistepuut

Wimmer ja Scheiblauer kritisoivat muistioptimoituja peräkkäispistepuita siitä, että ne eivät tue näkökentän ulkopuolisten pisteiden tehokasta karsimista ja siitä, ettei muistioptimointi yksinään riittänyt poistamaan tarvetta ulkoisen muistin algoritmeille. Ratkaisuksi he esittivät sisäkkäisiä oktettipuita (engl. *nested octree*). Oktettipuu⁶ on yksinkertainen avaruutta rekursiivisesti jakava tietorakenne, jonka jokainen sisäsolmu jakaa kuvaamansa avaruuden kahdeksaan samankokoiseen osaan. Oktettipuun rakennetta on havainnollistettu kuvassa ??.

Wimmerin ja Scheiblauerin tietorakenteessa oktettipuita on kahdessa tasossa. Ulompana oktettipuuta käytetään avaruuden jakamiseen, sen tehokkaiseen läpikäymiseen ja näkökentän ulkopuolisten alueiden karsimiseen. Ulomman puun jokainen solmu sisältää yhden sisemmän oktettipuun, joka vastaa samaa avaruuden osaa, kuin ulkoisen puun solmu. Pisteet sijoitetaan sisempiin puihin, yksi jokaiseen solmuun. [31]

Sisäkkäisistä oktettipuista luodaan tarkkuustasot siten, että sisemmistä puista kerätään

⁶Tätä suomennosta käyttää esimerkiksi Davidsson [33]. Vaihtoehtoinen suomennos on kahdeksanpuu.



Kuva 11: Sisäkkäinen binääripuu, jossa sekä ulomman, että sisempien puiden syvyys on kolme. Ulompaa puuta kuvaavat mustat neliöt ja sisempää värikäät ympyrät. Puista muodostuu viisi tarkkuustasoa. Kuva: [28]

pisteitä ulomman puun tasojen mukaan. Tarkkuustasoon kuuluvat pisteet sijaitsevat siis ulomman puun samalla tasolla, mutta useiden sisempien puiden eri tasolla. Tarkkuustasojen muodostumista on havainnollistettu kuvassa 11. Puut tallennetaan levylle tarkkuus-taso kerrallaan, mikä mahdollistaa ulkoisen muistin algoritmien käytön. Visualisoitaessa tarvitsee levyltä lukea pisteitä vain haluttuun tarkkuustasoon asti, eikä loppuja pisteitä tarvitse ladata muistiin. [31]

Scheiblauer jalostaa sisäkkäisten oktettipuiden ideaa väitöskirjassaan esittelemällä muokkavat sisäkkäiset oktettipuut (engl. *modifiable nested octree, MNO*). Jos edellä esiteltyjä sisäkkäisiä oktettipuita halutaan muokata rakentamisen jälkeen, on sisemmät puut rakennettava ja muokattu hierarkia tallennettava levylle uudestaan. Nimensä mukaisesti MNO mahdollistaa tehokkaan pisteiden lisäämisen ja poistamisen. [28]

MNO:n rakenne eroaa sisäkkäisistä oktettipuista siten, että sisemmät puut korvataan säädöllisillä kolmiulotteisilla ruudukoilla, joihin pisteet tallennetaan. MNO:n rakentaminen alkaa juurisolmesta, joka vastaa kaikki pisteet peittävää avaruutta. Solmun sisältämä

ruudukko jakaa solmua kuvaavan avaruuden osan $128^3 = 2097152$ soluun. Pisteitä lisätään puuhun yksi kerrallaan niin, että jokaiseen ruudukon soluun mahtuu vain yksi piste. Jos solu on varattu, sijoitetaan piste ylimääräiseen taulukkoon odottamaan, että vastaavia pisteitä kertyy tarpeeksi, jotta olisi järkevä luoda uusia solmuja puuhun. Kun ennalta-määritty vähimmäismäärä pisteitä on kertynyt ylimääräisten pisteiden taulukkoon, luo-daan ruudukon sisältävälle solulle lapsisolmuja ja sijoitetaan ylimääräiset pisteet niihin. Ruudukkoon sijoitettavien pisteiden määärälle on hyvä asettaa myös yläraja. [28]

Jokainen tietorakenteen solmu tallennetaan omaan tiedostoonsa levylle, josta niitä ladataan muistiin visualisointivaiheessa tarvittaessa. Renderöintialgoritmiin kuuluu käyttäjän asettama pistebudjetti, joka asettaa ylärajan yhdessä ruudunpäivityksessä piirrettävien pisteiden määärälle.⁷ Tätä rajaa säätämällä käyttäjä saa jonkinlaisen kontrollin ruudunpäivitystaajuuden suhteeseen. [28]

Tiedostorakenne mahdollistaa hierarkian tehokkaan muokkaamisen. Lisättäessä uusia pisteitä MNO:hon tarkastetaan ensin, sijoittuuko se juurisolmun kuvaamaan avaruuden osaan. Jos näin on, onnistuu lisääminen kuten rakennusvaiheessa. Muussa tapauksessa juurisolmulle luodaan vanhempiakin kunnes jokin niistä muodostaa tarvittavan kokoinen avaruuden, ja piste lisätään sen ruudukkoon. Kun puun vanhan juuren yläpuolelle luodaan uusia solmuja, jää niiden ruudukot vajaaksi. Tällöin alemmista solmuista nostetaan pisteitä ylöspäin niin kauan, kunnes vajaita ruudukoita on vain lehtisolmuissa. Pisteiden poistaminen puusta on triviaalia, kun sisäsolmuihin mahdollisesti jäävät tyhjät ruudukot täytetään kuten pisteitä lisättäessä. [28]

Scheiblauer oli ottanut MNO:ta kehittääseen vaikuttaneita Michael Wandin et al. esittämää oktettipuusta, jonka sisäsolmuihin kuuluu myös ruudukko. Pisteet tallennetaan kuitenkin ruudukon sijaan lehtisolmuihin joihin mahtuu kuhunkin enintään satatuhatta pistettä. Sisäsolmuissa pidetään kopioita yhdestä niiden läpi kulkeneesta pisteestä ja tarkkustasot muodostetaan näistä sisäsolmujen pisteistä. Wandin et al. tietorakenne käyttää MNO:n

⁷Scheiblauer testasi pistepilvivisualisoijansa asettamalla rajan vain sataantuhanteen pisteeseen.

tapaan ulkoista muistia ja mahdollistaa tehokkaat lisäys- ja poisto-operaatiot. [34]

Markus Schütz jatkoi Wimmerin ja Scheiblauerin työtä esittelemällä opinnäytetyös-sääni verkkoselaimessa ajettavan Potree-nimisen pistepilvivisualisoijan. Potreen käyttämä tietorakenne perustuu Scheiblauerin muokkattaviin sisäkkäisiin oktettipuihin, mutta hierarkian rakennusvaiheessa kiinnitetään huomiota pisteiden tasaiseen jakautumiseen solmu-jen välille. Oktettipuun sisäsolmujen ruudukoihin hyväksytään uusia pisteitä vain, jos ne ovat tarpeeksi kaukana muista ruudukon pisteistä. Lehtisolmut hyväksyvät ennaltamää-rätyyn rajaan saakka kaikki pisteet, kunnes ne muutetaan sisäsolmuiksi ja liian lähekkäin olevat pisteet jaetaan uusien lapsisolmujen kesken. [35]

Potree käyttää ulkoista muistia tehokkaasti ja pystyy käsittelemään jopa 640 miljardia pistettä sisältäviä pistepilviä.⁸ Rakennusvaiheessa oktettipuun solmuja tallennetaan tasai-sin väliajoin levylle, jottei muisti täytyisi. Kun jokainen solmu tallennetaan omaan tie-dostoonsa, on yksittäisten solmujen tallentaminen ja lukeminen levyltä helppoa. Massii-visia pistepilviä kuvaavat hierarkiatkin voivat olla satojen megatavujen kokoisia. Schütz ratkaisee suurten hierarkioiden nopean lataamisen verkon yli jakamalla senkin puuraken-teeseen. Nämä voidaan välttää sekä turhien pisteiden, että näkökentän ulkopuolella olevien hierarkian haarojen lataaminen muistiin. [35]

Potreen visualisointialgoritmi priorisoi niitä hierarkian solmuja, jotka ovat lähellä kat-selupistettä ja joiden kuvaruudulle projisoitu koko on suurin. Renderöinnin suorituskykyä voidaan säädellä Scheiblauerin toteutuksen mukaisesti käyttäjän asettamalla pistebudje-tilla. Schütz on kehittänyt Potreehen myös näytönohjaimella ajettavan algoritmin mukau-tuaan pisteiden koon määrittämiseen; pistepilven harvemmissa osissa piirretään pisteet suurempina, jottei reikiä esiintyisi. [35]

⁸Kyseinen pistepilvi (Actueel Hoogtebestand Nederland, ANH2, <http://ahn2.pointclouds.nl/>) kuvailee koko Alankomaiden valtiota ja se vaatii 7,68 teratavua tallennustilaan. Potreen tietorakenteessa pistepilvi jakautui 13:lle tasolle ja 38:aan miljoonaan solmuun.

2.2.4 kd-puut

Suosittujen oktettipuiden lisäksi on pistepilvien visualisoinnissa käytetty kd-puita (engl. *k-dimensional tree, kd-tree*)⁹. Oktettipuusta poiketen kd-puut eivät jaa pistepilveä yhtä suuren tilavuuden omaaviin osiin, vaan jokainen solmu jakaa kuvaamansa alueen kahtia niin, että tilanjakotason molemmille puolille jää yhtä monta pistettä. Tätä varten pisteet on järjestettävä, mikä pidentää puun rakennusaikaa. Toisaalta tietorakenteen siirtely kiintolevyltä muistiin ja sieltä näytönohjaimelle on helpompaa, kun puu on tasapainoinen, eli kaikki solmut ovat saman kokoisia. [36]

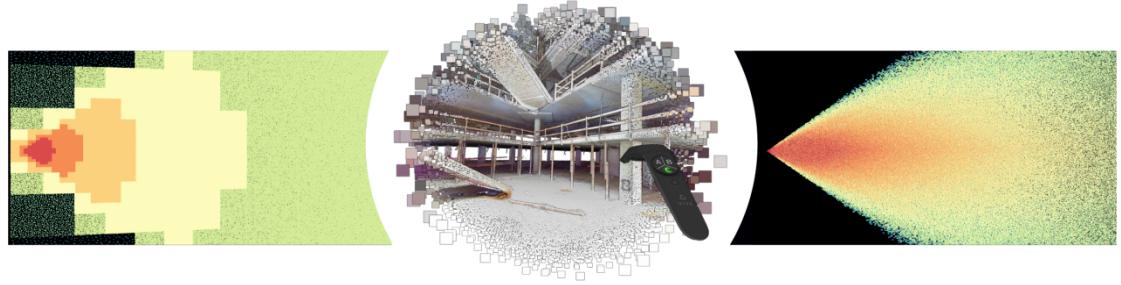
Richter et al. kehittivät massiivisten ulkoilmakeilausten visualisointiin kd-puuhun perustuvan pistepilvirenderöijän, joka käytti luokiteltua pistepilvidataa. Pisteet oli jaettu objektiluokkiin, kuten kasvillisuus, vesi tai rakennus, joille jokaiselle rakennettiin oma kd-puu. Renderöintivaiheessa kd-puista lähetetään solmuja näytönohjaimelle niiden kuvaruudulle projisoidun koon määräämässä järjestyksessä niin, että jokaisella objektiluokalla oli oma rajansa muistinkäytölle. [36]

Futterlieb et al. käyttivät samankaltaista kd-puuta luokittelemattomalle pistedatalle. Edellä mainitusta toteutuksesta poiketen tarkkuustaso valitaan etsimällä puusta yksi taso, jolta saadaan renderöityä sopiva määrä pisteitä. Varmistaakseen, että ruudulle saadaan aiina jonkinlainen tarkkuustaso renderöityä nopeasti Futterlieb et al. pitivät kahta miljoonaa pistettä näytönohjaimen muistissa jatkuvasti riippumatta siitä, olivatko ne näkyvissä. [37]

2.3 Ei-hierarkiset tekniikat

Pistepilvistä voi muodostaa tarkkuustasoja myös ilman hierarkista tietorakennetta. Markus Schütz et al. esittivät virtuaalitodellisuuslaseille suunnatun pistepilvirenderöijän, joka hierarkian muodostamisen sijaan käy koko pistepilveä läpi ja muodostaa siitä noin viiden ruudun välein uuden, sopivan tiheästi näytteistetyn osajoukon. Tämä jatkuvaksi tarkkuustasoiksi (engl. *continuous LOD*) kutsuttu tekniikka ei siis jaa pisteitä tietorakenteen sol-

⁹Kolmiulotteisten mallien visualisointiin käytetyissä kd-puissa luonnollisesti $k = 3$



Kuva 12: Vasemmalla pistepilvi on jaettu diskreetteihin tarkkuustasoihin, joiden välillä on selkeät rajat. Oikealla on käytetty jatkuvia tarkkuustasoja ja pisteet sulautuvat siististi kuvaan. Kuva: [38]

muihin, joilla on diskreetit tarkkuustasot, vaan pisteiden etäisyys toisistaan vaihtelee sen perusteella, kuinka kaukana ne ovat kamerasta. [38]

Jatkuvat tarkkuustasot ratkaisevat hierarkisia tietorakenteita käytettäessä usein esiintyvän ongelman diskreettien tarkkuustasojen näkyvistä rajoista. Renderöidyssä kuvassa ei näy selkeitä eroja matalalla ja korkeammalla tarkkuudella renderöityjen solmujen välillä kun tarkkuus laskee vähitellen kamerasta pois päin. Kuvassa 12 vasemmalla on havainnollistettu diskreetejä tarkkuustasoja ja oikealla jatkuvia tarkkuustasoja.

3 Laitossuunnitteluoohjelmistoon optimoitu tietorakenne

Tässä luvussa esitellään pistepilvien käsittelyyn ja visualisointiin soveltuva tietorakenne ja algoritmeja erityisesti laitossuunnitteluoohjelmiston tarpeisiin. Ensin määritellään vaatimukset tietorakenteelle tyypillisten pistepilvien käyttötapausten mukaan, jonka jälkeen valitaan alan kirjallisuudesta sovelluskohteelle hyödyllisimmät tekniikat.

3.1 Käyttötapaukset ja vaatimukset tietorakenteelle

Kolme yleistä käyttötapausta pistepilvien kanssa työskenneltäessä ovat mallintaminen, mittaaminen ja katselu, jotka asettavat erilaisia vaatimuksia pistepilviä käsitlevälle ja visualisoivalle laitossuunnitteluoohjelmistolle. Esitellään seuraavaksi käyttötapaukset ja niiiden asettamat vaatimukset.

3.1.1 Mallintaminen

Kun laitoksesta halutaan luoda ajantasalla oleva 3d-malli pistepilven avulla, täytyy se mallintaa suunnitteluoohjelmiston käyttämäksi geometriaksi pistepilveä mukailen. Lattiat ja seinät on tasoina helppo asettaa paikalleen, kuten myös suunnitteluoohjelmiston komponenttikirjastosta löytyvät laitteet. Suurin työ on yleensä putkistoissa, ilmakanavissa ja kaapeliradoissa. Useat suunnitteluoohjelmistot tarjoavat jonkinasteista automatisointia etenkin putkien reittykseen pistepilven päälle. Ohjelmisto voi automaattisesti tunnistaa pilvestä sylinterit ja asettaa niiden päälle sopivia putkisto-osia. Vaihtoehtoisesti käyttäjä voi valita pilvestä muutamia pisteitä ja ohjelmisto laskee niiden perusteella putken pituuden ja halkaisijan ja asettaa oikean osan paikalleen. Mallinnustyö ja etenkin automaattiset mudontunnistusalgoritmit asettavat ohjelmistolle vaatimuksen tarkkuudesta. Laitossuunnitteluoohjelmistossa käytetään yleensä millimetrejä perusyksikköinä, joten pistepilvessä ei sisisi esiintyä senttimetriien virheitä.

Mallintamisessa tärkeässä roolissa on suunnittelijan käyttämät näkymät ja pistepiven rajaaminen. Yleensä suunnittelija käyttää muutamaa koordinaattiakselien suuntaista nä-

kymää samanaikaisesti, jotta kurSORin saa helposti oIKEAan PAIKKAAN. NÄKYMÄN SYVYYS asetetaan usein hyvin pieneksi, jotta mallista näkyisi vain kulloisenkin mallinnustyön vaATIMA pieni siivu. Myös pistepilveä voidaan rajata niin, että siitä näkyy vain tarpeellinen osa. Pistepilviä visualisoivan ohjelmiston tulisi siis kyETÄ rajaamaan pilveä TOISTUVASTI ja NOPEASTI. KÄYTTÖKOKEMUS olisi paras, jos käyttäjä pystyisi hiirellä interaktiivisesti mÄÄRITTÄMÄÄN tilan, jonka sisäpuolella olevat pisteet renderöiTÄISIIN. Lisäksi pistepilvi tulee voida renderöidÄ useaan eri näkymään samanaikaisesti.

3.1.2 Mittaaminen

Toinen tärkeä ominaisuus pistepilvien kanssa työskennellessä on mittaaminen. Pistepilviä käytetään usein tarkastamaan, mahtuuko laitokseen jokin uusi laite tai putkisto. Tällöin on hyödyllistä suorittaa mittauksia joko kahden pistepilven pisteen, tai pisteen ja 3d-mallin geometrian välillä. Mittausoperaatiossa käyttäjä valitsee pistepilvestä kurSORilla haluamansa pisteen ja ohjelmisto palauttaa lähimäksi kursoria projisoidun pisteen. Käyttäjän kannalta olisi miellyttävää, jos mittausoperaatioita tehtäessä ei tarvitsisi odotaa, kun pistepilven miljoonien pisteiden joukosta etsitään juuri kurSORin alla oleva piste. Yksittäisten pisteiden hakeminen pilvestä tätyy siis olla nopeaa.

3.1.3 Katselu

Kolmas yleinen pistepilvien käyttökohde on 3d-mallin katselu joko laitossuunnitteluoHjelmistossa tai erityisessä mallinkatseluohjelmistossa. Etenkin suunnitteluprojektien esimiehet haluavat usein tarkastella suunnittelijoiden luomaa 3d-mallia helposti ja nopeasti. Luonnollisesti malliin kuuluvat pistepilvet tulevat myös näkyä katselijalle. Tämä saattaa tuottaa haasteita ohjelmiston kannalta, sillä katseluohjelmistojen käyttäjillä on käytettävissä harvoin yhtä järeää laitteistoa, kuin suunnittelijoiden työasemat. Mallinkatseluohjelmistossa pistepilveä harvemmin rajataan pienemmäksi, joten renderöiTÄVIÄ pisteitä on niin paljon, etteivät ne mahdu kerralla keskusmuistiin tai näytönohjaimen muistiin. Yleen-

sä käyttäjä myös liikuttaa näkymää mallin ympäri enemmän kuin mallinnustyössä, joten pistepilvisualisoinnin suorituskyky ja tarkkuustasot ovat entistäkin tärkeämpiä.

Tässä tutkielmassa kehitetään laitossuunnitteluoohjelmistolle optimoitu hierarkinen tietorakenne pistepilvien käsittelyyn. Esitetään tietorakenteelle seuraavat vaatimukset edellä mainittujen käyttötapausten perusteella:

1. On voitava visualisoida karkea yleiskuva pistepilvestä vain pienellä osalla datasta.
2. On käytettävä ulkoisen muistin algoritmeja, eli koko pilveä ei pidetä kerralla keskusmuistissa.
3. Pistepilven vaatima tallennustilan määrästä voidaan laskea harventamalla sen tiheasti näytteistettyjä osia.
4. Käyttäjän on voitava määrittää pilvestä alueita, joiden sisältävien tai ulkopuolelle jäävien pisteiden ominaisuuksia, kuten näkyvyyttä tai väriä, voidaan muuttaa.
5. Pilvestä on voitava nopeasti ja tarkasti valita yksittäisiä pisteitä.
6. Pistepilvessä ei saa esiintyä yli millimetrin suuruisia virheitä.

3.2 Tietorakenteen valinta

Luvussa 2.2 esitelly Potree on osoittaunut massiivisten pistepilvien interaktiivisen visualisoinnin olevan mahdollista jopa verkkoselaimessa, jossa etenkin tiedonsiirtonopeus rajoittaa renderöinnin nopeutta. Tarkastellaan siis sisäkkäispistepuiden soveltuvuutta laitossuunnitteluoohjelmistoon.

Oktettipuun läpikäyminen taso kerrallaan muodostaa tehokkaasti tarkkuustasot, joten vaatimus 1 on helppo tyydyttää. Pisteiden asettelu sisäkkäisten oktettipuiden solmuihin mahdollistaa myös vaatimuksen 2 mukaisesti ulkoisen muistin käyttämisen. Scheiblaueerin muokattavien sisäkkäisten oktettipuiden jokainen solmu sisältää ruudukon, johon pisteeet sijoitetaan. Mitä syvemmällä tasolla solmu on, sitä pienempiä ruudukon solut ovat.

Vaatimuksen 3 esittämä pilven harvennus onnistuu asettamalla puulle enimmäissyyvyys ruudukon koon mukaan ja hylkäämällä lehtisolmuissa kaikki pisteet, jotka tulisi lisätynsi jo varattuun soluun.

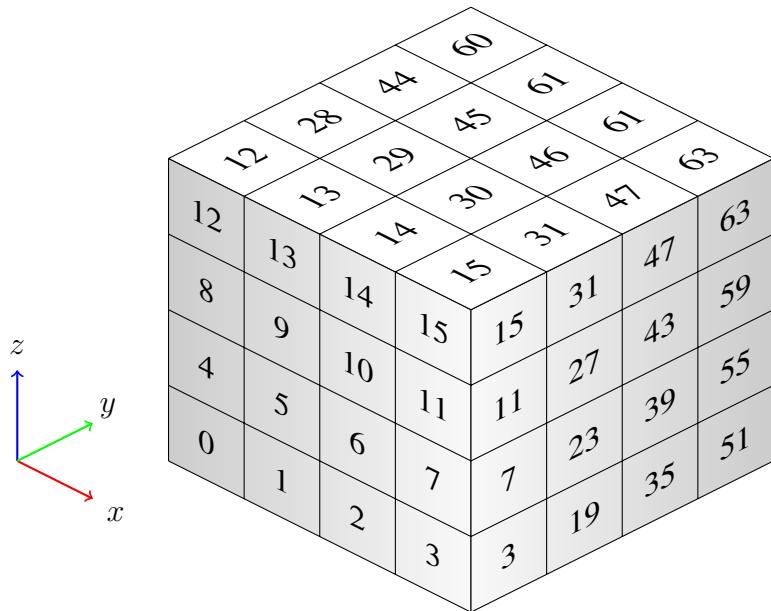
Valintaoperaatiot onnistuvat nopeasti oktettipuussa. Puun jokainen solmu sisältää tiedon sen sisältämien pisteiden rajauslaatikosta (engl. *bounding box*), joten jos valinnan sijainti ei osu rajauslaatikon sisälle, ei kyseisen solmun lapsisolmujakaan tarvitse tarkastaa. Yksittäisiä pisteitä tarvitsee tarkastella vasta kun valittavan alueen raja kulkee puun solmun rajauslaatikon läpi, tai kun käyttäjä haluaa valita vain yhden pisteen. Tietyllä aluella sijaitsevat pisteet jakautuvat useaan oktettipuun solmuun, minkä johdosta valintaoperaatiot eivät ole triviaaleja sisäkkäisissä oktettipuissa. Vaatimuksiin 4 ja 5 voidaan kuitenkin vastata sisäkkäisillä oktettipuilla. Scheiblauer ja Schütz eivät tiivistäneet pistepilviä, joten niiden tarkkuus ei kärsinyt. Näin myöskaän vaatimus 6 ei tuota ongelmia.

Sisäkkäispistepuut näyttävät siis soveltuvan myös laitossuunnitteluhjelmiston tarpeisiin. Scheiblauerin ja Schützin tietorakenteiden käyttämät ruudukot mahdollistavat kuitenkin myös joitakin laitossuunnittelusovelluksille tärkeitä optiomointeja. Esitellään seuraavaksi, miten pistedataa voi tiivistää ruudukon avulla.

3.3 Pistedatan esitysmuoto

Pistepilvet sisältävät usein satoja miljoonia tai jopa miljardeja pisteitä, mikä johtaa luonnollisesti isoihin tiedostokokoihin. Yleensä hierarkiatiedon osuus tiedoston sisällöstä on varsin pieni, joten tiedostokokoa saadaan pienennettyä parhaiten tiivistämällä itse pisteiden esitysmuotoa. Pisteistä tarvitsee tallentaa vähintään niiden sijainti ja väri. Yleensä sijainti esitetään kolmella nelitavuisella liukuluvulla ja väri RGB-koodauksella kolmella yksitavuisella kokonaisluvulla:

```
struct Point {
    float x;
    float y;
```



Kuva 13: 64:n solun ruudukko, jonka indeksointi alkaa vasemmasta alakulmasta

```
float z;  
  
unsigned char r;  
  
unsigned char g;  
  
unsigned char b;  
  
}
```

Yleensä näiden 15:a tavun lisäksi lisätään yksi pakkaustavu, jotta koko olisi mukava kahden potenssi. Miljardi pistettä tallennettuna tässä esitysmuodossa vaatii siis 16 gigatavua muistia.¹⁰ Muuttamalla pisteiden esitysmuotoa voidaan pistepilviä tiivistää ja joitakin operaatioita nopeuttaa.

Sisäkkäispistepuiden käyttämä ruudukko mahdolistaan yksinkertaisen pakkausalgoritmin. Puun rakennusvaiheessa pisteet lisätään jokaisessa solmussa olevaan kolmiulotteiseen ruudukkoon, jonka jokaiseen soluun mahtuu vain yksi piste. Kun piste lisätään ruudukon soluun, tallennetaan solun järjestysnumero hajautustauluun, josta voidaan jatkossa nopeasti tarkastaa, onko kyseinen solu varattu. Mahdollinen numeroointi ruudukolle on

¹⁰Pistepilvisovelluksen käyttäjän rahapussin koosta riippuen tämän kokoinen pilvi mahtuisi vielä keskusmuistiin ja jopa näytönohjaimen muistiin [39].

esitetti kuvassa 13. Numerointi alkaa vasemmasta alakulmasta indeksistä nolla ja etenee vasenkätisen koordinaatiston mukaisesti.

Puun solmuihin on tallennettu rajauslaatikko, joka määrittää myös ruudukon mitat ja sijainnin. Ruudukkoon lisättävien pisteen absoluuttinen sijainti voidaan unohtaa ja käyttää sijainnin tallentamiseen pisteen suhteellista sijaintia ruudukossa. Yksinkertaisimillaan voidaan kustakin pistestä tallentaa vain sen solun indeksi, jossa piste sijaitsee. Tällöin pisteen esitysmuoto on siis

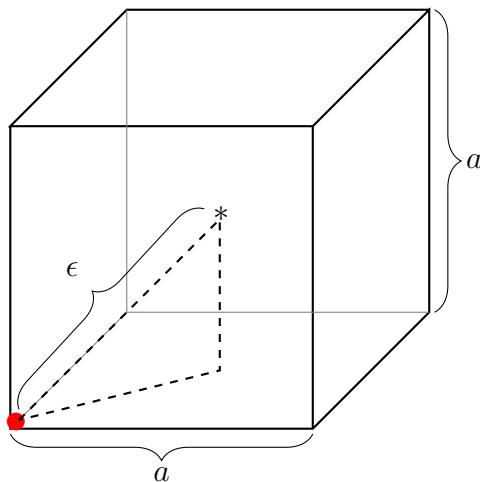
```
struct Point {
    unsigned int index;
    unsigned char r;
    unsigned char g;
    unsigned char b;
}
```

Kun solun indeksi tallennetaan etumerkittömänä nelitavuisena kokonaislukuna ja lisätään loppuun yksi pakkaustavu, tiivistyy piste kahdeksantavuiseksi. Näin miljardi pistettä vaatisi enää 8 gigatavua muistia.

Esitysmuota voidaan tiivistää vielä tästäkin. Pisteen etäisyyttä suhteessa ruudukkoon voidaan esittää kolmella tavulla niin, että jokainen tavu kuvailee koordinaattiakselien suuntaisten askelten määrää ruudusta 0 lähtien. Yhden tavun esittämä enimmäisarvo on 255, joten ruudukossa voi olla enintään $255^3 = 16581375$ solua. Tällainen kompressio tuo toki pistepilveen epätarkkuutta, johon palataan myöhemmin.

Usein laitossuunnittelussa käytetyissä laserkeilaimissa ei käytetä värikameraa antamaan pisteille värejä, vaan väri-informaatio johdetaan keilaimeen heijastuvan valon määristä. Tämä arvo normalisoidaan yleensä välille $[0, 255]$, josta saadaan jokin harmaan sävy. Tällaisissa pistepilvissä on siis turha tallentaa jokaiselle pisteelle r , g ja b -arvoja, kun vain yksi arvo riittäisi. Näin piste voidaan esittää muodossa

```
struct Point {
```



Kuva 14: Suurin mahdollinen ruudukossa esiintyvä virhe ϵ . Kuutio kuvaaa ruudukon solua ja asteriski sen visualisointiin käytettävää keskipistettä. Soluun lisätty punainen piste on juuri ja juuri solun sisällä.

```

unsigned char dx;
unsigned char dy;
unsigned char dz;
unsigned char intensity;
}
```

eli tarvitaan vain neljä tavua tilaa ja edellä mainittu pistepilvi saadaan tivistettyä neljännekkseen alkuperäisestä koostaan.

Kun pisteen tarkka sijainti unohdetaan ja se ilmaistaan suhteessa ruudukkoon, syntyy pistepilveen virheitä. Suurinta mahdollista virhettä on havainnollistettu kuvassa 14, kun piste visualisoidaan solun keskipisteenä, vaikka se oikeasti sijaitsisi aivan sen nurkassa. Jos oletetaan, että ruudukon solut ovat kuutioita, saadaan enimmäisvirhe laskettua helposti Pythagoraan lauseella muotoon

$$\epsilon = \frac{a\sqrt{3}}{2}. \quad (4)$$

Vaatimus 6 esitti virheen enimmäissuuruudeksi yhtä millimetriä. Oktettipuu rajauslaatikon sivun kahtia joka tasolla ja ruudukko sen edelleen pieniin soluihin. Jos ole-

tetaan esimerkiksi pistepilven rajauslaatikko kuutioksi, jonka sivun pituus on sata metriä ja ruudukon sisältävän Scheiblauerin ja Schützin käyttämää 128 solua jokaiseen koordinaattiakselin suuntaan, saavutetaan puun tasolla 11 ruudukko, jonka solun sivun pituus on $a = \frac{100m/2^{10}}{128} \approx 0,763mm$. Tällaisessa ruudukossa enimmäisvirhe on $\frac{0,763mm \cdot \sqrt{3}}{2} \approx 0,661mm$, joten kaikki tähän ruudukkoon lisättyt pisteet täytyvät vaatimuksen 6. Tämä ei tarkoita sitä, että puu voisi sisältää pisteitä vain tasolta 11 alas päin. Pisteitä voi hyväksyä suuriinkin soluihin, jos ne ovat tarpeeksi lähellä sen keskipistettä.

Pisteiden sijainnin suhteellinen esitysmuoto nopeuttaa myös pistepilven käsittelyä. Laitossuunnittelussa pistepilveä joudutaan usein sovellukseen latauksen jälkeen liikuttamaan ja skaalaamaan, jotta se istuisi hyvin 3d-malliin. Jos pisteisiin olisi tallennettu absoluuttinen sijainti, jouduttaisiin pistepilveä tallennettaessa uudelleenkirjoittamaan kaikki pisteet käyttäjän määrittämällä transformaatiomatriisilla kerrottuna. Edellä kuvatulla esitysmuodolla transformaatio täytyy suorittaa vain oktettiipuun solmujen rajauslaatikolla, joista tiivistetyn pisteen sijainti johdetaan. Tämä johtaa merkittävään parannukseen käyttökokemuksessa, sillä satojen miljoonien pisteiden kirjoittaminen levylle voi kestää kymmeniä minuutteja.

3.4 Tietorakenteen rakentaminen

Edellä todettiin sisäkkäispistepuiden sopivan pistepilven käsittelyyn käytettäväksi tietorakenteeksi laitossuunnitteluoohjelmistossa ja ehdotettiin pisteille kompaktimpaa esitysmuotoa. Käydään nyt läpi puun rakentamiseen käytettyjä algoritmeja.

Puun rakentaminen suoritetaan kahdessa vaiheessa. Ensin käydään kaikki syötepisteet läpi ja selvitetään niille rajauslaatikko. Tämän jälkeen luodaan tyhjä juurisolmu, jonka rajauslaatikkona toimii äsknen laskettu, kaikki pisteet sisältävä rajauslaatikko. Nyt syötepisteet voidaan lisätä yksitellen juurisolmuun, joka syöttää ne tarvittaessa edelleen lapsisolmuilleen. Rakentamisen ylintä tasoa on kuvattu algoritmissa 1.

Algoritmi 2 huolehtii pisteen pisteen lisäämisestä solmuun. Piste hyväksytään sol-

Algoritmi 1: RakennaOktettipuu

Syöte : Joukko pistepilviä P

Tuloste: Pisteet sisältävän oktettipuun juurisolmu s

```

1 // Selvitetään ensin pilvien rajauslaatikkojen unioni
2  $L \leftarrow$  tyhjä rajauslaatikko
3 for pistepilvi  $pc \in P$  do
4   for piste  $p \in pc$  do
5     if  $p \cap L = \emptyset$  then
6        $L \leftarrow L \cup p$ 
7     end
8   end
9 end

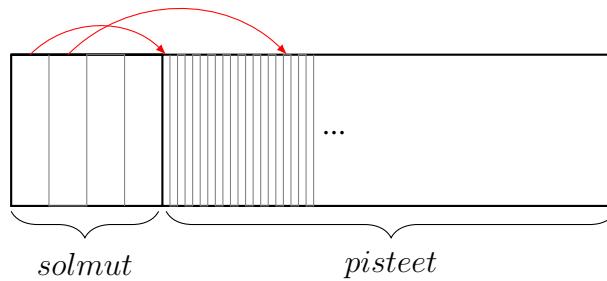
10 // Lisätään pisteet oktettipuuhun
11  $s \leftarrow$  juurisolmu, jonka rajauslaatikko on  $L$ 
12 for pistepilvi  $pc \in P$  do
13   for piste  $p \in pc$  do
14     LisääSolmuun( $s, p$ )
15   end
16 end

17 return  $s$ 

```

muun, jos sitä vastaava ruudukon solu on tyhjä, sen etäisyys solun keskipisteestä on pienempi kuin sallittu enimmäisvirhe ja solussa ei ole jo liikaa pisteitä. Solmujen sisältämien pisteiden määälle kannattaa asettaa yläraja, jotta saavutetaisiin sopiva haarautuminen.

Puun syvyyttä voi rajoittaa asettamalla ruudukon soluille vähimmäismitat. Yleensä la-serkeilaimen lähellä olevat pinnat on hyvin tiheästi näytteistettyjä ja tilannetta pahentaa, jos useat keilaimet ovat mitanneet samoja pintoja tiheästi. Algoritmin 2 rivillä 3 tarkastetaan, onko ylittäisikö uusi lapsisolmu puun enimmäissyvyyden. Kun ennaltamäärittylle



Kuva 15: Tiedostoon tallennetaan ensin puun solmut, joiden jälkeen kaikki pisteet ovat peräkkäin taulukossa. Punaiset nuolet kuvavat solmuihin tallennettuja indeksejä pisteitäulukkoon, josta siihen kuuluvat pisteet alkavat.

pohjatasolle hyväksytään kaikki pisteet, kunhan niitä vastaava ruudukon solu on vapaa, saadaan pistepilvelle tehokas ja globaali enimmäistiheys. Tällä tavalla pilveä saadaan harvnettua tehokkaasti ja vaatimus 3 voidaan tyydyttää.

Tietorakennetta tallennettaessa kirjoitetaan tiedostoon ensin tarvittavat tiedot puun solmuista ja pisteet vasta niiden jälkeen. Jokainen solmu sisältää tiedon siitä, kuinka monta pistettä siihen kuuluu, sekä ensimmäisen pisteen indeksin pistetaulukossa. Tämä mahdollistaa sen, että pistepilveä käsiteltäessä luetaan tiedostosta ensin vain kevyt hierarkia, jonka jälkeen vain tarvittavat pisteet voidaan lukea muistiin. Tiedoston rakennetta on havainnollistettu kuvassa 15. Pisteiden lukumäärän ja ensimmäisen pisteen indeksin lisäksi jokaisesta solmusta tallennetaan rajauslaatikko ja sijaintikoodi, joka kertoo sen sijainnin puussa. Sijaintikoodin numerot kertovat reitin juurisolmusta kyseiseen solmuun. Esimerkiksi koodi 014 tarkoittaa juurisolmun toisessa oktetissa sijaitsevan lapsen viidennenä oktetissa sijaitsevaa lasta. Lopuksi tarvitsee levylle kirjoittaa vielä millä puun tasolla se sijaitsee, jotta tiedostoa lukiessa tiedettäisiin sijaintikoodin pituus.

Yksittäinen solmu voidaan siis kirjoittaa levylle muodossa

```
struct Node {
    float min_x;
    float min_y;
    float min_z;
```

Algoritmi 2: LisääPisteSolmuun

Syöte : Puun solmu s ,

piste p , jolla on sijainti (x, y, z) ja väri (r, g, b)

```

1  $i \leftarrow$  sen ruudukon solun indeksi, jossa piste sijaitsee
2  $h \leftarrow$  hajautustaulu, johon solmun  $s$  pisteet tallennetaan
3 if Solmun  $s$  syvyys = puun enimmäissyvyys then
4   if  $i \notin h$  then
5     lisää  $i$  ja  $(r, g, b)$  hajautustauluun  $h$ 
6   else
7     Hylkää piste  $p$ 
8   end
9 else if  $s$  on täynnä then
10   $l \leftarrow$  uusi lapsisolmu
11  return LisääPisteSolmuun( $l, p$ )
12 else if  $i \in h$  then
13   $l \leftarrow$  uusi lapsisolmu
14  return LisääPisteSolmuun( $l, p$ )
15 else if  $\epsilon >$  ennalta määritetty enimmäisvirhe then
16   $l \leftarrow$  uusi lapsisolmu
17  return LisääPisteSolmuun( $l, p$ )
18 else
19    // Solmussa  $s$  ja sen ruudukon solussa  $i$  on tilaa, eikä virhe  $\epsilon$ 
      ole liian suuri. Piste voidaan lisätä solmuun  $s$ .
20  lisää  $i$  ja  $(r, g, b)$  hajautustauluun  $h$ 
21 end

```

```

    float max_x;
    float max_y;
    float max_z;
    unsigned char depth;
    unsigned char location_code[];
    unsigned int num_points;
    unsigned int point_index;
}

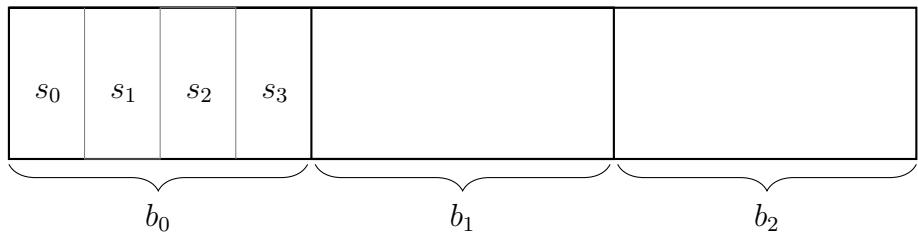
```

Nelitavuisilla liukuluvuilla ja kokonaisluvuilla vaatii solmu siis tallennustilaan $33+d$ tavua, missä d on solmun syvyys puussa.

Edellä kuvattu rakennusalgoritmi pitää koko oktettipiun muistissa ennen kuin pisteen kirjoitetaan levylle. Massiivisten pistepilvien tapauksessa keskusmuisti saattaa täytyä vaikka pilveä olisi harvennettu ja pisteitä kompressoitu. Muisin täytyessä käyttöjärjestelmä alkaa pitää osaa prosessin muistiavaruudesta kiintolevyllä sivutuksen (engl. *paging*) avulla [40]. Jatkotutkimuksen aiheeksi jää selvittää, onko nopeampaa antaa käyttöjärjestelmän siirtää sivuja kiintolevylle, vai esimerkiksi tallentaa ensin pisteitä väliaikaiseen tiedostoon niiden lisäsjärjestykseen, josta ne kopioidaan oikeassa järjestyksessä lopulliseen tiedostoon.

3.5 Renderöinti

Nopein tapa renderöidä pistepilvi on pitää kaikki pisteen näytönohjaimen muistissa ja joka ruudunpäivityksellä piirtää kaikki näkymässä näkyvät pisteen. Vaikka koko pistepilvi mahtuisi näytönohjaimen muistiin, täytyy laitossuunnitteluoohjelmistossa visualisoida muitakin grafiikkaa. Näytönohjaimen muistia voidaan säädää käyttämällä niin kutsuttua puskurivirtaa (engl. *buffer streaming*). Yleinen ongelma grafiikan renderöinnissä on se, että näytönohjain renderöi dataa nopeammin kuin suoritin ehtii sille syöttää. Puskurivirran ajatuksena on pitää näytönohjain mahdollisimman toimeliaana jakamalla puskuri,



Kuva 16: Pistedatan renderöinnissä käytetty puskuri, joka on jaettu osioihin b_0 , b_1 ja b_2 . Osioon b_0 on kirjoitettu pisteet solmuista s_0-s_3 . Puskurivirran tarkoituksesta on vähentää tiedonsiirrosta aiheutuvaa latenssia.

jonka kautta dataa siirretään keskusmuistista näytönohjaimen muistiin, kolmeen osaan. Samalla kun näytönohjain käsittelee yhtä puskurin osaa, suoritin voi täyttää toista datalla, ja kolmas on jo palautumassa suorittimen täytettäväksi. Teoriassa näytönohjain voi vain vaihtaa luettavaa puskuria ilman, että sen tarvitsisi odottaa lainkaan hidasta tiedonsiirtoa. [41]

Kuvassa 16 pistepuskuri on jaettu osioihin b_0 , b_1 ja b_2 . Puskurin koko on valittu niin, että jokaiseen osioon mahtuu neljän täyden solmun pisteet. Käytännössä solmut eivät kuitenkaan aina ole täysiä, joten on mahdollista, että osiot jäävät vajaaksi. Puskureita kuitenkin vaihdetaan tiuhaan, joten ei ole järkevää käyttää aikaa puskurin täytöasteen maksimointiin.

Tietorakenteen renderöintiä testatessa puskurivirta ei kuitenkaan tuottanut tarpeeksi miellyttäävisä visuaalista lopputulosta. Etenkin pilveä pyöriteltäessä ja sen läpi liikuttaessa puskurivirralla ehdittiin pilvestä renderöidä vain hyvin karkea tarkkuus niin, että ruudunpäivitystaajuus pysyi interaktiivisena. Ratkaisuna tähän näytti toimivan hyvin toinen pistepuskuri, jossa säilytetään karkeaa yleiskuvaa pilvestä. Tämä yleiskuvapuskuri pidetään näytönohjaimen muistissa, josta se on nopea renderöidä jokaisella ruudunpäivityksellä.

Algoritmissa 3 yleiskuvapuskuri täytetään ensimmäisellä renderöintikerralla puun ylimpien tasojen solmujen pisteillä, jotka muodostavat karkean, mutta kattavan yleiskuvan pilvestä. Yleiskuvapuskurista voisi yrittää vaihtaa näkymän ulkopuolella olevia pisteitä näkyviin solmuihin, mutta näkyvyystarkastelun suorittaminen ja puskurin muokkaaminen

Algoritmi 3: RenderöiYleiskuvapuskuri

Syöte : Sisäkkäispistepuu P

```

1  $b_{yleiskuva} \leftarrow$  pistepuskuri, jota pidetään näytönohjaimen muistissa
2 if Ensimmäinen renderöintikerta then
3   while  $b_{yleiskuva}$  ei ole täynnä do
4     // Käydään puuta läpi taso kerrallaan
5     for Solmu  $s \in P$  do
6       | Lisää solmun  $s$  pisteet puskuriin  $b_{yleiskuva}$ 
7     end
8   end
9   Lähetä  $b_{yleiskuva}$  näytönohjaimelle renderöitäväksi.
10  // Loput pisteet renderöidään puskurivirralla
11   $S \leftarrow$  näkyvissä olevat puun solmut, joita ei lisätty puskuriin  $b_{yleiskuva}$ 
12  Järjestä  $S$  ruudulle projisoidun koon mukaan
13  RenderöiPuskurivirta( $S$ )

```

jokaisella ruudunpäivityksellä osoittautui liian aikaavieväksi.

Yleiskuvan renderöinnin jälkeen jäljelle jäävät solmut järjestetään niiden kuvaruudulle projisoidun koon mukaan, koska pistepilvi näyttää tarkentuvan nopeammin, kun ensin renderöidään suuret ja lähellä kameraa olevat solmut. Järjestämisen jälkeen loput pisteet voidaan renderöidä algoritmissa 4 kuvatulla puskurivirralla.

Algoritmi 4 lisää kunkin solmun pisteet täyttövuorossa olevaan puskurivirran osioon. Puskuriosion täyttyessä vaihdetaan täytettäväksi seuraava osio. Puskurivirtaa käytäessä täytyy varmistaa, ettei sama osio ole sekä suorittimen kirjoitettavana, että näytönohjaimen luettavana. Tämä tapahtuu lähettämällä näytönohjaimelle jokaisen puskuriosion täytymisen jälkeen synkronointiobjekti (engl. *sync object*). Kun uutta osiota otetaan kirjoitettavaksi, tarkastetaan, että näytönohjain on merkannut kyseisen osion synkronointiobjektiin

Algoritmi 4: RenderöiPuskurivirta

Syöte : Sisäkkäispistepuun solmujoukko S

```

1  $b_0, b_1, b_2 \leftarrow$  kolmeen osioon jaettu pistepuskuri
2  $i \leftarrow 0$  // Täytettävän puskuriosion indeksi
3 for Solmu  $s \in S$  do
4   if Puskuriosiossa  $i$  ei ole tilaa solmun  $s$  pisteille then
5     Lähetä puskuriosio  $i$  näytönohjaimelle renderöitäväksi
6     Lähetä osion  $i$  synkronointiobjekti näytönohjaimelle
7      $i = (i + 1) \text{ mod } 3$ 
8     Odota, että näytönohjain on käsitellyt osion  $i$  synkronointiobjektiin
9   else
10    Lisää solmun  $s$  pisteet puskuriosioon  $i$ 
11 end

```

käsitellyksi. [42]

Algoritmi 3 suorittaa pistepilvelle näkyvyyskarsintaa (engl. *visibility culling*) valitsemaan renderöitäviksi vain ne solmut jotka ovat täysin tai osittain näkymäkartion (engl. *view frustum*) sisällä. Näkyvyyskarsinnan lisäksi algoritmin voidaan katsoa suorittavan yksinkertaista yksityiskohtien karsintaa (engl. *detail culling*), kun kuvaruudulle suurena projisoidut solmut renderöidään ensin.

Yksityiskohtien karsinta on suosittu nopeutustekniikka tietokonegrafiikassa. Ajatuksena on renderöidä vain tärkeimmät objektit ja jättää kaukana olevat tai pienet objektit renderöimättä, jotta ruudunpäivitystaajuus pysyy interaktiivisenä. Yksinkertainen tapa karsia yksityiskohtia on järjestää objektit niiden kuvaruudulle projisoidun koon mukaan ja renderöidä niitä tiettyyn rajaan asti, tai kunnes aika loppuu kesken. Akenine-Möller et al. [43] esittävät kuvaruudulle projisoidun objektiin koon arviolle kaavaa

$$a = \pi \left(\frac{nr}{\mathbf{d} \cdot (\mathbf{c} - \mathbf{v})} \right)^2, \quad (5)$$

jossa n on katselupisteen etäisyys kuvatasosta, r objektiin rajauspallon säde ja \mathbf{c} keskipiste, \mathbf{d} on normalisoitu katsomissuunta, ja \mathbf{v} katselupiste. [26]

Mikko Yllikäinen huomautti pro gradu -tutkielmassaan [26], ettei tarkkaa arvioita objektiien koosta kannata selvittää, jos halutaan selville vain suuruusjärjestys. Yllikäinen yksinkertaistaa kaavan muotoon

$$\begin{aligned} a &= \frac{r}{\mathbf{d} \cdot (\mathbf{c} - \mathbf{v})} \\ &\approx \frac{r}{\sqrt{(c_x - v_x)^2 + (c_y - v_y)^2 + (c_z - v_z)^2}} \\ &\propto \frac{r}{(c_x - v_x)^2 + (c_y - v_y)^2 + (c_z - v_z)^2}. \end{aligned} \tag{6}$$

Yllikäinen nimittää tällä kaavalla muodostetu suuruusjärjestyksen käyttämistä yksityiskohtien karsinnassa kontribuutiokarsinnaksi (engl. *contribution culling*). On huomattava, että tällainen karsinta ei ole mahdollista paralleliprojektiomaisemissa, joissa katseluetäisyys ei vaikuta objektiien kokoon. [26]

3.6 Pisteiden valitseminen

Yksittäisten pisteiden valitseminen pistepilvestä on hyvin raskas operaatio, jos pisteet eivät ole hierarkisessa tietorakenteessa. Oktettipuuta käytettäessä kaikkia pisteitä ei tarvitse kuitenkaan käydä läpi. Käyttäjän valitessa hiirellä piste ammutaan kamerasta säde kursorin projisoidun sijainnin läpi pistepilveen. Nyt pisteitä tarvitsee etsiä vain niistä oktettipuun solmuista, joihin säde osuu. Yksinkertaisimmillaan voidaan valita sädettä lähinnä oleva piste. Tällöin valituksi saattaisi kuitenkin tulla muukalaispiste, joka sattuu olemaan juuri kameran edessä. Yksinkertainen tapa välttää muukalaispisteiden valitsemista olisi hyvä sellaiset puun solmut, joissa säde osuu vain yhteen tai muutamaan pisteeseen.

Oktettipuun rakenne auttaa myös vaatimuksen 4 tyydyttämisessä. Jos pistepilvestä halutaan piilottaa tai korostaa tiettyä osaa, voi käyttäjä valita maisemasta laatikon ja muokata sen sisältämien pisteiden näkyvyyttä. Oktettipuuta käytettäessä ei jokaisen pisteen sisältymistä valintalaatikoihin tarvitse selvittää, vaan riittää tarkastaa, leikkaako valinta-

laatikko solmun rajauslaatikon kanssa. Vain tässä tapauksessa tarkastus tehdään pisteille.

Scheiblauer esittää väitöskirjassaan sisäkkäisille muokattaville oktettipuille erityistä tietorakennetta valittujen pisteiden käsittelyyn. Scheiblauer valitsee pisteitä puusta laatikkovalitsimella tai kolmiulotteisella siveltimellä (engl. *volumetric brush*), ja lisää ne erityiseen valintaoktettipuuhun (engl. *selection octree*). Kun halutut pisteet on valittu, kuvaa valintaoktettipuu valittujen pisteiden asuttamaa avaruuden osaa. Valintaoktettipuuta käytetään esimerkiksi pisteiden piilottamiseen näkymästä. Pisteitä renderöitääessä tarkastetaan, osuuko sen sijainti valintaoktettipuun solmuihin ja päätetään sen perusteella, hyvätkö piste vai ei. [28]

4 Tietorakenteen arvointi

Luvussa 3 todettiin sisäkkäispistepuun sopivan pistepilven käsitteilyyn ja visualisointiin myös laitossuunnitteluhjelmistossa ja esitettiin puun solmujen sisältämien ruudukoiden mahdollistama yksinkertainen kompressiotekniikka. Luvussa 4.1 mitataan sisäkkäispistepuun rakentamisen, tallentamisen ja läpikäynnin suorituskykyä ja arvioidaan kompression vaikutusta siihen. Luvussa 4.2 mitataan pistepilvien renderöintinopeutta luvun 3.5 algoritmeilla. Testikoneessa on Intel i7-8850H -suoritin, 32 gigatavua keskusmuistia, SSD-levy ja Nvidia Quadro P2000 -näytönohjain.

4.1 Tietorakenteen rakentaminen ja lataaminen

Tietorakenteen rakentamista, tallentamista ja muistiin lataamista arvioitiin kolmella pistepilvellä. *Pannuhuone*-pilvi sisälsi 20 keilausta, jotka veivät 18,7 gigatavua tilaa tallennettuna ascii-formaatissa. Keilaiksista muodostettiin puu, jossa oli 546572600 pistettä 528017:ssa solmussa, jotka jakautuivat yhdeksälle tasolle. *Toimisto* sisälsi 8,43 gigatavua pisteitä 21 keilauksesta ja siitä muodostetussa puussa oli 240727221 pistettä 608002:ssa solmussa 11:llä tasolla. *Pumput* oli testattavista pienin, vain 41:n megatavun kokoinen pistepilvi. Siinä oli 5 keilausta, joista rakennetussa puussa oli 1213990 pistettä 4561:ssä solmussa seitsemällä tasolla.

pisteiden esitysmuoto	tiedoston koko	rakentaminen ja tallentaminen	läpikäynti
16 tavua	8,17 GB	2574s = 42min 54s	6227ms
8 tavua	4,10 GB	1652s = 27min 32s	16190ms
4 tavua	2,07 GB	1602s = 26min 42s	13462ms

Taulukko 1: Pannuhuone-pilvestä muodostetun oktettipuun rakentaminen ja pisteiden läpikäyminen

Taulukoissa 1, 2 ja 3 on mitattu testipilvien rakentamiseen, tallentamiseen ja pisteiden

pisteiden esitysmuoto	tiedoston koko	rakentaminen ja tallentaminen	läpikäynti
16 tavua	3,59GB	891s = 14min 51s	2763ms
8 tavua	1,83GB	724s = 12min 4s	8492ms
4 tavua	961MB	712s = 11min 52s	6985ms

Taulukko 2: Toimisto-pilvestä muodostetun oktettipuun rakentaminen ja pisteiden läpikäyminen

pisteiden esitysmuoto	tiedoston koko	rakentaminen ja tallentaminen	läpikäynti
16 tavua	18,6MB	12s	14ms
8 tavua	9,57MB	10s	36ms
4 tavua	4,94MB	9s	29ms

Taulukko 3: Pumput-pilvestä muodostetun oktettipuun rakentaminen ja pisteiden läpikäyminen

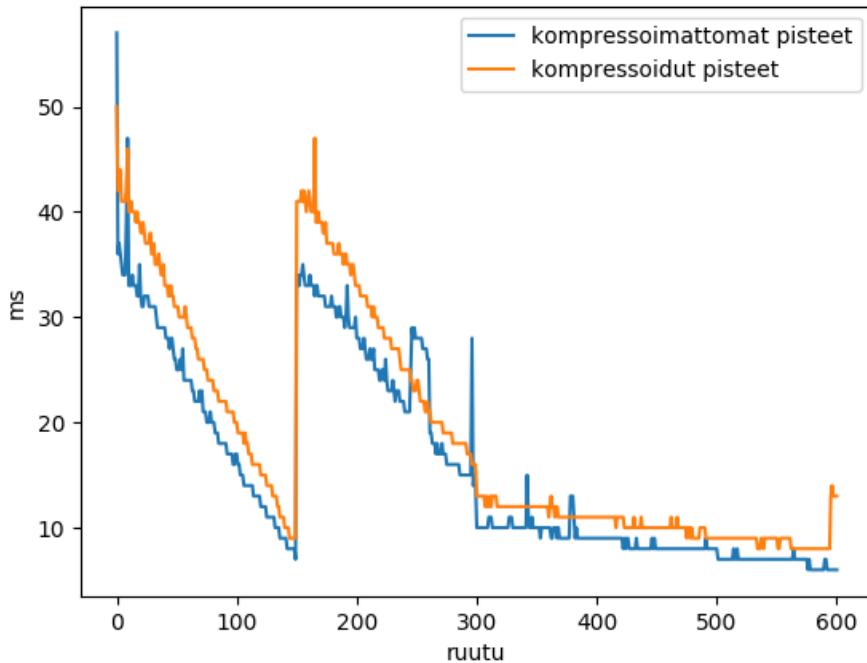
läpikäyntiin kuluva aika käyttää kolmea luvussa 3.3 käytettyä pisteiden esitysmuotoa: maailmakoordinaatit ja väri (16 tavua), ruudukon solun indeksi ja väri (8 tavua), sekä solun suhteelliset koordinaatit ja valon heijastumisen intensiteetti (4 tavua).

Yllätyksettömästi 16:n tavun pistedatan muistiin lataaminen ja läpikäyminen oli huomattavasti nopeampaa kuin kompressoitujen pisteiden. Nelitavuisten pisteiden lataaminen ja kompression purkaminen oli nopeampaa kuin kahdeksantavuisten. Tietorakennetta rakentaessa näyttää siltä, että pistedatan kirjoittaminen levylle vie huomattavan osan suoritusajasta. Tästä syystä on ajansäästön kannalta kannattavaa käyttää laskenta-aikaa pistedatan kompressointiin, jotta kirjoitettavia tavuja olisi vähemmän.

4.2 Renderöinti

Tietorakenteen renderöitiä arvioitaessa käytetään kahta pistepilveä. *Ilmanvaihtohuone* on keilattu Elomatic Oy:n Jyväskylän toimiston ilmanvaihdon konehuoneesta ja siinä on 13

keilausta, joista muodostetussa puussa on 506366789 pistettä 267641 solmussa kahdeksassa tasossa. *wroksite*-pilvi on Leica Geosystemsin testidataa, joka sisältää 7 keilausta, joiden 53881180 pistettä jakautuu 543105 solmuun yhdeksälle puun tasolle.



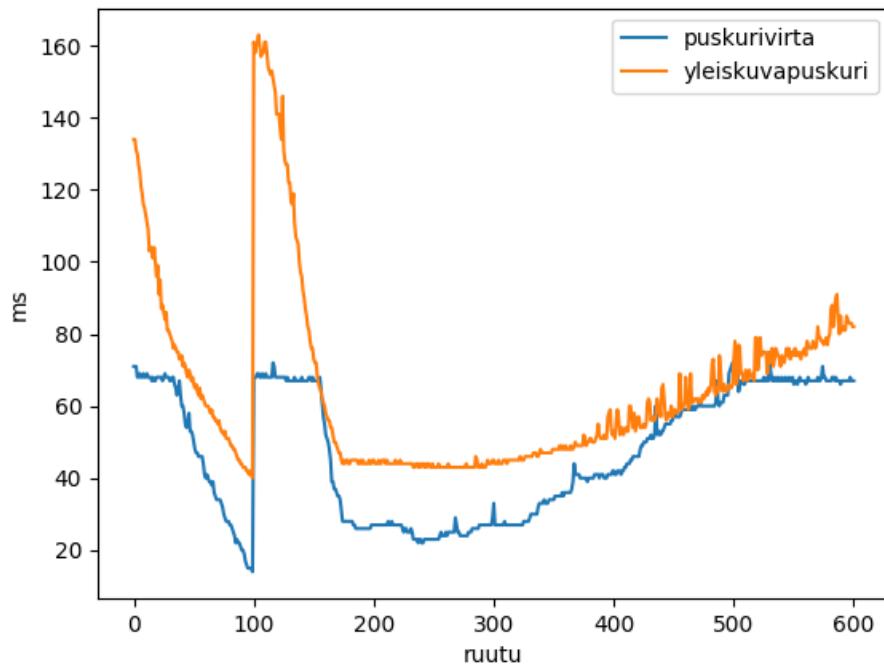
Kuva 17: Worksite-pilven kahden miljoonan pisteen renderöintiin vaadittu aika milisekunteina käyttäen kompressoimattomia 16:n tavun pisteitä ja kahdeksaan tavaan kompressoituja pisteitä.

Arviodaan ensin luvussa 3.3 esitellyn kompression vaikutusta pistepilven renderöintiaikaan. Kuvassa 17 on esitetty kaavio kahden miljoonan pisteen renderöinnin vaatimasta ajasta kamera-ajon jokaisella ruudunpäivityksellä. Sininen viiva kuvailee renderöintiaikaa kompressoimattomilla pisteillä ja oranssi viiva värit säilyttäväällä kompressiolla. Renderöinnissä on käytetty luvussa 3.5 esitelyä puskurivirta-algoritmia.

Kamera-ajo alkaa maiseman reunalta ja kulkee työmaan ohi lähestyen sen reunaa sitten, että näkyvissä olevien puun solmujen määrä laskee tasaisesti. Tämä näkyy myös kuvassa 17 ruudun renderöintiajan laskiessa. Näkymäkarttona sisällä olevien solmujen määrä

kasvaa äkkinäisesti noin 150:nnen ruudun kohdalla, jolloin kamera käännyy niin, että koko pilvi on näkyvässä. Lopuksi kamera lähestyy vastakkaisista seinää ja näkyvässä olevien solmujen määrä laskee.

Kuvaajasta huomataan, että kompressoimattomien 16-tavuisten pisteiden renderöinti on jonkin verran nopeampaa kuin kompressoitujen kahdeksantavuisten pisteiden. Eron selittää kompression avaamiseen vaadittu laskenta. Jokaisesta kompressoidusta pisteestä etsitään kompressoidun pisteen indeksiä vastaava ruudukon solu ja lasketaan sen keskipiste. Renderöintiajan ero on kuitenkin pieni ja voidaan katsoa, että miltei puolittunut tallennustilan tarve oikeuttaa pistedatan kompression.

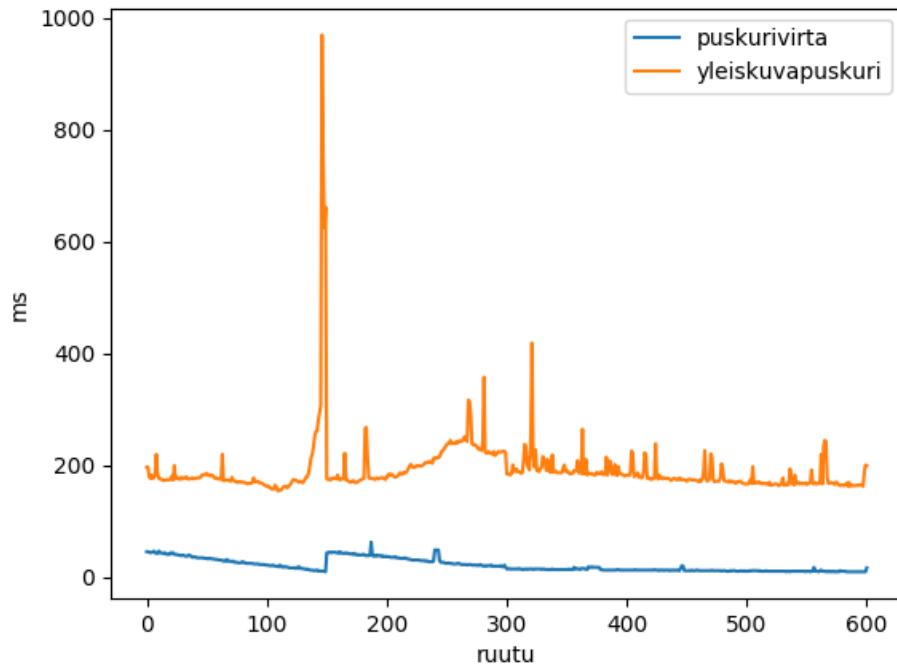


Kuva 18: Ilmanvaihtohuone: kahden miljoonan pisteen renderöintiin vaadittu aika millisekunteina kahdella eri renderöintialgoritmilla.

Luvussa 3.5 esiteltiin puskurivirran lisäksi yleiskuvapuskuria käyttävä algoritmi, joka pitää osaa pistepilvestä näytönohjaimen muistissa. Kuvassa 18 on mitattu kahden miljoonan pisteen renderöintiaikaa ilmanvaihtohuone-pilvessä. Sininen viiva kuvailee renderöinti-

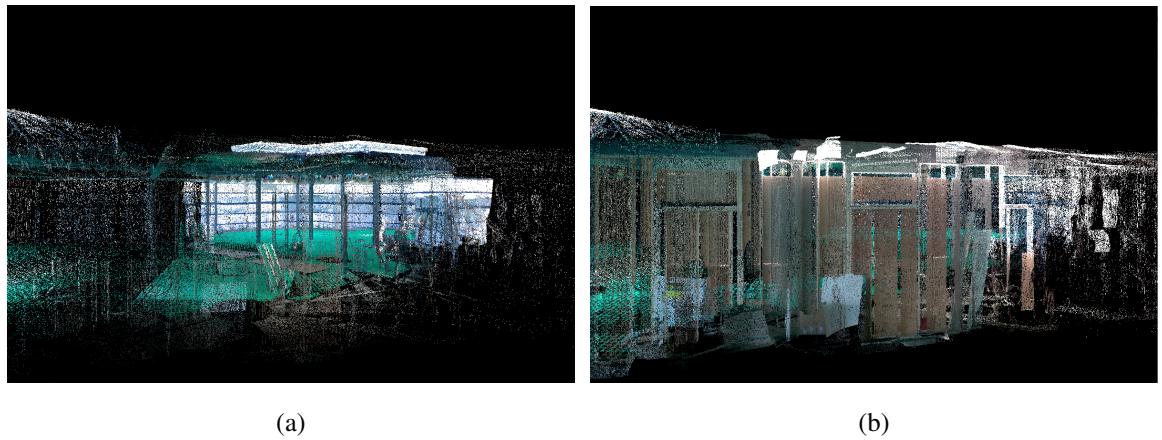
tiaakaan käytettäessä pelkkää puskurivirtaa ja oranssin viivan kuvaamassa mittauksessa on ensin renderöity yleiskuvapuskuri, minkä jälkeen jäljelle jäävät puun solmut on järjestetty kuvaruudulle projisoidun koon mukaan ja renderöity puskurivirralla. Yleiskuvapuskurissa on puun neljä ensimmäistä tasoa, joissa on yhteensä 286 solmua ja niissä 723834 pistettä.

Ilmanvaihtohuoneen kamera-ajo alkaa huoneen reunalta kameran osoittaessa vastakkaiselleseinälle. Kamera liikkuu kohti vastakkaista seinää, jolloin renderöitävien solmujen määrä vähenee. Kameran saavutettua vastakkaisen seinän noin sadan ruudun jälkeen se käännyy ympäri osoittamaan huoneen poikki. Näkyvissä olevien solmujen äkkinäinen kasvaminen näkyy jyrkkänä piikkinä kuvassa 18. Tämän jälkeen kamera lähestyy taas seinää ja renderointiaika kasvaa. Lopuksi kamera peruttaa pois päin seinästä ja näkyvillä olevien solmujen kasvava määrä pitkittää ruutujen renderointia.



Kuva 19: Worksite-pilven kahden miljoonan pisteen renderointiin vaadittu aika millisekunteina kahdella eri renderointialgoritmilla

Kuvassa 19 on tehty vastaavat mittaukset worksite-pilvelle. Oranssin viivan kuva-



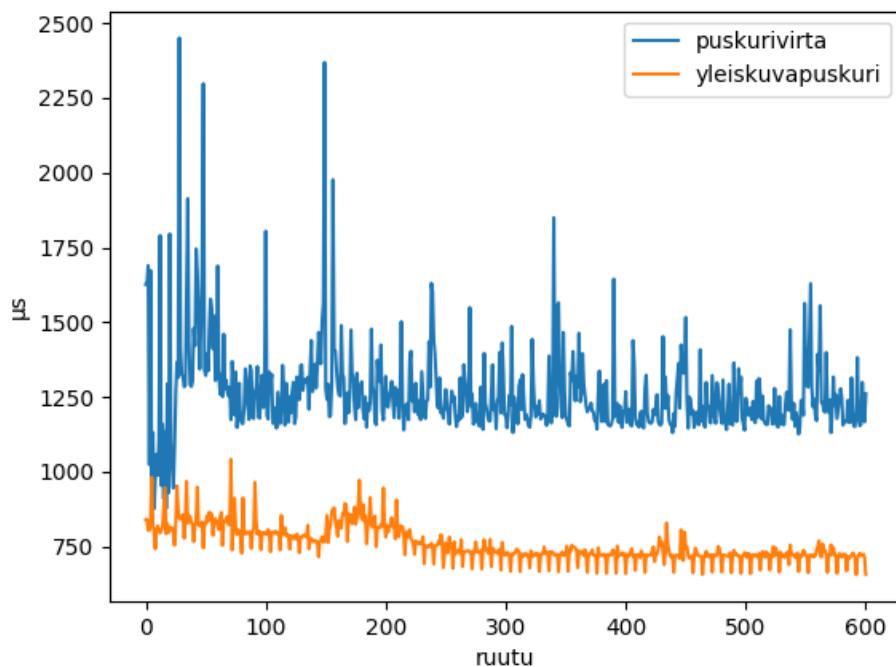
Kuva 20: Worksite-pilvi renderöity kahdella miljoonalla pisteellä käyttäen (a) pelkkää puskurivirtaa ja (b) yleiskuvapuskurin ja puskurivirran yhdistelmää. Pistepilvi on Leica Geosystemsin omaisuutta.

massa renderöintitavassa on yleiskuvapuskurissa puun viisi ylintä kerrosta, 1862 solmua ja 481663 pistettä.

Mittauksista selviää, että pelkän puskurivirran käyttäminen puun renderöinnissä on selkeästi nopeampaa kuin yleiskuvapuskurin ja puskurivirran yhdistelmällä. Tämä ero se- lityy puun solmujen järjestämiseen vaaditulla ajalla. Vaikka järjestäminen vie arvokasta laskenta-aikaa, voidaan sen parantavan lopputuloksen laatu. Kuvassa 20 vasemmalla puolella näkyy, kuinka puskurivirta on renderöinyt koko näkyvillä olevan pistepilven sa- malla pistetihedyllä ja taaimmainen seinä näyttää tarkemmalta kuin kameraa lähempänä oleva. Oikeanpuoleisessa kuvassa puun solmut on yleiskuvan renderöinnin jälkeen järjes- tetty ruudulle projisoidun koon mukaan, minkä seurausena etualalla oleva seinä näkyy selvästi. Solmujen järjestämistä voisi nopeuttaa helposti säikeistämällä renderöintialgor- timi niin, että yksi säie valikoi puusta solmuja prioriteettijonoon, josta toinen säie ottaa aina suurimman prioriteetin omaavan solmun renderöitäväksi.

Kameran liikkuessa pistepilven ohi riittää usein renderöidä vain karkea yleiskuva pil- vestä. Kun yleiskuvan pisteet pidetään näytönohjaimen muistissa, on niiden renderöimi- nen jokaisella ruudunpäivityksellä nopeaa. Kuvassa 21 on verrattu yleiskuvan renderöin-

tiä kun yleiskuvapuskuri on valmiiksi näytönohjaimen muistissa siihen, kun pisteet ladataan keskusmuistista puskurivirralla näytönohjaimelle. Yleiskuvapuskurin tapauksessa renderöintiaika on mitattu piirtokomennon suorittamisesta siihen, että näytönohjain on saanut kaikki pisteet renderöityä ja merkattua synkronointiobjektiin käsitellyksi. Näin kaikki pisteet on varmasti renderöity ruudulle ajastimen pysähtyessä. Pisteiden lataaminen levyltä ja kompression purkaminen on näissä mittauksissa jätetty pois.



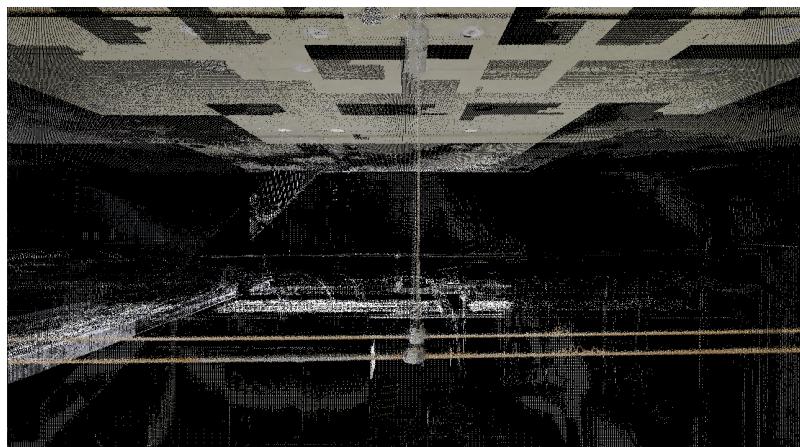
Kuva 21: Worksite-pilvestä muodostetun puun viiden ylimmän tason renderöintiin käytetty aika mikrosekunteina puskurivirralla ja kun pisteet ovat valmiiksi yleiskuvapuskurissa näytönohjaimen muistissa.

4.3 Pohdintaa

Luvussa 4.1 selvisi, että pistedatan kompressoiminen johtaa pienempien tiedostokokojen lisäksi myös oktettipuun rakentamisen nopeutumiseen. Toisaalta kompression purkaminen vie runsaasti laskenta-aikaa renderöintivaiheessa. Laitossuunnitteluhjelmistossa

tallennustilan tehokas käyttö ja pistepilven saaminen nopeasti katsottavaksi ovat tärkeää, joten olisi syytä tutkia, saadaanko kompression purkamista nopeutettua. Yksi mahdollisuus olisi säikeistää pisteiden lataamista niin, että yksi säie lukee pisteitä levyltä, toinen purkaa niiden kompressiota ja kolmas kopioi niitä näytönohjaimelle.

Pistebudjetin käyttö pistepilveä renderöitääessä mahdolistaa interaktiivisen ruudunpäivitysnopeuden, mutta huonontaa renderöidyn kuvan laatua. Kuvassa 22 näkyy ilmanvaihtohuoneen katossa ikäviä tarkkuustasojen eroja. Kun puusta renderöidään solmuja niiden kuvaruudulle projisoidun koon mukaan eikä taso kerrallaan, voi kuvassa esiintyä suuria tiheyseroja. Tiheyseroja voisi vähentää ja kuvan laatua parantaa implementoimalla esimerkiksi Schützin ?? ehdottaman muokkautuvan pistekoon algoritmin.



Kuva 22: Kahden miljoonan pisteen budjetin käyttäminen aiheuttaa ilmanvaihtohuonepilvessä häiritseviä tarkkuustasojen välisiä eroja.

5 Yhteenveton

Viitteet

- [1] T. W. Marc Levoy, Technical report, Computer Science Department, University of North Carolina at Chapel Hill (unpublished).
- [2] T. Qu ja W. Sun, Journal of Civil Engineering and Architecture **9**, 1269 (2015).
- [3] H. Rauno *et al.*, Technical report, Tiehallinto (unpublished).
- [4] C. Scheiblauer ja M. Pregebaumer, in *Proceedings of the 16th International Conference on Cultural Heritage and New Technologies* (PUBLISHER, 2011), pp. 242–247.
- [5] v. . Verkkolähde, <https://www.carnuntum.at/en/science-history/carnuntum-in-roman-times>.
- [6] F. Menna *et al.*, ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences **XLI-B5**, 675 (2016).
- [7] J. Hecht, Optics and Photonics News **29**, 26 (2018).
- [8] J. Huhtanen, Helsingin sanomat .
- [9] A. Saxena ja B. Sahay, *Computer Aided Engineering Design* (Anamaya Publishers F-230, Lado Sarai, New Delhi-110 030, India, 2005).
- [10] v. . verkkolähde, https://leica-geosystems.com/-/media/images/leicageosystems/about-us/news%20room/reporter/reporter-83/09-discovering-the-power-of-scanning/leica-espresso_expert_insights_640x750_slider4.ashx?la=en&ver=1.0
- [11] L. G. AG, 2018.
- [12] J. Fabritius, Opinnäytettyö, Tampereen ammattikorkeakoulu, 2009.
- [13] H. Houshiaar, J. Elseberg, D. Borrmann ja A. Nuchter, Geo-spatial Information Science **18**, (2015).
- [14] v.-h.-v. . 3DTK, The 3D Toolkit.
- [15] P. J. Besl ja N. D. McKay, IEEE Transactions on Pattern Analysis and Machine Intelligence **14**, 239 (1992).
- [16] C. Harris ja M. Stephens, in *In Proc. of Fourth Alvey Vision Conference* (PUBLISHER, 1988), pp. 147–151.
- [17] E. Rosten ja T. Drummond, in *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1* (PUBLISHER, 2005), Vol. 2, pp. 1508–1515 Vol. 2.
- [18] D. G. Lowe, in *Proceedings of the Seventh IEEE International Conference on Computer Vision* (PUBLISHER, 1999), Vol. 2, pp. 1150–1157 vol.2.

- [19] M. Weinmann, *Reconstruction and Analysis of 3D Scenes: From Irregularly Distributed 3D Points to Object Classes*, 1st ed. (Springer Publishing Company, Incorporated, 2016).
- [20] J. Giesen ja F. Cazals, (2006).
- [21] M. Berger *et al.*, Computer Graphics Forum (2016).
- [22] R. Schnabel, R. Wahl ja R. Klein, Comput. Graph. Forum **26**, 214 (2007).
- [23] C. M. Huang ja Y.-H. Tseng, in *29th Asian Conference on Remote Sensing 2008, ACRS 2008, 29th Asian Conference on Remote Sensing 2008, ACRS 2008* (PUBLISHER, 2008), pp. 1925–1930.
- [24] M. Piipponen, Opinnäytettyö, Satakunnan ammattikorkeakoulu, 2012.
- [25] Aveva, AVEVA Laser Modeller.
- [26] M. Yllikäinen, Master's thesis, 2013.
- [27] CADMATIC, Keskustelut CADMATIC:n henkilöstön kanssa, 2019.
- [28] C. Scheiblauer, Ph.D. thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, 2014.
- [29] S. Rusinkiewicz ja M. Levoy, Proceedings of SIGGRAPH **2000**, (2001).
- [30] C. Dachsbacher, C. Vogelsgang ja M. Stamminger, ACM Transactions on Graphics **22**, 657 (2003).
- [31] M. Wimmer ja C. Scheiblauer, in *Proceedings Symposium on Point-Based Graphics 2006*, Eurographics (Eurographics Association, 2006), pp. 129–136.
- [32] J. Yang, R. Li, Y. Xiao ja Z.-G. Cao, in *3D reconstruction from non-uniform point clouds via local hierarchical clustering* (PUBLISHER, 2017), p. 1042038.
- [33] J. Davidsson, Technical report, 3point Oy, Lapinlahdenkatu 16 00180 Helsinki (unpublished).
- [34] M. Wand *et al.*, Chen, Baoquan; Zwicker, Matthias; Botsch, Mario; Pajarola, Renato: *Symposium on Point-Based Graphics 2007 : Eurographics / IEEE VGTC Symposium Proceedings*, Eurogrphahics Association, 37-46 (2007) (2008).
- [35] M. Sch Master's thesis, .
- [36] R. Richter, S. Discher ja J. Döllner, in *Out-of-Core Visualization of Classified 3D Point Clouds* (PUBLISHER, 2014).
- [37] J. Futterlieb, C. Teutsch ja D. Berndt, IADIS International Journal on Computer Science and Information Systems **11**, 146 (2016).
- [38] (PUBLISHER, YEAR).

- [39] v. . Verkkolähde, <https://www.nvidia.com/en-gb/design-visualization/quadro/rtx-8000/>.
- [40] A. S. Tanenbaum ja H. Bos, *Modern Operating Systems*, 4th ed. (Prentice Hall PressUSA, 2014).
- [41] v. . Verkkolähde, https://www.khronos.org/opengl/wiki/Buffer_Object_Streaming.
- [42] v. . Verkkolähde, https://www.khronos.org/opengl/wiki/Sync_Object.
- [43] T. Akenine-Moller, T. Moller ja E. Haines, *Real-Time Rendering*, 2nd ed. (A. K. Peters, Ltd.Natick, MA, USA, 2002).