Pistepilvien visualisointi laitossuunnitteluohjelmistossa

Pro gradu -tutkielma
Turun yliopisto
Tulevaisuuden teknologioiden laitos
Tietojenkäsittelytiede
Maaliskuu 2020
Timo Heinonen
Tarkastajat:
Paavo Nevalainen
Mika Murtojärvi

TURUN YLIOPISTO
Tulevaisuuden teknologioiden laitos
Timo Heinonen Pistepilvien visualisointi laitossuunnitteluohjelmistossa
Pro Gradu, 61 s., 3 liites.
Tietojenkäsittelytiede
8. maaliskuuta 2020
Turun yliopiston laatujärjestelmän mukaisesti tämän julkaisun alkuperäisyys on tarkastettu Turnitin Originality Check -järjestelmällä.

Pistepilvet ovat useimmiten laserkeilaimilla mitattuja malleja jostakin tosimaailman esineestä tai maisemasta. Monimutkaisen geometrian sijaan kohdetta kuvataan valtavalla määrällä pisteitä, jotka visualisoimalla saadaan kuva yhtenäisistä pinnoista. Laserkeilaimella taltioituja pistepilviä käytetään esimerkiksi arkkitehtuurissa, rakentamisessa ja kuluttajatuotteiden suunnittelussa. Tässä tutkielmassa keskitytään pistepilvien käyttöön laitossuunnitteluohjelmistossa, 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 sellaisen tarkkuustason valitseminen, joka mahdollistaa interaktiivisen ruudunpäivitystaajuuden.

Tässä tutkielmassa perehdytään pistepilvien visualisoinnissa käytettyihin hierarkisiin tietorakenteisiin ja todetaan niin kutsuttujen sisäkkäispistepuiden soveltuvan laitossuunnitteluohjelmiston pistepilvivisualisoijassa käytettäväksi. Lisäksi esitellään sisäkkäispistepuille yksinkertainen kompressiotekniikka ja tärkeimpiä pisteitä priorisoiva visualisointialgoritmi.

Lopuksi mitataan kompression ja visualisointialgoritmin vaikutusta suorituskykyyn. Pistedatan kompressointi pienentää tiedostokokoja ja lyhentää tietorakenteen rakennusaikaa, mutta kompression purkaminen vaatii laskenta-aikaa visualisointivaiheessa. Esitelty visualisointialgoritmi piirtää pisteitä ruudulle hitaammin kuin suoraviivainen toteutus, mutta katselijaa lähinnä olevat pisteet tulevat piirretyksi tarkemmalla resoluutiolla ja tyydyttävä tulos saavutetaan pienemmällä määrällä visualisoituja pisteitä.

Avainsanat: pistepilvi, laserkeilain, tietokoneavusteinen suunnittelu, tietorakenteet, tietokonegrafiikka

Sisältö

1 Johdanto

Pistepilveksi kutsutaan suurta joukkoa pisteitä kolmiulotteisessa avaruudessa, jolla kuvataan esineiden, rakennusten tai maisemien pinnanmuotoja. Pistepilvi tuotetaan yleensä laserkeilaimella (engl. *laser scanner*), joka ampuu ympärilleen laserpurskeita ja mittaa etäisyyksiä pisteisiin, joista purske heijastuu takaisin. Katvealueiden välttämiseksi suoritetaan useita laserkeilauksia mitattavan kohteen ympäriltä. Laserkeilaimet mittaavat pisteitä niin tiheästi, että keilauksen lopputuloksena saadut pisteet näyttävät muodostavan yhtenäisiä pintoja.

Pistepilviä voidaan tuottaa myös synteettisesti mistä tahansa 3d-mallista. Yksinkertaisimmillaan kolmioverkosta voidaan unohtaa kärkipisteiden liitäntätieto, ja käyttää vain kärkipisteiden muodostamaa joukkoa kuvaamaan mallia. 1980-luvulta lähtien pisteitä on ehdotettu yleisiksi piirtoprimitiiveiksi kuvaamaan mitä tahansa geometriaa [1]. Ajatus on nykyään vielä ajankohtaisempi, sillä visualisoitavat mallit monimutkaistuvat jatkuvasti ja usein yksittäiset kolmiot projisoituvat kuvaruudulle alle pikselin kokoisina. Tällöin kolmion kärkipisteiden sijaan olisi tehokkaampaa säilyttää muistissa vain yhtä pistettä ja sen normaalivektoria. Grafiikkakirjastot ja näytönohjaimet on kuitenkin vielä toistaiseksi optimoitu kolmioiden käsittelyyn.

Pistepilvillä kuvataan hyvin erikokoisia kohteita yksittäisistä esineistä kokonaisiin valtioihin ja niitä käytetään esimerkiksi kuluttajatuotteiden suunnittelussa, rakentamisessa ja maanmittauksessa. Laserkeilauksen etu muihin mittaustekniikoihin on sen nopeus ja helppous: muutamalla keilaimen pyörähdyksellä saadaan taltioitua kokonainen huone millimetritarkkuudella. Laserkeilauksen jälkeen pistepilviä pitää usein esikäsitellä ennen kuin niitä voi käyttää suunnittelu- tai katseluohjelmistoissa. Näitä esikäsittelyvaiheita ovat muun muassa rekisteröinti, normaalivektorien etsiminen ja häiriönpoisto. Joitakin esimerkkejä pistepilvien sovelluskohteista ja esikäsittelyvaiheista on esitetty luvussa 2.

Keilauksen kohteesta riippuen tarvitaan yleensä miljoonia tai jopa miljardeja pisteitä, jotta saavutettaisiin tarpeeksi tiheä näytteistys. Pisteiden valtava määrä johtaa haasteisiin

niitä käsittelevissä ja visualisoivissa ohjelmistoissa. Pistepilviä sisältävät tiedostot voivat olla niin suuria, etteivät ne mahdu kerralla keskusmuistiin. Toinen ongelma on suuren pistepilven visualisointiin vaadittu aika. Jos halutaan säilyttää korkea ruudunpäivitystaajuus, ei koko pistepilveä voida piirtää jokaiselle ruudulle. Luvussa 3 etsitään näihin ongelmiin ratkaisuja alan kirjallisuudesta. Useimmat ratkaisuehdotukset käsittelevät hierarkisia tietorakenteita ja algoritmeja, jotka mahdollistavat pistepilvien asteittaisen lataamisen ja visualisoinnin. Näistä tietorakenteista valitaan tarkempaa tarkastelua varten sellainen, joka sopii laitossuunnitteluohjelmiston tarpeisiin.

Laitossunnitteluohjelmistoissa käytetään pistepilviä yleensä muutostöiden yhteydessä, kun laitoksesta halutaan saada ajan tasalla oleva 3d-malli. Nämä pistepilvet vaihtelevat kooltaan yhdestä huoneesta kokonaisiin tehdasalueisiin. Laitossuunnitteluohjelmistossa käytettävän pistepilvivisualisoijan on kyettävä pistepilven koosta riippumatta visualisoimaan reaaliaikaisella ruudunpäivitystaajuudella sekä yksityiskohtia läheltä katsellessa, että yleiskuva katselupisteen ollessa kauempana. Pistepilven tarkkuus ei saa kärsiä, sillä ohjelmistoa käyttävän suunnittelijan täytyy voida tehdä mittauksia pisteiden välillä. Luvussa 4 esitellään tekniikoita, joilla kirjallisuuden pohjalta valittua tietorakennetta, niin kutsuttua sisäkkäispistepuuta, hyödynnetään ja miten pisteiden esitysmuotoa muuttamalla saadaan pistepilveä kompressoitua. Luvussa 5 arvioidaan tietorakenteen ja kompression vaikutusta pistepilvivisualisoijan suorituskykyyn. Lopuksi luvussa 6 mainitaan muutamia tietorakennetta testattaessa havaittuja ongelmia ja jatkotutkimusaiheita.

Tämä tutkielma on tehty CADMATIC Oy:n toimeksiantona. CADMATIC on suomalainen ohjelmistoyritys, joka kehittää tuotteita laivojen ja laitosten tietokoneavusteiseen suunnitteluun. CADMATIC on toiminut alalla 1980-luvulta lähtien ja sillä on asiakkainaan yli tuhat organisaatiota yli viidestäkymmenestä maasta [2]. Yrityksen pääkonttori on Turussa. Tutkielmassa kehitettävää tietorakennetta testataan CADMATIC Plant Modeller -laitossuunnitteluohjelmistossa, sekä eBrowser-mallinkatseluohjelmistossa.

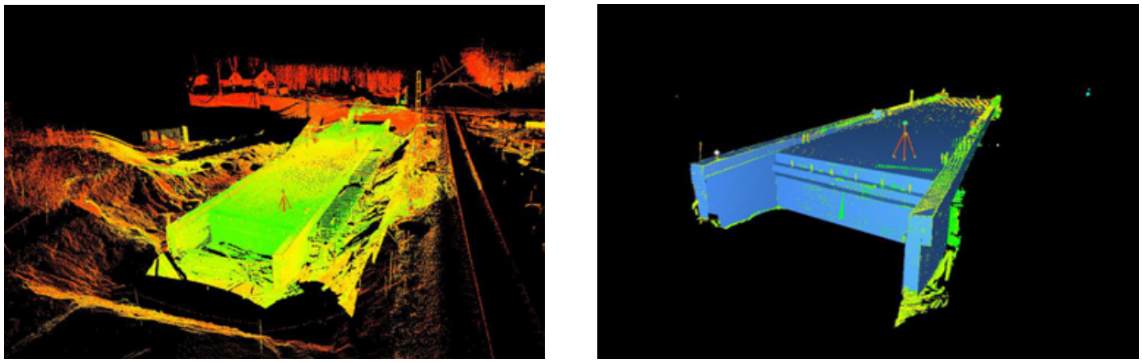
2 Pistepilvet

Pistepilville on useita käyttökohteita, joista arkkitehtuuri ja rakentaminen ovat tärkeimpiä. Pistepilvien hyödyntäminen voidaan aloittaa hyvin varhaisessa vaiheessa rakennusprojektia. Rakennettavan tontin ympäristöstä voidaan ottaa laserkeilauksia, jotta suunniteltavan rakennuksen sopimista tontille voidaan helposti arvioida. Rakennusprojektin aikana säännöllisesti tehdyillä laserkeilauksilla voidaan seurata tarkasti projektin etenemistä ja havaita mahdollisia ongelmia ajoissa. Pistepilvillä on myös tärkeä rooli rakennuksen valmistumisen jälkeen, sillä muutostöitä tehtäessä halutaan rakennuksesta saada ajantasalla oleva 3d-malli. Manuaalinen mallintaminen olisi hyvin työlästä verrattuna muutaman kymmenen pistepilven mittaamiseen laserkeilaimella, mikä voidaan tehdä päivässä. [3]

Eräs mielenkiintoinen sovelluskohde laserkeilaukselle on siltojen rakentaminen. Sillanrakennuksessa haasteita tuottavat tien ja maaston geometrian yhteensovittamisen lisäksi projektin pitkä kesto. Silta on rakennettava osissa ja lopputulokseen kohdistuu tiukkoja turvallisuusvaatimuksia, 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 mitattiin pistepilviä sillan kannesta ja rakenteista, minkä jälkeen niistä muodostettiin pintoja 3d-suunnitteluohjelmaan. Pistepilvi ja siitä muodostettu geometria on esitetty kuvassa 1. [4]

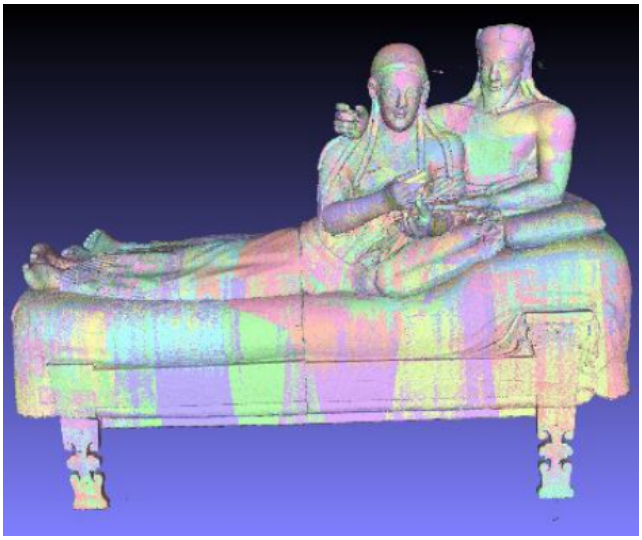
Pistepilviä käytetään hyväksi myös arkeologiassa. Roomalaiset perustivat vuoden 40 tienoilla Tonavan varrelle nykyisen Itävallan alueelle sotilasleirin, josta kasvoi myöhemmin Carnuntumin kaupunki. Kaupungin muurien ulkopuolelle rakennettiin amfiteatteri, johon mahtui 13 000 katsojaa. Vuonna 2007 Ala-Itävallan osavaltion hallitus aloitti amfiteatterin alueella arkeologiset kaivaukset, joiden yhteydessä alueesta muodostettiin kattava pistepilvi noin kahdellasadalla laserkeilauksella. Ruudunkaappaus kyseisestä pistepilvestä on esitetty kuvassa 2. [5]

Pistepilviä käytetään historiallisen kulttuuriperinnön säilyttämiseen myös pienemmäs-



Kuva 1: Silta Joroisten ja Varkauden välissä Kuvasintiellä. Vasemmalla pistepilvi, oikealla pistepilvestä muodostetua geometriaa suunnitteluohjelmassa. [4]

Kuva 2: Carnuntumin kaupungin amfiteatteri taltioituna laserkeilauksella. [6]



Kuva 3: Etruskilaisesta sarkofagista muodostettu pistepilvi. [7]

sä mittakaavassa. 1800-luvulla tehdyissä Cerveterin arkeologisissa kaivauksissa Italiassa löydettiin 500-luvulla terrakottasavesta tehty etruskilainen sarkofagi, joka kuvasi avioparia rentoutumassa tuonpuoleisessa. Satoihin palasiin hajonnut sarkofagi restauroitiin vuonna 1893 ja se digitoitiin käyttämällä laserkeilausta ja fotogrammetriaa vuonna 2013 taidenäyttelyä varten. Sarkofagista keilattu pistepilvi on esitetty kuvassa 3. [7]

Eräs vaativa pistepilvien sovelluskohde on itseohjautuvat kulkuneuvot. Voidakseen navigoida liikenteessä itseohjautuva auto tarvitsee kameroiden ja ultraäänisensoreiden lisäksi katolleen laserkeilaimen, jolla voidaan tarkkailla auton etäisyyttä muihin tienkäyttäjiin ja esteisiin. Itseohjautuvat autot ovat merkittävä tutkimuskohde myös pistepilvien käsittelyn kannalta. Auton katolle asennettavan laserkeilaimen tulisi olla tarkka, nopea ja edullinen, ja sen tuottamaa pistedataa täytyy voida käsitellä reaaliajassa. [8]

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 tarkemmilla laserkeilamilla myös rakennuksista. [9]

Laserkeilauksen kohteena ei ole aina maasto tai eloton rakennelma, vaan myös ihmisistä voidaan mitata pistepilviä. Esimerkiksi moottoripyöräkypärien suunnittelija voi käyttää hyväkseen ihmisen päätä kuvaavia pistepilviä selvittääkseen sopivan muodon kypärän sisukselle. Kypärän ulkokuori voidaan sen jälkeen mallintaa suunnitteluohjelmistolla niin, että se täyttää 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. [10]

2.1 Pistepilvien käsittely

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

Nykyaikaisella laserkeilaimella saadaan muodostettua hyvin tarkka ja tiheä pistepilvi nopeasti. Esimerkiksi kuvassa 4 näkyvän Leica Geosystemsin RTC360 -keilaimen luvataan mittaavan jopa kaksi miljoonaa pistettä sekunnissa ja kiertävän täyden ympyrän alle kahdessa minuutissa. Keilaimen lisäksi laitteessa on digitaalinen kamera, jolla saadaan määritettyä pisteille oikeat värit. [12]

Laserkeilaimien toimintaperiaatteissa on eroja. Kaksi yleisintä toimintaperiaatetta ovat kulkuaikatekniikka (engl. *time-of-flight*) ja vaihesiirtotekniikka (engl. *phase shift*). Kulkuaikatekniikkassa pisteen etäisyys keilaimesta selviää ajasta, joka kuluu laserpurskeen lähetettämisestä sen heijastuksen vastaanottamiseen. Pisteen etäisyys keilaimesta laske-

Kuva 4: Kulkuaikatekniikkaan perustuva Leica RTC360 -laserkeilain. [11]

taan yksinkertaisesti kaavalla

$$d = \frac{c \cdot t}{2}, \tag{1}$$

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

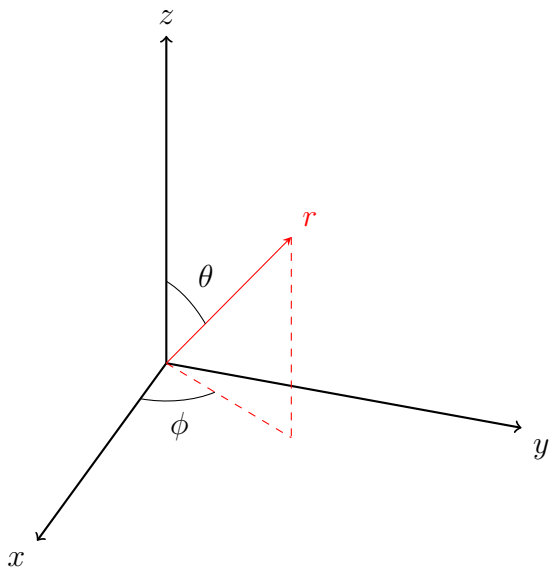
$$d = n \cdot \lambda + \frac{\Phi \cdot \lambda}{2 \cdot \pi}, \tag{2}$$

missä n on havainnon täysien aaltojen määrä, λ on signaalin aallonpituus ja Φ on lähtevän ja palaavan signaalin vaihe-ero. [13]

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. Pallokoordinaatistoa on havainnollistettu kuvassa 5.

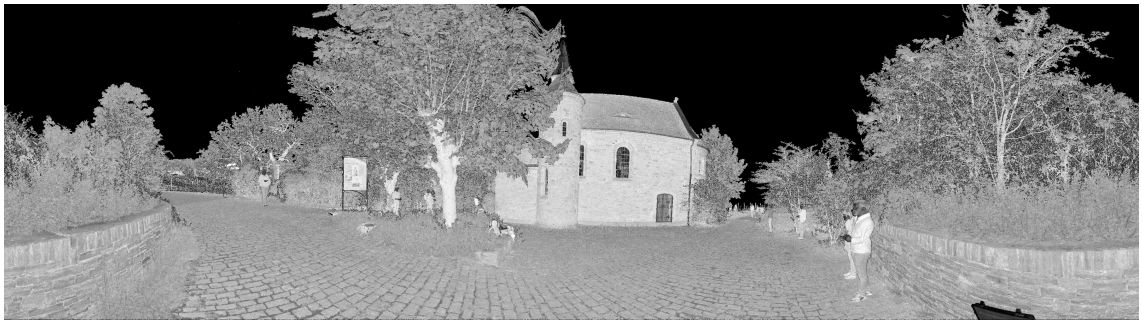


Kuva 5: Pallokoordinaatisto. Pisteen sijainti avaruudessa ilmaistaan korotuskulmalla θ, atsimuuttikulmalla ϕ ja säteellä r.

Laserkeilauksella tuotettuja pistepilviä ei usein käytetä sellaisenaan, vaan niitä täytyy ensin esikäsitellä. Yleisiä esikäsittelyvaiheita ovat panoraamakuvien luominen, usean pistepilven rekisteröinti yhteen koordinaatistoon, poikkeavien pisteiden (engl. *outlier*) poistaminen ja pintojen rekonstruointi.

Yksinkertaisin tapa visualisoida pistepilvi on muodostaa siitä kaksiulotteinen kuva. Pilvestä voidaan luoda panoraamakuva projisoimalla laserkeilaimen ympäröimät pisteet kaksiulotteiselle kuvatasolle.[1] Kuvakoosta riippuen voidaan pistepilvestä karsia huomattava määrä pisteitä, joiden koko olisi liian pieni kuvatasolle projisoituna. Panoraamakuvat toimivat siis myös pistedatan pakkausalgoritmina. Panoraamakuvan pikseleille voidaan määrittää värit joko sen perusteella, kuinka paljon valoa heijastuu takaisin niitä vastaavista pilven pisteistä, kuinka kaukana pisteet ovat keilaimesta tai kuinka korkealla pisteet ovat maanpinnasta. Panoraamakuvaan on helppo soveltaa erilaisia kuvankäsittelyalgoritmeja, kuten piirteentunnistusta (engl. *feature detection*). Kuvassa 6 on esimerkki pano-

[1]Projektio vaikuttaa suuresti panoraamakuvan laatuun. Hyvän yleiskatsauksen erilaisiin projektiohin antaa [14]. Kuvassa 6 on käytetty tasavälistä lieriöprojektiota (engl. *equirectangular projection*).



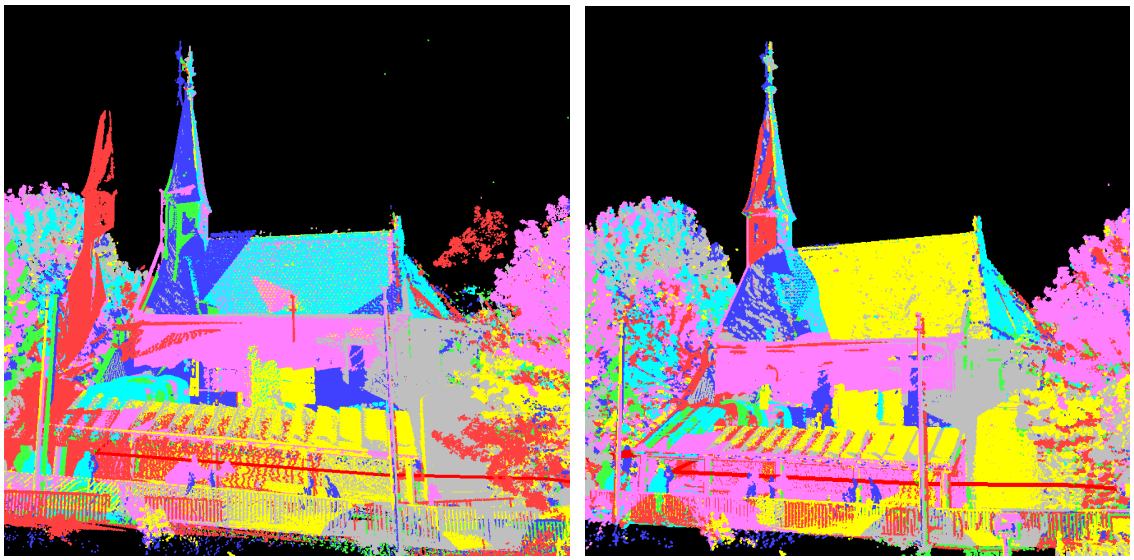
Kuva 6: Saksan Würzburgissa sijaitsevasta Randersackerin Neitsyt Marian kärsimysten kappelista keilatusta pistepilvestä muodostettu panoraamakuva.

raamakuvasta, joka on väritetty laserpurskeiden heijastuksien intensiteetin perusteella.

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

Pistepilviä voidaan rekisteröidä usealla eri tavalla. Joskus keilattavaan kohteeseen asetetaan erityisiä merkkikuvioita, jotka näkyvät useasta keilaimesta. Kun tiedetään merkkien etäisyys ja suunta kustakin keilaimesta, voidaan pistepilvet sovittaa helposti yhteen koordinaatistoon trigonometrian avulla. 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 [15].

Pistepilviä voidaan rekisteröidä myös ilman merkkikuvioita tai käyttäjän apua. Iteratiivinen lähimmän pisteen 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. Yksinkertaisimmillaan pistettä vastaavaksi pisteeksi valitaan toisesta pilvestä se, jonka euklidinen etäisyys edellä mainittuun on pienin. Iteratiivisen algoritmin kunkin transformaation sopivuus lasketaan kaikkien vastaavien pisteiden välisten etäisyyksien muodostamasta vir-



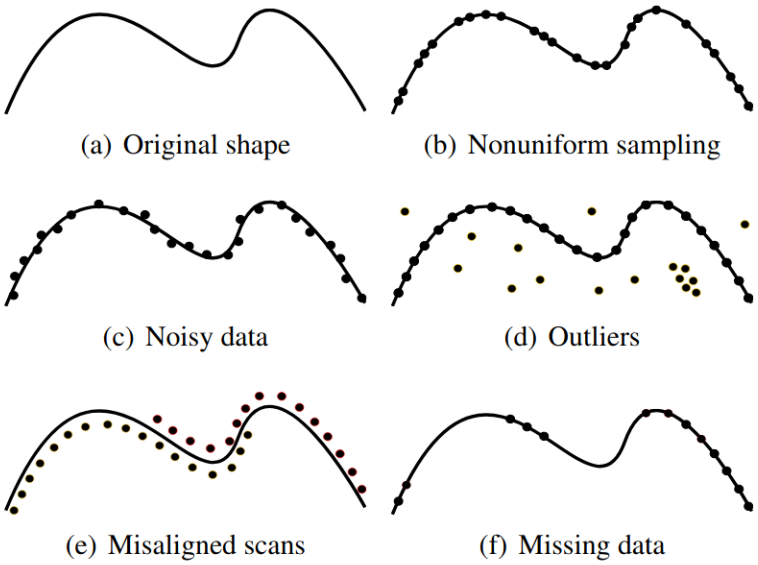
(a) (b)

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

heestä. Algoritmin iteroiminen voidaan lopettaa, kun jokin ennaltamäärätty raja virheelle on alitettu. Kuvassa 7 on ICP-algoritmilla sovitettu yhteen kuusi pistepilveä. [16]

ICP-algoritmi tarvitsee kuitenkin käyttäjältä hyvän alkuarvauksen keilausten sijainneista yhteisessä koordinaatistossa, jotta virhe suppenisi ja pilvien sovittaminen onnistuisi. Se on myös erittäin raskas suorittaa isoille pistepilville. Yleensä ICP-algoritmia käytetäänkin karkeamman rekisteröintialgoritmin suorittamisen jälkeiseen hienosäätämiseen. Karkeampi pilvien yhteensovittaminen tehdään yleensä etsimällä kahdesta pistepilvestä muodostetuista panoraamakuvista toisiaan vastaavia avainpisteitä (engl. *keypoints*). Näiden avainpisteiden sijainneista saadaan harva joukko pisteitä, joiden yhteensovittaminen on paljon helpompaa kuin kokonaisten pistepilvien. Suosittuja tekniikoita avainpisteiden löytämiseen ovat esimerkiksi Harrisin nurkat [17] sekä FAST ja SIFT -piirteet [18][19]. [20]

Joissakin sovelluksissa halutaan luoda pistedatasta kohteen pintoja kuvaava polygoniverkko, jotta visualisointi olisi nopeampaa nykyaikaisilla grafiikkakirjastoilla ja visuaali-



Kuva 8: Kaarevasta pinnnasta (a) keilatussa pistepilvissä esiintyviä mahdollisia virheitä: (b) epätasainen pistetiheys, (c) häiriötä pisteiden sijainneissa, (d) poikkeavia pisteitä, (e) epätarkka rekisteröinti ja (f) puuttuvaa dataa. [22]

nen 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. Kolmiointi luodaan Voronoin diagrammista siten, että pisteiden välillä on kaari, mikäli niitä vastaavat Voronoin solut jakavat Voronoin janan. [21]

Oikeat, laserkeilaimella taltioidut pistepilvet sisältävät kuitenkin usein erilaisia virheitä, kuten kuvassa 8 esitellyt epätasainen keilaus, keilaimen häiriö, poikkeavat pisteet, ongelmat rekisteröinnissä ja puuttuva data. Virheiden johdosta Delaunayn kolmiointi ei ole kovin tehokas algoritmi kolmioverkon muodostamiseen pistepilvestä. Useissa pintojen rekonstruointialgoritmeissa tehdään joitakin oletuksia pistepilven ominaisuuksista.

Etenkin tietokoneavusteisessa suunnittelussa keilauksen kohteet muodostuvat yksinkertaisista geometrisista primitiiveistä, kuten tasoista ja sylintereistä. Esimerkiksi RANSAC-algoritmi [23] sovittelee primitiivejä satunnaistetusti pistepilveen ja arvioi niiden sopivuutta. [22]

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 vaikkapa tuulen heiluttamista puiden lehdistä johtuen on pistepilvissä usein poikkeavia pisteitä. Yksinkertainen tapa havaita ja poistaa tällaisia pisteitä on vertailla pisteiden normaalivektoreita niiden naapuruston normaaleihin ja etsiä poikkeavuuksia.

Pisteen p normaalivektori voidaan selvittää pääkomponenttianalyysillä (engl. *principal component analysis, PCA*). Ensin on etsittävä pilvestä pisteen p k lähintä naapuria, jonka jälkeen naapuruston pisteistä lasketaan ominaisarvot. Kahta suurinta ominaisarvoa vastaavaa ominaisvektoria voidaan käyttää kuvaamaan tasoa, joka sovitetaan naapuruston päälle. Jäljelle jäävä ominaisvektori kuvaa pisteen p normaalia. [24]

2.2 Pistepilvien hyödyntäminen suunnitteluohjelmistoissa

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

Tietokoneavusteisessa suunnittelussa pistepilviä käytetään olemassaolevien rakenteiden taltiointiin. Usein käytetty esimerkki on autoteollisuuden alalta: ryhmä suunnittelijoita kokeilee uutta korimallia rakentamalla prototyyppiauton helposti muovattavasta materiaalista. Kun prototyyppi on todettu aerodynaamiseksi ja miellyttävän näköiseksi, täytyy se saada digitoitua, jotta se voidaan siirtää tuotantoon. Usein yksinkertaisin ja kustannustehokkain tapa on laserkeilata prototyyppi ja jatkokäsitellä pistepilveä niin, että saadaan

luotua haluttu 3d-malli.

Laitossuunnittelussa yleinen ongelma on 3d-mallin vanhentuminen tai sen puuttuminen kokonaan. Vanhempien laitosten suunnitteluun on usein käytetty vain 2d-piirroksia tai pienoismalleja. 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 suunnitteluohjelmalla. [25]

Pistepilvien ja niiden pohjalta muodostettujen 3d-mallien pääasiallinen käyttötarkoitus laitossuunnittelussa on muutostöiden suunnittelu. Muuttuneet tuotantotarpeet tai lainsäädäntö saattavat aiheuttaa tarpeen suurillekin muutoksille. Jos laitokseen täytyy esimerkiksi asentaa savukaasupesuri, voidaan laitoksen sisätiloista luoda pistepilvi, asettaa pesurin 3d-malliobjekti suunnitellulle paikalle ja reitittää tarvittavat hormit ja putket paikalleen. Tämän jälkeen voidaan tarkastaa, osuvatko malliobjektit pistepilven pisteisiin.

Kuten sillanrakennuksessa, myös laitoksia rakentaessa on joskus syytä suorittaa tarkastusmittauksia ja verrata niitä alkuperäiseen 3d-malliin. Laserkeilaus on tähänkin tarkoitukseen sopivia tekniikka, sillä pistepilvien tuottaminen on helppoa ja niitä pääsee tarkastelemaan nopeasti keilauksen jälkeen. Tarkkuudessakaan pistepilvet eivät häviä perinteisemmille mittaustavoille. Nykyaikaisten laserkeilainten tuottamat pistepilvet ovat niin tarkkoja, että niistä voi havaita esimerkiksi putkien roikkumisen ja lämpölaajenemisen [25].

Usein pistepilvellä kuvattu laitos halutaan muuntaa suunnitteluohjelmiston käyttämäksi solidigeometriaksi. Lattiat ja seinät on tasoina helppo asettaa paikalleen, kuten myös suunnitteluohjelmiston komponenttikirjastosta löytyvät laitteet. Suurin työ on yleensä putkistoissa, ilmakanavissa ja kaapeliradoissa. Useat suunnitteluohjelmistot tarjoavat jonkinasteista automatisointia etenkin putkien reititykseen pistepilven päälle. Ohjelmisto voi automaattisesti tunnistaa pilvestä sylintereitä ja asettaa niiden päälle sopivia putkistoosia. Vaihtoehtoisesti käyttäjä voi valita pilvestä muutamia pisteitä ja ohjelmisto laskee

niiden perusteella putken pituuden ja halkaisijan ja asettaa oikean osan paikalleen. Markkinoilla on myös ohjelmistoja, joiden luvataan tuottavan pistepilvestä automaattisesti älykäs 3d-malli komponenttitietoineen [26].

Huomattava osa suunnittelutyöstä koostuu putkistojen reitityksestä ja putkien tunnistamista pistepilvistä onkin tutkittu paljon. Ahmed, Haas ja Haas (2013) ehdottavat Hough-muunnokseen (engl. *Hough-transform*) perustuvaa tekniikkaa putkistojen automaattiseen tunnistukseen pistepilvestä. Tekniikka vaatii toimiakseen käyttäjältä syötteenä laserkeilattujen putkien ominaisuuksia, kuten putkien suunnan pistepilven koordinaatistossa, putkien säteet ja niiden määrän. Pistepilvi jaetaan ensin ohuihin siivuihin, jotka jakavat syötteenä annettuun suuntaan kulkevat putket lyhyihin pätkiin. Pisteet projisoidaan kaksiulotteiselle kuvatasolle jossa ne kuvataan ympyröinä, joiden halkaisija on sama kuin potentiaalisten putkien poikkileikkaus. Näiden ympyröiden leikkauspisteet selvitetään ja putken keskilinjoiksi tulkitaan ne kohdat, joissa on eniten ympyröiden leikkauspisteitä. [27]

Liu et al. (2013) esittelivät samankaltaisen lähestymistavan, jossa ongelma redusoitiin ympyröiden sovittamiseksi pisteisiin. [28] Toteutus pystyi kuitenkin havaitsemaan vain vaakatasossa ja pystysuorassa kulkevat putket. Qiu, Zhou, ja Neumann (2014) esittelivät tekniikan, joka löytää myös viistossa kulkevat putket ja pystyy yhdistämään putkia linjastoiksi asettamalla niiden väliin mutka- ja haaraosia. Algoritmi ei vaadi lähtöoletuksia pistepilvestä mutta pisteiden normaalivektorit täytyy olla selvillä. Pistepilvi jaetaan ensin pienempiin kuutioihin joista etsitään putkien mahdollisia kulkusuuntia valitsemalla satunnaisten pisteiden normaalivektorien kanssa kohtisuora vektori ja laskemalla kuinka monen muun pisteen normaalivektorit ovat sen kanssa kohtisuoria. Vektori tulkitaan putken keskilinjaksi kun tarpeeksi monen pisteen normaalivektorit ovat sen kanssa kohtisuorassa. Seuraavaksi putkien paksuudet selvitetään litistämällä pisteet keskilinjojen suuntaisesti tasoon, minkä jälkeen pistepilven päälle voidaan sovittaa putkia kuvaavia sylintereitä. Lopuksi sylinterien päihin sovitetaan sopivia liitososia. [29]

2.3 Pistepilvien visualisoinnin haasteet

Suurin haaste pistepilvien käsittelyssä ja visualisoinnissa on niiden koko. Nykyaikainen laserkeilain, kuten aiemmin esitetty Leica Geosystemsin RTC360, tuottaa satoja miljoonia pisteitä sisältävän pistepilven. Kun tällaisella keilaimella tehdään useita keilauksia, on pisteiden määrä valtava. Oletetaan esimerkiksi, että suuressa projektissa käytetään pistepilviä, joissa on yhteensä miljardi pistettä. Kun koordinaatit tallennetaan kolmella nelitavuisella 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 metadatan lisäksi. Tämän kokoista pilveä ei haluta pitää kerralla keskusmuistissa, vaan pisteitä tulisi hakea levyltä muistiin vain tarvittaessa.

Pisteiden tallentaminen levylle aiheuttaa kuitenkin ongelmia tiedonsiirron hitauden takia, sillä tiedonsiirtonopeus kiintolevyltä on jopa kaksi kertaluokkaa hitaampi kuin keskusmuistista. Ongelman ratkaisemiseksi käytetään usein ulkoisen muistin algoritmeja (engl. *out-of-core algorithm*), jotka lataavat pisteitä levyltä muistiin isoissa palasissa, mikä on huomattavasti nopeampaa kuin yksittäisten pisteiden lataaminen. [30]

Pistepilvien koon vuoksi ei ole realistista olettaa, että kaikki pisteet voitaisiin visualisoida reaaliajassa.[2] Tästä syystä on kehitetty erilaisia tekniikoita pistepilven harventamiseen ja osittaiseen piirtämiseen. Yksinkertainen tapa harventaa pistepilveä on jakaa sen peittämä alue niin kutsuttuihin vokseleihin (engl. *volume element, voxel*), eli säännöllisiin kuutioihin, ja visualisoimalla vain yksi piste kustakin vokselista. Pilven harventaminen kuitenkin aiheuttaa yksityiskohtien katoamista pilvestä, joten sitä täytyy käyttää sovelluskohteesta riippuen maltillisesti.

Toinen keino vähentää visualisoitavien pisteiden määrää on määrittää pistepilvelle tarkkuustasot (engl. *level-of-detail, LOD*). Tarkkuustasot mahdollistavat pistepilven astettaisen tarkentamisen karkeasta yleiskuvasta yksityiskohtiin. Usein interaktiivisissa oh-

[2]Miellyttävän käyttökokemuksen takaamiseksi pitäisi ruutu päivittää vähintään kymmenen kertaa sekunnissa.

jelmistoissa korkean ruudunpäivitystaajuuden ylläpitäminen vaatii sitä, että pistepilvi piirretään matalimmalla tarkkuudella esimerkiksi käyttäjän pyörittäessä näkymää. Toisaalta kun kamera pysähtyy paikalleen, voidaan visualisointiin käyttää enemmän aikaa ja pistepilveä tarkentaa.

3 Pistepilvien visualisointiin käytetyt tietorakenteet

Edellä mainittuihin haasteisiin on otettu kantaa laajalti alan julkaisuissa. Lupaavin tekniikka tarkkuustasojen muodostamiseen, ulkoisen muistin 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 kulkiessa tarkkuus kasvaa. Hierarkiasta voidaan joko valita tarkkuustaso, joka takaa nopean ruudunpäivityksen tai tarkentaa kuvaa inkrementaalisesti, kunnes visualisointiaika loppuu kesken tai katselupiste siirtyy ja pistepilvi täytyy piirtää uudestaan toisesta kuvakulmasta.

Aihealueen pioneerityönä pidetään Rusinkiewiczin ja Levoyn (2000) QSplatia [31], joka onkin antanut paljon vaikutteita uudemmille tietorakenteille. Dachsbacher, Vogelgsang ja Stamminger (2003) esittelivät peräkkäispistepuut [32], joiden avulla pistepilviä voidaan piirtää hyvin nopeasti näytönohjaimen avulla. Viime vuosina pistepilvien visualisoinnin tutkimuksen kirkkainta kärkeä on edustanut Wienin teknillisen yliopiston tietokonegrafiikan tutkimusyksikkö. Tämän tutkielman päälähteinä käytetään Claus Scheiblauerin (2014) ja Markus Schützin (2016) julkaisuja sisäkkäispistepuista [30][33]. Schütz et al. (2019) ovat esitelleet myös joitakin mielenkiintoisia, ei-hierarkisia tekniikoita [34][35], jotka kuitenkin rajoittuvat näytönohjaimen muistiin mahtuviin pistepilviin. Joitakin tärkeitä ominaisuuksia esitellyistä tietorakenteista on koottu taulukkoon 1.

Edellä mainitut tekniikat on esitelty tarkemmin luvuissa 3.1 - 3.5. Luvussa 3.6 selvitetään laitossuunnitteluohjelmiston asettamia vaatimuksia pistepilvivisualisoijan käyttämälle tietorakenteelle ja todetaan sisäkkäispistepuiden vaikuttavan lupaavilta myös laitossuunnitteluohjelmistossa käytettäväksi.

3.1 QSplat

Yksi ensimmäisistä pistedatan visualisointiin käytetyistä hierarkisista tietorakenteista on Rusinkiewiczin ja Levoyn (2000) esittämä QSplat, joka on kehitetty kolmioverkon visua-

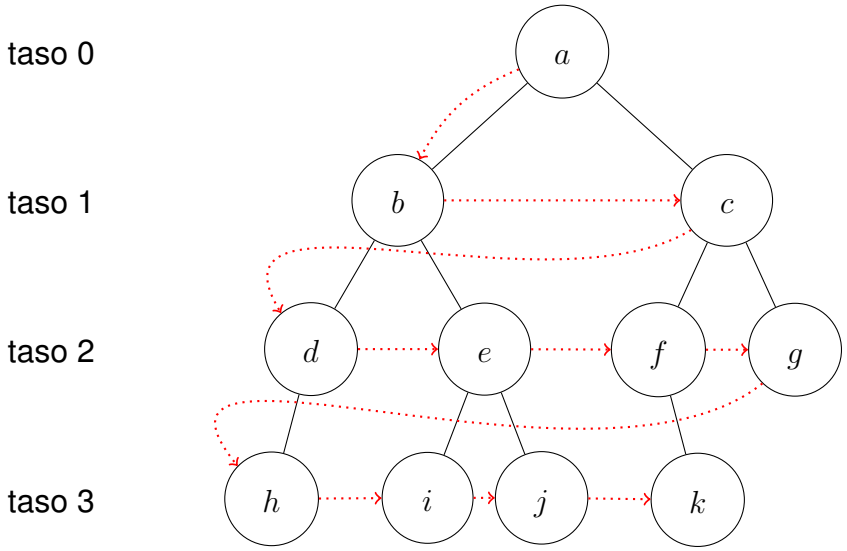
tietorakenne	**tarkkuustasot**	**pistedatan säilytys**	**luo ylimääräistä dataa**
QSplat [31]	valitaan yksi	ulkoinen muisti	kyllä
Peräkkäispistepuut [32]	valitaan yksi	näytönohjaimen muisti	kyllä
Muistioptimoidut peräkkäispistepuut [36]	valitaan yksi	näytönohjaimen muisti	ei
Sisäkkäispistepuut [30][33]	valitaan yksi tai tarkennetaan inkrementaalisesti	ulkoinen muisti	ei
kd-puut [37][38]	valitaan yksi tai tarkennetaan inkrementaalisesti	ulkoinen muisti	ei
Ei-hierarkiset tekniikat [34][35]	valitaan yksi tai tarkennetaan inkrementaalisesti	näytönohjaimen muisti	ei

Taulukko 1: Luvussa 3 esiteltyjen tietorakenteiden ominaisuuksia

lisointiin pisteiden avulla. Tietorakenteen muodostaminen vaatii, että mallin kolmioiden normaalivektorit tunnetaan, joten se ei suoraan sovellu raa'an pistepilvidatan käsittelyyn.[3] QSplatissa on käytetty kuitenkin monia kiinnostavia tekniikoita, joita voi hyödyntää pistepilvidatan käsittelyssä. [31]

QSplat perustuu puurakenteeseen, jonka solmuissa on rajauspalloja (engl. *bounding sphere*). Pallot jakavat avaruutta rekursiivisesti pienempiin osiin siten, että juuren pallo sisältää kaikki mallin kolmiot ja jokainen sisäsolmu jakaa avaruuden keskimäärin neljään osaan. Puun latva saavutetaan, kun avaruuden jakamisen seurauksena jäljelle jää yksi kolmio. Jäljelle jäävästä kolmiosta muodostetaan lehtisolmu, jonka rajauspallo sisältää koko kolmion. Puun visualisointi onnistuu piirtämällä jokaisen pallon kohdalle sopivan kokoi-

[3]Itse asiassa Rusinkiewicz ja Levoy käyttivät laserkeilatusta pistepilvestä muodostettua kolmioverkkoa, jonka tietorakenne esitti yksinkertaistettuna pistedatana.



Kuva 9: QSplatin käyttämän pallopuun läpikäyminen taso kerrallaan muodostaa luonnolliset tarkkuustasot.

nen täplä (engl. *splat*). Puurakenne mahdollistaa myös tehokkaan pisteiden karsimisen niiden näkyvyyden perusteella. Jos solmun rajauspallo ei ole näkökentässä, eivät sen lapsetkaan ole ja haaran läpikäyntiä ei tarvitse jatkaa. [31]

Puurakenne tallennetaan levylle leveysjärjestyksessä (engl. *breadth-first*). Tämän ansiosta puun tasot muodostavat luonnolliset tarkkuustasot: juurisolmun rajauspallo esittää koko mallia, ensimmäinen taso sisältää muutaman pienemmän pallon, ja niin edelleen. Kun tällainen tiedoston sisäinen rakenne yhdistetään ulkoisen muistin tekniikoihin, voidaan täplien piirtäminen aloittaa heti, kun tarpeeksi puun solmuja on ladattu levyltä muistiin. Puuta käydään läpi kunnes solmujen rajauspallot ovat niin pieniä, ettei niiden kohdalle enää kannata piirtää täpliä. Puun rakennetta on havainnollistettu kuvassa 9. [31]

Toinen hyödyllinen QSplatissa käytetty tekniikka on koordinaattien kvantisointi (engl. *quantization*). Kun tarkkuudesta voidaan tinkiä, solmujen rajauspallojen 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

vielä hylätään vanhemman pallon ulkopuolella olevat keskipisteet ja käytetään hakutaulua, voidaan pallon sijainti esittää vain 13:lla bitillä, kun normaali liukulukuesitys vaatisi vähintään 16 tavua. [31]

QSplat onnistui piirtämään 1,5-2,5 miljoonaa pistettä sekunnissa, mikä on sen aikaisella laitteistolla erinomainen tulos [31]. Kuten sanottu, se ei sellaisenaan kuitenkaan sovellu laserkeilattujen pistepilvien käsittelyyn. Pistepilvien pisteiden normaaleja ei yleensä tiedetä, joten ne pitäisi esiprosessointivaiheessa selvittää esimerkiksi luvussa 2.1 esitetyllä tekniikalla.

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

3.2 Peräkkäispistepuut

Dachsbacher, Vogelgsang ja Stamminger (2003) esittelevät niin kutstutun peräkkäispistepuun (engl. *sequential point tree*), joka yrittää hyödyntää QSplatin hierarkian lisäksi näytönohjaimen laskentatehoa. Peräkkäispistepuiden hierarkia litistetään yksiulotteksi taulukoksi, joka voidaan ladata näytönohjaimen muistiin. Visualisointivaiheessa näytönohjaimen tarvitsee vain valita mitkä osat taulukosta piirretään ruudulle. Peräkkäispistepuut siirtävät siis työtä näytönohjaimelle ja jättävät suorittimen vapaaksi muuta laskentaa varten. [32]

Peräkkäispistepuita on käytetty kuvaamaan yksittäisiä objekteja isommassa mallissa. Jokaisesta objektista näytteistetyt pisteet järjestetään pallopuuhun samaan tapaan kuin QSplatissa. Pisteet sijaitsevat puun lehtisolmuissa, joiden rajauspallot ovat lähes yhtäsuuria ja sisäsolmut kuvaavat lapsiensa unionia siten, että niiden rajauspallo juuri ja juuri

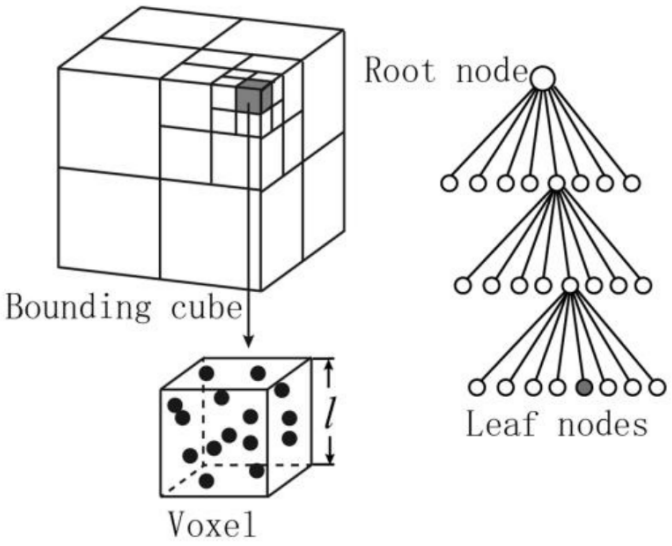
peittää lastensa pallot. Visualisointiin käytetään rajauspallojen kokoisia täpliä joiden väri määräytyy puun haaran alempien solmujen värien keskiarvon perusteella. [32]

Jokaiselle solmulle lasketaan virhe e sen perusteella, kuinka hyvin sen rajauspallo peittää lapsisolmujensa rajauspallot. Ajatuksena on käydä hierarkiaa läpi syvemmälle, jos $e/r < \epsilon$, missä r on etäisyys katselupisteestä solmuun ja ϵ käyttäjän asettama arvo. Tästä johdetaan katseluetäisyydelle rajat r_{min} ja r_{max}, joiden välillä solmu valitaan piirrettäväksi. Visualisointivaiheessa hierarkia litistetään taulukoksi ja pisteet järjestetään solmujen r_{max}-arvon mukaan, jonka jälkeen ne ladataan näytönohjaimen muistiin. Näin näytönohjaimen täytyy käydä taulukkoa läpi vain siihen asti, kun katselupisteen etäisyys r ylittää käsiteltävän pisteen r_{max}-arvon. [32]

Peräkkäispistepuut voidaan visualisoida nopeasti näytönohjaimen tehokkaan käytön ansiosta, mutta niissä on myös heikkouksia. Tietorakenteen vaatimuksena on, että kaikki data mahtuu näytönohjaimen muistiin. Näin on näytönohjaimesta riippuen vain pienillä malleilla ja tilannetta pahentaa se, että peräkkäispistepuut eivät ole kovin säästäväisiä muistin suhteen. Hierarkian jokaisessa sisäsolmussa luodaan QSplatin tapaan lisää dataa, kun lapsisolmujen unionia kuvataan uudella täplällä, jolle tarvitsee tallentaa sijainti, koko ja väri.

Wimmer ja Scheiblauer (2006) esittävät parannuksia peräkkäispistepuihin muistioptimoiduilla peräkkäispistepuilla (engl. *memory optimized sequential point tree, MOSPT*). Uusien täplien luomisen sijaan puun sisäsolmuissa valitaan lapsisolmuista edustaja, joka parhaiten kuvaa sisäsolmusta alkavaa haaraa. Haaran solmujen väreistä lasketaan keskiarvo ja edustajasolmuksi valitaan sellainen, jonka väri on lähinnä keskiarvoa. Olemassa olevien solmujen käyttäminen tarkkuustasojen muodostamiseen on tärkeä oivallus, sillä käsiteltäessä massiivisia pistepilviä tulisi välttää ylimääräisen datan luomista. [36]

Wimmer ja Scheiblauer (2006) määrittävät solmun virheeksi yksinkertaisesti rajauspallon halkaisijan, minkä johdosta kunkin tason jokaisella solmulla on samat r_{min} ja r_{max} -arvot. Tämän vuoksi visualisointivaiheessa ei tarvitse tarkastaa jokaisen solmun r_{max} ar-



Kuva 10: Oktettipuun juurisolmu sisältää kaikki pisteet sisältävän rajauslaatikon. Jokainen sisäsolmu jakaa rajauslaatikkonsa kahdeksaan osaan. [39]

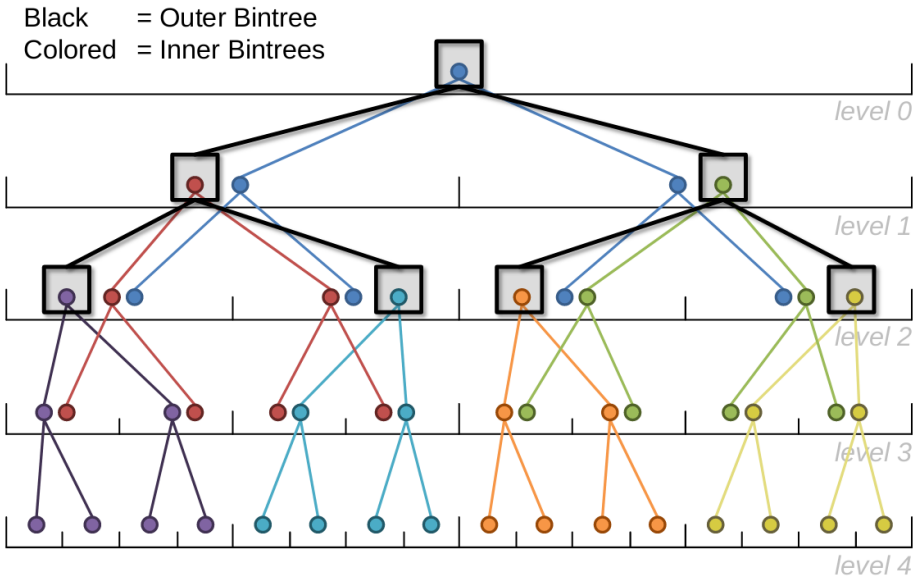
voa, vaan riittää, kun tiedetään mistä pistetaulukon indeksistä alkaa mikäkin hierarkian taso. Tämän jälkeen taulukosta piirretään pisteitä kunnes saavutetaan taso, jolla rajauspallot projisoituisivat alle pikselin kokoisiksi kuvaruudulle. [36]

3.3 Sisäkkäispistepuut

Wimmer ja Scheiblauer (2006) 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*). Oktettipuu[4] on yksinkertainen avaruutta rekursiivisesti jakava tietorakenne, jonka jokainen sisäsolmu jakaa kuvaamansa avaruuden kahdeksaan samankokoiseen osaan. Oktettipuun rakennetta on havainnollistettu kuvassa 10.

Wimmerin ja Scheiblauerin tietorakenteessa oktettipuita on kahdessa tasossa. Ulompaa oktettipuuta käytetään avaruuden jakamiseen, sen tehokkaseen läpikäymiseen ja nä-

[4]Tätä suomennosta käyttää esimerkiksi Davidsson [40]. Vaihtoehtoinen suomennos on kahdeksanpuu.



Kuva 11: Sisäkkäinen binääripuu, jossa sekä ulomman, että sisempien puiden syvyys on kolme. Ulompaa puuta kuvaavat mustat neliöt ja sisempää värikkäät ympyrät. Puista muodostuu viisi tarkkuustasoa. [30]

kökentän 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. [36]

Sisäkkäisistä oktettipuista luodaan tarkkuustasot siten, että sisemmistä puista kerätään pisteitä ulomman puun tasojen mukaan. Tarkkuustasoon kuuluvat pisteet sijaitsevat siis ulomman puun samalla tasolla, mutta useiden sisempien puiden eri tasoilla. Tarkkuustasojen muodostumista on havainnollistettu kuvassa 11. Puut tallennetaan levylle tarkkuustaso kerrallaan, mikä mahdollistaa ulkoisen muistin algoritmien käytön. Visualisoitaessa tarvitsee levyltä lukea pisteitä vain haluttuun tarkkuustasoon asti, eikä loppuja pisteitä tarvitse ladata muistiin. [36]

Scheiblauer (2014) jalostaa sisäkkäisten oktettipuiden ideaa väitöskirjassaan esittelemällä muokattavat 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. [30]

MNO:n rakenne eroaa sisäkkäisistä oktettipuista siten, että sisemmät puut korvataan säännöllisillä kolmiulotteisilla ruudukoilla, joihin pisteet tallennetaan. MNO:n rakentaminen alkaa juurisolmusta, joka vastaa kaikki pisteet peittävää avaruutta. Solmun sisältämä ruudukko jakaa solmua kuvaavan avaruuden osan 128^3 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 järkevää luoda uusia solmuja puuhun. Kun ennaltamäärätty vähimmäismäärä pisteitä on kertynyt ylimääräisten pisteiden taulukkoon, luodaan solmulle lapsisolmuja ja sijoitetaan ylimääräiset pisteet niihin. Ruudukkoon sijoitettavien pisteiden määrälle on hyvä asettaa myös yläraja. [30]

Jokainen tietorakenteen solmu tallennetaan omaan tiedostoonsa levylle, josta niitä ladataan muistiin visualisointivaiheessa tarvittaessa. Visualisointialgoritmiin kuuluu käyttäjän asettama pistebudjetti, joka asettaa ylärajan yhdessä ruudunpäivityksessä piirrettävien pisteiden määrälle.[5] Tätä rajaa säätämällä käyttäjä saa jonkinlaisen kontrollin ruudunpäivitystaajuuden suhteen. Vaihtoehtoisesti visualisointia voidaan jatkaa usean ruudun ajan, jolloin saavutetaan miellyttävä inkrementaalinen tarkennus. [30]

Tietorakenne 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 vanhempia kunnes jokin niistä muodostaa tarvittavan kokoisen avaruuden, ja piste lisätään sen ruudukkoon. Kun puun vanhan juuren yläpuolelle luodaan uusia solmuja, jäävät 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. [30]

[5] Scheiblauer testasi pistepilvivisualisoijaansa asettamalla rajan vain sataantuhanteen pisteeseen.

Scheiblauer oli ottanut MNO:ta kehittäessään vaikutteita Wandin et al. (2008) esittämästä oktettipuusta, jonka sisäsolmuihin kuuluu myös ruudukko. Wandin et al. tietorakenteessa pisteet tallennetaan kuitenkin ruudukon sijaan lehtisolmuihin joihin mahtuu kuhunkin enintään satatuhatta pistettä. Sisäsolmujen ruudukoihin tallennetaan rakennusvaiheessa kunkin solun läpi kulkeneiden pisteiden lukumäärä, minkä perusteella tarkkuustasot muodostetaan. Wandin et al. tietorakenne käyttää MNO:n tapaan ulkoista muistia ja mahdollistaa tehokkaat lisäys- ja poisto-operaatiot. [41]

Markus Schütz (2016) jatkoi Wimmerin ja Scheiblauerin työtä esittelemällä opinnäytetyössään verkkoselaimessa ajettavan Potree-nimisen pistepilvivisualisoijan. Potreen käyttämä tietorakenne perustuu Scheiblauerin muokattaviin sisäkkäisiin oktettipuihin, mutta hierarkian rakennusvaiheessa kiinnitetään huomiota pisteiden tasaiseen jakautumiseen solmujen välille. Oktettipuun sisäsolmujen ruudukoihin hyväksytään uusia pisteitä vain, jos ne ovat tarpeeksi kaukana muista ruudukon pisteistä. Lehtisolmut hyväksyvät ennaltamäärättyyn rajaan saakka kaikki pisteet, kunnes ne muutetaan sisäsolmuiksi ja liian lähekkäin olevat pisteet jaetaan uusien lapsisolmujen kesken. [33]

Potree käyttää ulkoista muistia tehokkaasti ja pystyy käsittelemään jopa 640 miljardia pistettä sisältäviä pistepilviä.[6] Rakennusvaiheessa oktettipuun solmuja tallennetaan tasaisin väliajoin levylle, jottei muisti täyttyisi. Kun jokainen solmu tallennetaan omaan tiedostoonsa, on yksittäisten solmujen tallentaminen ja lukeminen levyltä helppoa. Massiivisia pistepilviä kuvaavat hierarkiatkin voivat olla satojen megatavujen kokoisia. Schütz ratkaisee suurten hierarkioiden nopean lataamisen verkon yli jakamalla senkin puurakenteeseen. Näin voidaan välttää sekä turhien pisteiden, että näkökentän ulkopuolella olevien hierarkian haarojen lataaminen muistiin. [33]

Potreen visualisointialgoritmi priorisoi niitä hierarkian solmuja, jotka ovat lähellä katselupistettä ja joiden kuvaruudulle projisoitu koko on suurin. Visualisoinnin suoritusky-

[6]Kyseinen pistepilvi (Actueel Hoogtebestand Nederland, ANH2, `http://ahn2.pointclouds.nl/`) kuvaa koko Alankomaiden valtiota ja se vaatii 7,68 teratavua tallennustilaa. Potreen tietorakenteessa pistepilvi jakautui 13:lle tasolle ja 38:aan miljoonaan solmuun.

kyä voidaan säädellä Scheiblauerin toteutuksen mukaisesti käyttäjän asettamalla pistebudjetilla. Schütz on kehittänyt Potreehen myös näytönohjaimella ajettavan algoritmin mukautuvaan pisteiden koon määrittämiseen; pistepilven harvemmissa osissa piirretään pisteet suurempina, jottei reikiä esiintyisi. [33]

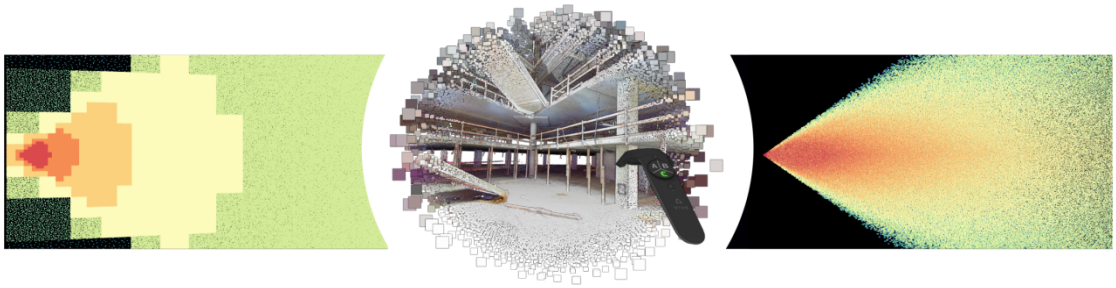
3.4 kd-puut

Suosittujen oktettipuiden lisäksi on pistepilvien visualisoinnissa käytetty kd-puita (engl. *k-dimensional tree, kd-tree*)[7]. Oktettipuusta poiketen kd-puut eivät jaa pistepilveä yhtäsuuren 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. [37]

Richter, Discher ja Döllner (2014) kehittivät massiivisten ulkoilmakeilausten visualisointiin kd-puuhun perustuvan pistepilvivisualisoijan, joka käytti luokiteltua pistepilvidataa. Pisteet oli jaettu objektiluokkiin, kuten kasvillisuus, vesi tai rakennus, joille jokaiselle rakennettiin oma kd-puu. Visualisointivaiheessa kd-puista lähetetään solmuja näytönohjaimelle niiden kuvaruudulle projisoidun koon määräämässä järjestyksessä niin, että jokaisella objektiluokalla oli oma rajansa muistinkäytölle. [37]

Futterlieb, Teutsch ja Berndt (2016) käyttivät samankaltaista kd-puuta luokittelemattomalle pistedatalle. Edellä mainitusta toteutuksesta poiketen tarkkuustaso valitaan etsimällä puusta yksi taso, jolta saadaan piirrettyä sopiva määrä pisteitä. Varmistaakseen, että ruudulle saadaan aina jonkinlainen tarkkuustaso visualisoitua nopeasti Futterlieb et al. pitivät kahta miljoonaa pistettä näytönohjaimen muistissa jatkuvasti riippumatta siitä, olivatko ne näkyvissä. [38]

[7] Kolmiulotteisten 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. [34]

3.5 Ei-hierarkiset tekniikat

Pistepilvistä voi muodostaa tarkkuustasoja myös ilman hierarkista tietorakennetta. Schütz, Krösl ja Wimmer (2019) esittivät virtuaalitodellisuuslaseille suunnatun pistepilvivisualisoijan, joka hierarkian muodostamisen sijaan käy koko pistepilveä läpi ja muodostaa siitä noin viiden ruudun välein uuden, sopivan tiheästi näytteistetyn osajoukon. Tämä jatkuviksi tarkkuustasoiksi (engl. *continuous LOD*) kutsuttu tekniikka ei siis jaa pisteitä tietorakenteen solmuihin, joilla on diskreetit tarkkuustasot, vaan pisteiden etäisyys toisistaan vaihtelee sen perusteella, kuinka kaukana ne ovat kamerasta. [34]

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

Schütz et al. (2019) esittelivät hiljattain myös toisen ei-hierarkisen tekniikan. Pisteitä ladataan näytönohjaimelle, joka asettaa niitä verteksipuskureihin satunnaisessa järjestyksessä. Pisteiden piirtäminen aloitetaan heti kun riittävä määrä pisteitä on saatu ladattua ja kuvaa tarkennetaan seuraavilla ruuduilla samalla kun näytönohjaimen muistiin ladataan lisää pisteitä. Puskurien täyttäminen satunnaisessa järjestyksessä saa aikaan kuvan

miellyttävän tarkentumisen. [35]

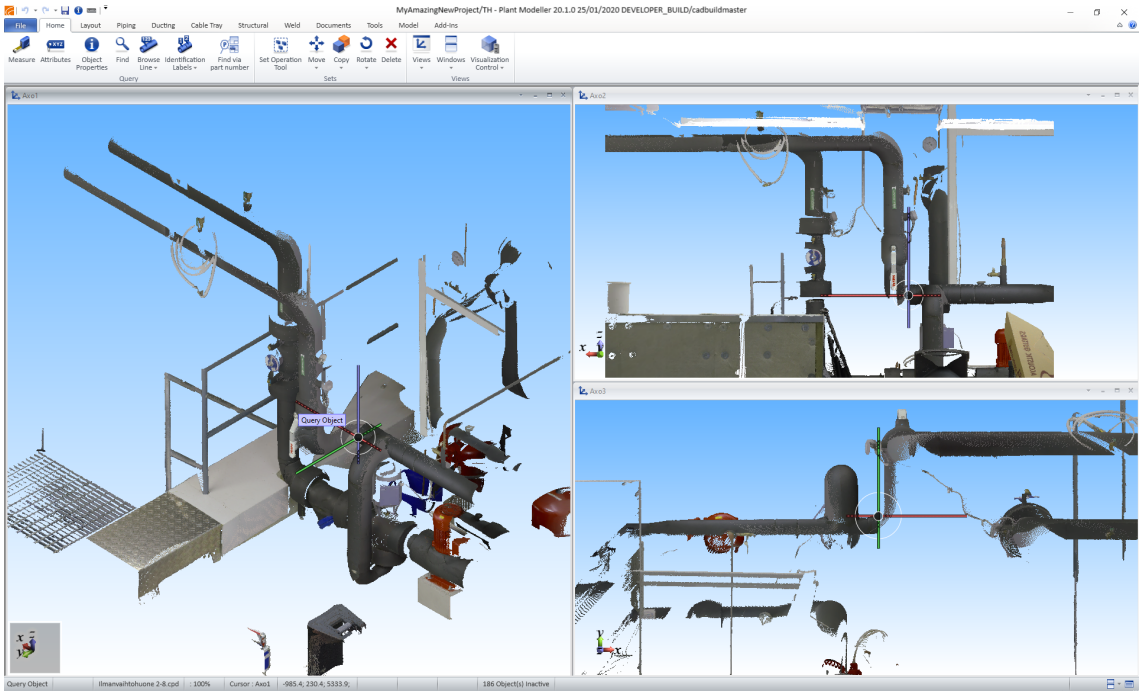
Nämä kaksi ei-hierarkista tekniikkaa toimivat vain silloin, kun pistepilvet mahtuvat kokonaisuudessaan näytönohjaimen muistiin. Tämä on kuitenkin vähenevissä määrin rajoittava tekijä, sillä uusimmissa näytönohjamissa voi olla jopa 48 gigatavua muistia, johon mahtuu jo erittäin suuria pistepilviä [42].

3.6 Laitossuunnitteluohjelmistoon soveltuva tietorakenne

Laitossuunniteluohjelmisto esittää pistepilvivisualisoijan käyttämälle tietorakenteelle tiettyjä vaatimuksia. Selvitetään näitä vaatimuksia käyttötapausten perusteella. Kaksi yleistä käyttötapausta pistepilvien kanssa työskenneltäessä ovat mallintaminen ja katselu.

Mallintamisessa tärkeässä roolissa on suunnittelijan käyttämät näkymät ja pistepilven rajaaminen. Yleensä suunnittelija käyttää muutamaa koordinaattiakselien suuntaista näkymää 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. Kuvassa 13 on esitetty tyypillinen näkymien asettelu. 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 piirrettäisiin. Lisäksi pistepilvi tulee voida piirtää useaan eri näkymään samanaikaisesti.

Mallinnustyössä käytetään usein hyväksi mittaustyökalua. 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 lähimmäksi kursoria projisoidun pisteen. Käyttäjän kannalta olisi miellyttävää, jos mittausoperaatioita tehtäessä ei tarvitsisi odottaa, kun pistepilven miljoonien pisteiden joukosta etsitään juuri kursorin alla oleva piste. Yksittäisten pisteiden



Kuva 13: Tyypillinen mallinnustyössä käytetty näkymien asettelu CADMATIC Plant Modeller laitossuunnitteluohjelmistossa. Pistepilvi on Elomatic Oy:n omaisuutta.

hakeminen pilvestä täytyy siis olla nopeaa. Mallinnustyö ja etenkin mittaaminen asettavat ohjelmistolle vaatimuksen tarkkuudesta. Laitossuunnitteluohjelmistossa käytetään yleensä perusyksikköinä millimetrejä, joten pistepilvessä ei saisi esiintyä senttimetrien virheitä.

Toinen 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 visualisoitavia pisteitä on niin paljon, etteivät ne mahdu kerralla keskusmuistiin tai näytönohjaimen muistiin. Yleensä käyttäjä myös liikuttaa näkymää mallin ympäri enemmän kuin mallinnustyössä, joten pistepilvivisualisoinnin suorituskyky ja tarkkuustasot ovat entistäkin

tärkeämpiä.

Molemmissa käyttötapauksissa täytyy ohjelmistoon ladata useasta laserkeilauksesta muodostettuja pistepilviä. Usein mallinnustyössä käyttäjä joutuu välttävän ruudunpäivitystaajuuden takaamiseksi piilottamaan niiden keilauksien pisteitä, jotka eivät sillä hetkellä ole välttämättömiä työn kannalta. Ohjelmiston käytettävyys parantuisi huomattavasti jos suorituskyky ei kärsisi vaikka pistepilvi sisältäisi usean laserkeilauksen pisteitä. Tällöin käyttäjän ei tarvitsisi säädellä yksittäisten keilausten näkyvyyttä, vaan sopivan siivun pilvestä saisi näkyviin esimerkiksi rajaustyökalulla.

Tämän tutkielman osana kehitetään laitossuunnitteluohjelmistolle optimoitu hierarkinen tietorakenne pistepilvien käsittelyyn. Esitetään tälle 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 vaatimaa tallennustilan määrää voidaan laskea harventamalla sen tiheästi näytteistettyjä osia.
4. Yhden pistepilven on voitava sisältää usean laserkeilauksen pisteet.
5. Käyttäjän on voitava määrittää pilvestä alueita, joiden sisältävien tai ulkopuolelle jäävien pisteiden ominaisuuksia, kuten näkyvyyttä tai väriä, voidaan muuttaa.
6. Pilvestä on voitava nopeasti ja tarkasti valita yksittäisiä pisteitä.
7. Pistepilvessä ei saa esiintyä yli millimetrin suuruisia virheitä.

Luvussa 3.3 esitelty Potree on osoittautunut massiivisten pistepilvien interaktiivisen visualisoinnin olevan mahdollista jopa verkkoselaimessa, jossa etenkin tiedonsiirtonopeus rajoittaa visualisoinnin suorituskykyä. Tarkastellaan siis sisäkkäispistepuiden soveltuvuutta laitossuunnitteluohjelmistoon.

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. Scheiblauerin muokattavien sisäkkäisten oktettipuiden jokainen solmu sisältää ruudukon, johon pisteet sijoitetaan. Mitä syvemmällä tasolla solmu on, sitä pienempiä ruudukon solut ovat. Vaatimuksen 3 esittämä pilven harvennus onnistuu asettamalla puulle enimmäissyvyys ruudukon koon mukaan ja hylkäämällä lehtisolmuissa kaikki pisteet, jotka tulisi lisätyksi jo varattuun soluun. Nämä ominaisuudet mahdollistavat myös vaatimuksen 4 mukaisesti usean laserkeilauksen sovittamisen yhteen tietorakenteeseen niin, ettei visualisoinnin suorituskyky kärsi.

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 5 ja 6 voidaan kuitenkin vastata sisäkkäisillä oktettipuilla. Scheiblauer ja Schütz eivät tiivistäneet pistepilviä, joten niiden tarkkuus ei kärsinyt. Näin myöskään vaatimus 7 ei tuota ongelmia.

Sisäkkäispistepuut näyttävät siis soveltuvan myös laitossuunnitteluohjelmiston tarpeisiin. Scheiblauerin ja Schützin tietorakenteiden käyttämät ruudukot mahdollistavat kuitenkin myös joitakin laitossuunnittelusovelluksille tärkeitä optimointeja, kuten luvussa 4.1 esitellyn pistedatan kompression.

4 Pistepilven visualisointi sisäkkäispistepuilla

Luvussa 3.6 todettiin Scheiblauerin ja Schützin ehdottamien sisäkkäispistepuiden soveltuvan laitossuunnitteluohjelmistossa käytettävän pistepilvivisualisoijan tarpeisiin. Sisäkkäispistepuiden rakenne mahdollistaa seuraavaksi esiteltävän yksinkertaisen tekniikan pistedatan kompressioon. Tämän jälkeen esitellään algoritmeja sisäkkäispistepuun rakentamiseen, visualisointiin ja pisteiden valitsemiseen.

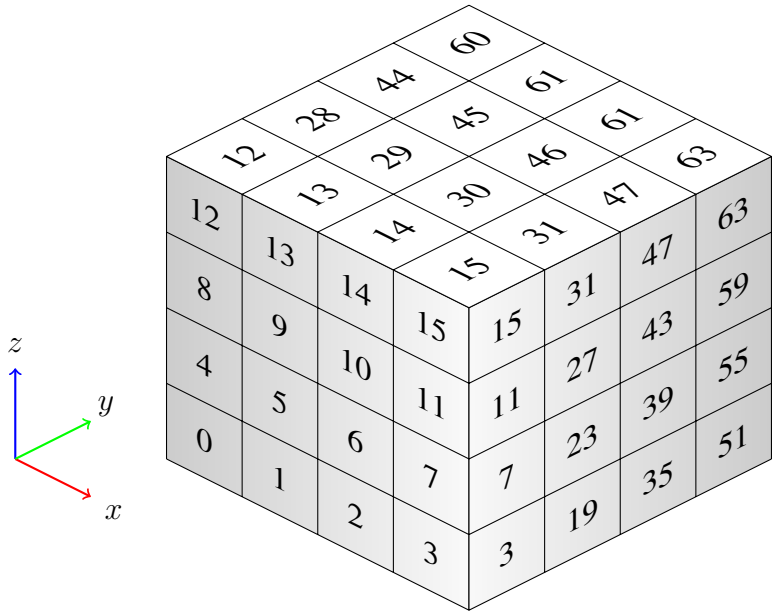
4.1 Pistedatan esitysmuoto

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 liukuluvuilla ja väri RGB-koodauksella kolmella tavulla:

```
struct Point {
    float x;
    float y;
    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 vaatisi siis 16 gigatavua muistia. Muuttamalla pisteiden esitysmuotoa voidaan pistepilviä tiivistää ja joitakin operaatioita nopeuttaa.

Puun rakennusvaiheessa pisteet lisätään jokaisessa solmussa olevaan kolmiulotteiseen



Kuva 14: 64:n solun ruudukko, jonka indeksointi alkaa vasemmasta alakulmasta

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 numerointi ruudukolle on esitetty kuvassa 14. Numerointi alkaa vasemmasta alakulmasta indeksistä nolla ja etenee vasenkätisen koordinaatiston mukaisesti.

Sisäkkäispistepuiden käyttämä ruudukko mahdollistaa yksinkertaisen pakkausalgoritmin. Puun solmuihin on tallennettu rajauslaatikko, joka määrittää myös ruudukon mitat ja sijainnin. Ruudukkoon lisättävien pisteiden absoluuttinen sijainti voidaan unohtaa ja käyttää sijainnin tallentamiseen pisteen suhteellista sijaintia ruudukossa. Yksinkertaisimmillaan voidaan kustakin pisteestä 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.

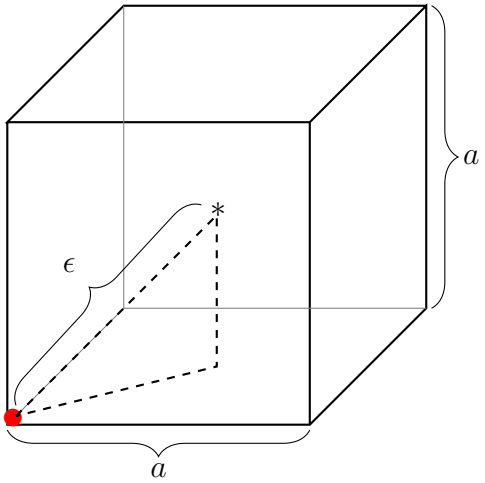
Esitysmuotoa voidaan tiivistää vielä tästäkin. Pisteen etäisyyttä suhteessa ruudukkoon voidaan esittää kolmella tavulla niin, että jokainen tavu kuvaa koordinaattiakselien suuntaisten askelten määrää ruudusta 0 lähtien. Yhden tavun esittämä enimmäisarvo on 255, joten ruudukossa voi olla enintään 255^3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

Usein laitossuunnittelussa käytetyissä laserkeilaimissa ei käytetä värikameraa antamaan pisteille värejä, vaan väri-informaatio johdetaan keilaimeen heijastuvan valon määrästä. 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 {
    unsigned char dx;
    unsigned char dy;
    unsigned char dz;
    unsigned char intensity;
}
```

eli tarvitaan vain neljä tavua tilaa ja edellä mainittu pistepilvi saadaan tiivistettyä neljännekseen alkuperäisestä koostaan.

Kun pisteen tarkka sijainti unohdetaan ja se ilmaistaan suhteessa ruudukkoon, syntyy pistepilveen virheitä. Suurinta mahdollista virhettä on havainnollistettu kuvassa 15, kun piste visualisoidaan solun keskipisteenä, vaikka se oikeasti sijaitsisi aivan sen nurkassa. Jos oletetaan, että ruudukon solut ovat kuutioita, saadaan enimmäisvirhe laskettua helposti



Kuva 15: Suurin mahdollinen ruudukossa esiintyvä virhe ϵ. Kuutio kuvaa ruudukon solua ja asteriski sen visualisointiin käytettävää keskipistettä. Soluun lisätty punainen piste on juuri ja juuri solun sisällä.

Pythagoraan lauseella muotoon

$$\epsilon = \frac{a\sqrt{3}}{2}. \tag{4}$$

Vaatimus 7 esitti virheen enimmäissuuruudeksi yhtä millimetriä. Oktettipuu jakaa rajauslaatikon sivun kahtia joka tasolla ja ruudukko sen edelleen pieniin soluihin. Jos oletetaan 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 lisätyt pisteet täyttävät vaatimuksen 7. Tämä ei tarkoita sitä, että puu voisi sisältää pisteitä vain tasolle 11 ja sitä syvemmälle. Pisteitä voi hyväksyä suuriinkin soluihin, jos ne sattuvat osumaan tarpeeksi lähelle sen keskipistettä. Käytännössä suurissa pistepilvissä on pisteitä yleensä niin tiheästi, että puun ylimmillekin tasoille tulee tallennetuksi pisteitä.

Pisteiden sijainnin suhteellinen esitysmuoto nopeuttaa myös pistepilvien käsittelyä. Laitossuunnittelussa pistepilveä joudutaan usein sovellukseen latauksen jälkeen liikutta-

maan 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 oktettipuun solmujen rajauslaatikoilla, joista tiivistetyn pisteen sijainti johdetaan. Tämä johtaa merkittävään parannukseen käyttökokemuksessa, sillä satojen miljoonien pisteiden kirjoittaminen levylle voi kestää useita minuutteja.

4.2 Tietorakenteen rakentaminen

Edellä esitellyn oktettipuun rakentaminen suoritetaan kahdessa vaiheessa. Ensin käydään läpi kaikki syötteenä annettujen laserkeilausten pisteet ja selvitetään niille rajauslaatikko. Tämän jälkeen luodaan tyhjä juurisolmu, jonka rajauslaatikkona toimii äsken laskettu, kaikki pisteet sisältävä tila. 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 solmuun, 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 määrälle kannattaa asettaa yläraja, jotta saavutettaisiin sopiva haarautuminen.

Puun syvyyttä voi rajoittaa asettamalla ruudukon soluille vähimmäismitat. Yleensä laserkeilaimen lähellä olevat pinnat tulevat näytteistetyksi hyvin tiheästi ja tilannetta pahentaa, jos useat keilaimet ovat mitanneet samoja pintoja tiheästi. Algoritmin 2 rivillä 3 tarkastetaan, ylittäisikö uusi lapsisolmu puun enimmäissyvyyden. Kun ennaltamäärätylle pohjatasolle hyväksytään kaikki pisteet, kunhan niitä vastaava ruudukon solu on vapaa, saadaan pistepilvelle tehokas ja globaali enimmäistiheys. Tällä tavalla pilveä saadaan harvennettua tehokkaasti ja vaatimus 3 voidaan tyydyttää.

Jotta säästettäisiin satojen tuhansien tiedostojen avaamisen ja sulkemisen vaatima ai-

Algoritmi 1: RakennaOktettipuu

Syöte : Joukko pistepilviä P

Tuloste: Pisteet sisältävän oktettipuun juurisolmu s

```
1  // Selvitetään ensin pilvien rajauslaatikkojen unioni
2  L ← tyhjä rajauslaatikko
3  for pistepilvi pc ∈ P do
4      for piste p ∈ pc do
5          if p on laatikon L ulkopuolella then
6              // Kasvata laatikkoa L niin, että se sisältää pisteen p
7              L ← bbox(L, p)
8          end
9      end
10 end
11 // Lisätään pisteet oktettipuuhun
12 s ← juurisolmu, jonka rajauslaatikko on L
13 for pistepilvi pc ∈ P do
14     for piste p ∈ pc do
15         LisääSolmuun(s, p)
16     end
17 end
18 return s
```

ka, voidaan laitossuunnitteluohjelmistossa tietorakenne tallentaa yhteen tiedostoon. 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

Algoritmi 2: LisääPisteSolmuun

Syöte : Puun solmu s,

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

$i \leftarrow$ sen ruudukon solun indeksi, jossa piste sijaitsee

$h \leftarrow$ hajautustaulu, johon solmun s pisteet tallennetaan

if *Solmun s syvyys = puun enimmäissyvyys* **then**

  if $i \notin h$ **then**

    lisää i ja (r, g, b) hajautustauluun h

  else

    Hylkää piste p

  end

else if *s on täynnä* **then**

  $l \leftarrow$ uusi lapsisolmu

  LisääPisteSolmuun(l, p)

else if $i \in h$ **then**

  $l \leftarrow$ uusi lapsisolmu

  LisääPisteSolmuun(l, p)

else if $\epsilon >$ *ennalta määrätty enimmäisvirhe* **then**

  $l \leftarrow$ uusi lapsisolmu

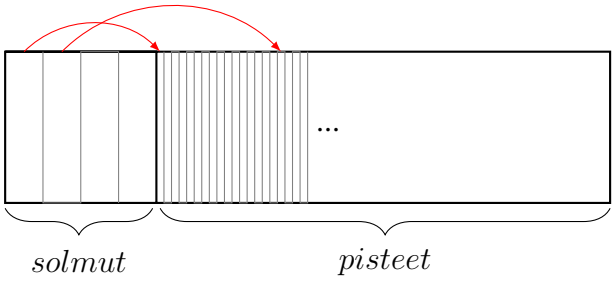
  LisääPisteSolmuun(l, p)

else

  `// Solmussa` s `ja sen ruudukon solussa` i `on tilaa, eikä virhe` ϵ `ole liian suuri. Piste voidaan lisätä solmuun` s`.`

  lisää i ja (r, g, b) hajautustauluun h

end



Kuva 16: Tiedostoon tallennetaan ensin puun solmut, joiden jälkeen kaikki pisteet ovat peräkkäin taulukossa. Punaiset nuolet kuvaavat solmuihin tallennettuja indeksejä pistetaulukkoon, josta siihen kuuluvat pisteet alkavat.

16. 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 viidennessä 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;
    float max_x;
    float max_y;
    float max_z;
    unsigned char depth;
    unsigned char location_code[];
    unsigned int num_points;
    unsigned int point_index;
}
```

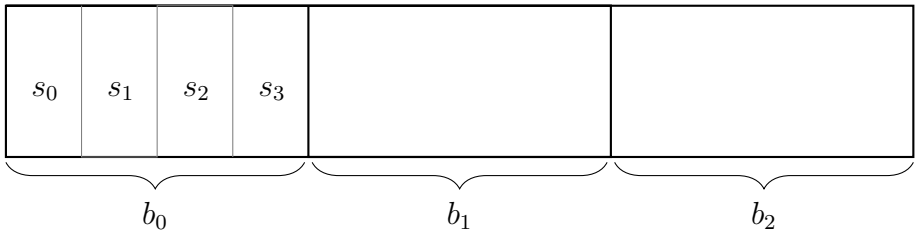
Nelitavuisilla liukuluvuilla ja kokonaisluvuilla vaatii solmu siis tallennustilaa $33+d$ tavua, missä d on solmun syvyys puussa.

Edellä kuvattu rakennusalgoritmi pitää koko oktettipuun muistissa ennen kuin pisteet kirjoitetaan levylle. Massiivisten pistepilvien tapauksessa keskusmuisti saattaa täyttyä vaikka pilveä olisi harvennettu ja pisteet esitetty tiiviimmässä muodossa. Muistin täyttyessä käyttöjärjestelmä alkaa pitää osaa prosessin muistiavaruudesta kiintolevyllä sivutuksen (engl. *paging*) avulla [43]. Jatkotutkimuksen aiheeksi jää selvittää, onko nopeampaa antaa käyttöjärjestelmän siirtää sivuja kiintolevylle, vai esimerkiksi tallentaa ensin pisteitä väliaikaisiin tiedostoihin niiden lisäysjärjestykessä, mistä ne kopioidaan oikeassa järjestyksessä lopulliseen tiedostoon.

4.3 Visualisointi

Nopein tapa visualisoida pistepilvi on pitää kaikki pisteet näytönohjaimen muistissa ja joka ruudunpäivityksellä piirtää kaikki näkymässä näkyvät pisteet. Vaikka koko pistepilvi mahtuisi näytönohjaimen muistiin, täytyy laitossuunnitteluohjelmistossa visualisoida muutakin grafiikkaa. Näytönohjaimen muistia voidaan säästää käyttämällä niin kutsuttua puskurivirtaa (engl. *buffer streaming*). Yleinen ongelma grafiikan piirtämisessä on se, että näytönohjain prosessoi dataa nopeammin kuin suoritin ehtii sille syöttää. Puskurivirran ajatuksena on pitää näytönohjain mahdollisimman toimeliaana jakamalla puskuri, 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 osiota ilman, että sen tarvitsisi odottaa lainkaan hidasta tiedonsiirtoa. [44]

Kuvassa 17 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 vajaiksi. Puskureita kuitenkin vaihdetaan tiuhaan, joten ei ole järkevää käyttää aikaa puskurin täyttöasteen maksi-



Kuva 17: Pistedatan visualisoinnissa 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 tarkoituksena on vähentää tiedonsiirrosta aiheutuvaa latenssia.

mointiin.

Tietorakenteen visualisointia testatessa puskurivirta ei kuitenkaan tuottanut tarpeeksi miellyttävää visuaalista lopputulosta. Etenkin pilveä pyöriteltäessä ja sen läpi liikuttaessa puskurivirralla ehdittiin pilvestä piirtää vain hyvin karkea tarkkuus niin, että ruudunpäivitystaajuus pysyi interaktiivisena. Ratkaisuna tähän näytti toimivan hyvin Futterliebin et al. [38] innoittamana toinen pistepuskuri, jossa säilytetään karkeaa yleiskuvaa pilvestä. Tämä yleiskuvapuskuri pidetään näytönohjaimen muistissa, josta se on nopea piirtää jokaisella ruudunpäivityksellä.

Algoritmissa 3 yleiskuvapuskuri täytetään ensimmäisellä visualisointikerralla 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 jokaisella ruudunpäivityksellä osoittautui liian aikaavieväksi.

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

Algoritmi 4 lisää kunkin solmun pisteet täyttövuorossa olevaan puskurivirran osioon. Puskuriosion täyttyessä vaihdetaan täytettäväksi seuraava osio. Puskurivirtaa käyttäessä täytyy varmistaa, ettei sama osio ole sekä suorittimen kirjoitettavana, että näytönohjaimen

Algoritmi 3: PiirräSisäkkäispistepuu

Syöte : Sisäkkäispistepuu P

```
1  b_yleiskuva ← pistepuskuri, jota pidetään näytönohjaimen muistissa
2  if Ensimmäinen visualisointikerta then
3      while b_yleiskuva ei ole täynnä do
4          // Käydään puuta läpi taso kerrallaan
5          for Solmu s ∈ P do
6              Lisää solmun s pisteet puskuriin b_yleiskuva
7          end
8      end
9  Lähetä b_yleiskuva näytönohjaimelle piirrettäväksi.
10 // Loput pisteet visualisoidaan puskurivirralla
11 S ← näkyvissä olevat puun solmut, joita ei lisätty puskuriin b_yleiskuva
12 Järjestä S ruudulle projisoidun koon mukaan
13 PiirräPuskurivirta(S)
```

luettavana. Tämä tapahtuu lähettämällä näytönohjaimelle jokaisen puskuriosion täyttymisen jälkeen synkronointiobjekti (engl. *sync object*). Kun uutta osiota otetaan kirjoitettavaksi, tarkastetaan, että näytönohjain on merkannut kyseisen osion synkronointiobjektin käsitellyksi. [45]

Algoritmi 3 suorittaa pistepilvelle näkyvyyskarsintaa (engl. *visibility culling*) valitessaan rivillä 11 visualisoitavaksi 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 visualisoidaan ensin.

Yksityiskohtien karsinta on suosittu nopeutustekniikka tietokonegrafiikassa. Ajatuksena on visualisoida vain tärkeimmät objektit ja jättää kaukana olevat tai pienet objek-

Algoritmi 4: PiirräPuskurivirta

Syöte : Sisäkkäispistepuun solmujoukko S

$b_0, b_1, b_2 \leftarrow$ kolmeen osioon jaettu pistepuskuri
$i \leftarrow 0$ `// Täytettävän puskuriosion indeksi`
for *Solmu* $s \in S$ **do**
    if *Puskuriosiossa* i *ei ole tilaa solmun* s *pisteille* **then**
        Lähetä puskuriosio i näytönohjaimelle piirrettäväksi
        Lähetä osion i synkronointiobjekti näytönohjaimelle
        $i = (i + 1) \bmod 3$
        Odota, että näytönohjain on käsitellyt osion i synkronointiobjektin
    end
    Lisää solmun s pisteet puskuriosioon i
end

tit käsittelemättä, jotta ruudunpäivitystaajuus pysyy interaktiivisena. Yksinkertainen tapa karsia yksityiskohtia on järjestää objektit niiden kuvaruudulle projisoidun koon mukaan ja piirtää niitä tiettyyn rajaan asti, tai kunnes aika loppuu kesken. Akenine-Möller et al. [46] esittävät kuvaruudulle projisoidun objektin koon arviolle kaavaa

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

jossa n on katselupisteen etäisyys kuvatasosta, r objektin rajauspallon säde ja $\mathbf{c}$ keskipiste, $\mathbf{d}$ on normalisoitu katsomissuunta, ja $\mathbf{v}$ katselupiste. [47]

Mikko Yllikäinen huomautti pro gradu -tutkielmassaan [47], ettei tarkkaa arvioita objektien koosta tarvitse selvittää, jos halutaan selville vain suuruusjärjestys. Yllikäinen yk-

sinkertaistaa kaavan muotoon

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

Yllikäinen nimittää tällä kaavalla muodostetun suuruusjärjestyksen käyttämistä yksityiskohtien karsinnassa kontribuutiokarsinnaksi (engl. *contribution culling*). On huomattava, että tällainen karsinta on tarkoitettu käytettäväksi perspektiiviprojektiota käyttävissä maisemissa. Paralleeliprojektiomaisemissa katseluetäisyys ei vaikuta objektien kokoon. [47]

4.4 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 pisteen 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 häiriöstä aiheutunut poikkeava piste, joka sattuu olemaan juuri kameran edessä. Yksinkertainen tapa välttää tällaisen pisteen valitseminen olisi hylätä sellaiset puun solmut, joissa säteen lähellä on vain yksi tai muutama piste. Toinen vaihtoehto olisi kerätä useita kandidaattipisteitä ja hylätä varmuuden vuoksi muutama katselupistettä lähinnä oleva piste.

Oktettipuun rakenne auttaa myös vaatimuksen 5 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, onko solmun rajauslaatikko valintalaatikon sisällä. Vain siinä tapauksessa, että rajauslaatikko on vain osittain valintalaatikon sisällä, tarvitsee tarkastus tehdä kaikille pisteille. Valintalaatikoita

voidaan pitää muistissa visualisointivaiheessa ja jokaisen solmun kohdalla tarkistaa sen sisältyvyys valintalaatikoihin.

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 piirrettäessä tarkastetaan, osuuko sen sijainti valintaoktettipuun solmuihin ja päätetään sen perusteella, hylätäänkö piste vai ei. [30]

Alustavien tulosten perusteella pisteiden valinta oli riittävän nopeaa ilman Scheiblauerin ehdottamia valintaoktettipuita. Jatkotutkimuksen aiheeksi jää selvittää, minkälainen ero suorituskyvyssä on edellä esitetyn, yksinkertaisen valintatekniikan ja valintaoktettipuiden välillä.

5 Tietorakenteen arviointi

Luvussa 4 esitettiin sisäkkäispistepuun solmujen sisältämien ruudukoiden mahdollistama yksinkertainen kompressiotekniikka ja algoritmeja puun rakentamiseen ja visualisointiin. Ensin mitataan sisäkkäispistepuun rakentamisen, tallentamisen ja läpikäynnin suorituskykyä ja arvioidaan kompression vaikutusta siihen. Tämän jälkeen mitataan pistepilvien visualisointinopeutta eri tekniikoilla käyttäen maiseman läpi kulkevia kamera-ajoja. Testikoneessa on Intel i7-8850H -suoritin, 32 gigatavua keskusmuistia, SSD-levy ja Nvidia Quadro P2000 -näytönohjain.

Tietorakenteen rakentamista, tallentamista ja muistiin lataamista arvioitiin kolmella pistepilvellä. *Konehuone*-pilvi sisälsi 20 keilausta, jotka veivät 18,7 gigatavua tilaa tallennettuna pakkaamattomaan tekstitiedostoon. Keilauksista muodostettiin puu, jossa oli 546 572 600 pistettä 528 017:ssa solmussa, jotka jakautuivat yhdeksälle tasolle. *Toimisto* sisälsi 8,43 gigatavua pisteitä 21 keilauksesta ja siitä muodostetussa puussa oli 240 727 221 pistettä 608 002:ssa solmussa 11:llä tasolla. *Pumput* oli testattavista pienin, vain 41:n megatavun kokoinen pistepilvi. Siinä oli 5 keilausta, joista rakennetussa puussa oli 1 213 990 pistettä 4561:ssä solmussa seitsemällä tasolla.

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

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

Taulukoissa 2, 3 ja 4 on mitattu testipilvien rakentamiseen, tallentamiseen ja pisteiden läpikäyntiin kuluva aika ja binääritiedoston koko käyttäen kolmea luvussa 4.1 käytettyä pisteiden esitysmuotoa: maailmakoordinaatit ja väri (16 tavua), ruudukon solun indeksi ja väri (8 tavua), sekä solun suhteelliset koordinaatit ja laserkeilaimeen takaisin heijastuneen

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 3: 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

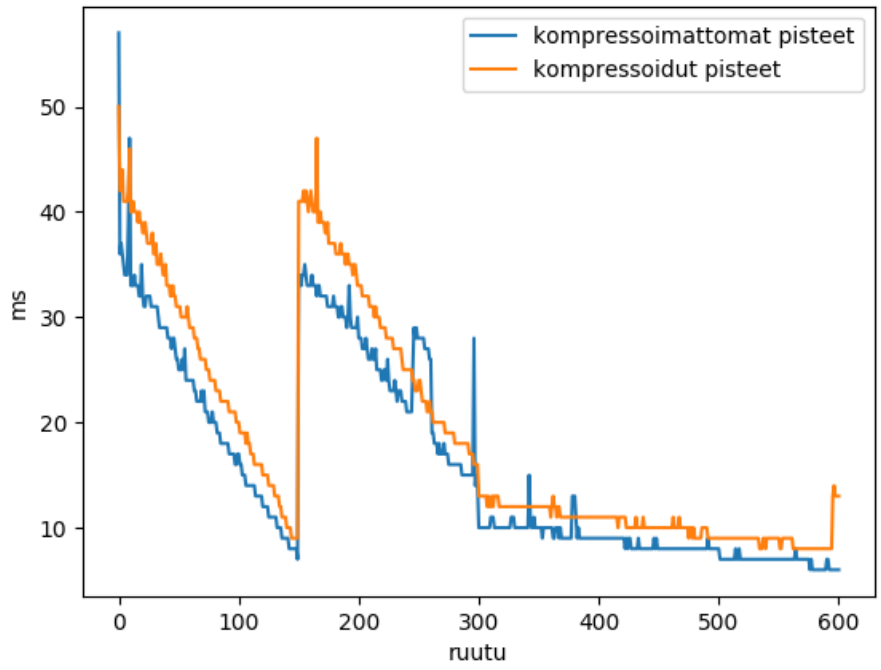
Taulukko 4: Pumput-pilvestä muodostetun oktettipuun rakentaminen ja pisteiden läpikäyminen

valon 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 rakennettaessa 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.

Tietorakenteen visualisointia arvioitaessa käytetään kahta pistepilveä. *Ilmanvaihtohuone* on keilattu Elomatic Oy:n Jyväskylän toimiston ilmanvaihdon konehuoneesta ja siinä on 13 keilausta, joista muodostetussa puussa on 506 366 789 pistettä 267 641 solmussa kahdeksassa tasossa. *Worksite*-pilvi on Leica Geosystemsin testidataa, joka sisältää 7 keilausta, joiden 53 881 180 pistettä jakautuu 543 105 solmuun yhdeksälle puun tasolle.

Arvioidaan ensin luvussa 4.1 esitellyn kompression vaikutusta pistepilven visuali-



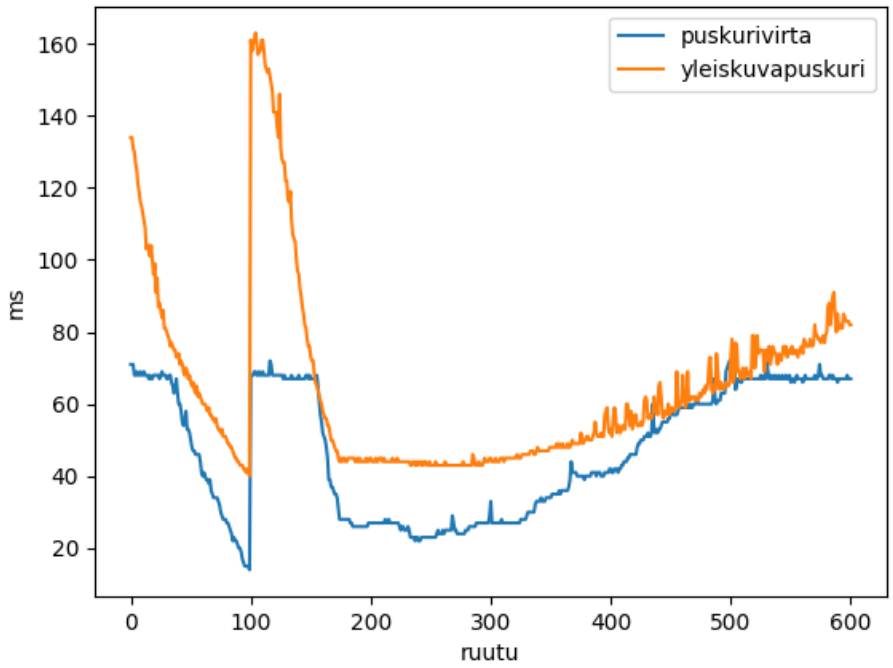
Kuva 18: Worksite-pilven kahden miljoonan pisteen piirtämiseen vaadittu aika millisekunteina käyttäen kompressoimattomia 16:n tavun pisteitä ja kahdeksaan tavuun kompressoituja pisteitä.

sointiaikaan. Kuvassa 18 on esitetty kaavio kahden miljoonan pisteen piirtämisen vaatimasta ajasta kamera-ajon jokaisella ruudunpäivityksellä. Sininen viiva kuvaa piirtoaikaa kompressoimattomilla pisteillä ja oranssi viiva värit säilyttävällä kompressiolla. Visualisoinnissa on käytetty luvussa 4.3 esiteltyä puskurivirta-algoritmia.

Kamera-ajo alkaa maiseman reunalta ja kulkee työmaan ohi lähestyen sen reunaa siten, että näkyvissä olevien puun solmujen määrä laskee tasaisesti. Tämä näkyy myös kuvassa 18 ruudun visualisointiajan laskiessa. Näkymäkarton sisällä olevien solmujen määrä kasvaa äkkinäisesti noin 150:nnen ruudun kohdalla, kun kamera kääntyy niin, että koko pilvi on näkyvissä. Lopuksi kamera lähestyy vastakkaista seinää ja näkyvissä olevien solmujen määrä laskee.

Kuvaajasta huomataan, että kompressoimattomien 16-tavuisten pisteiden piirtäminen

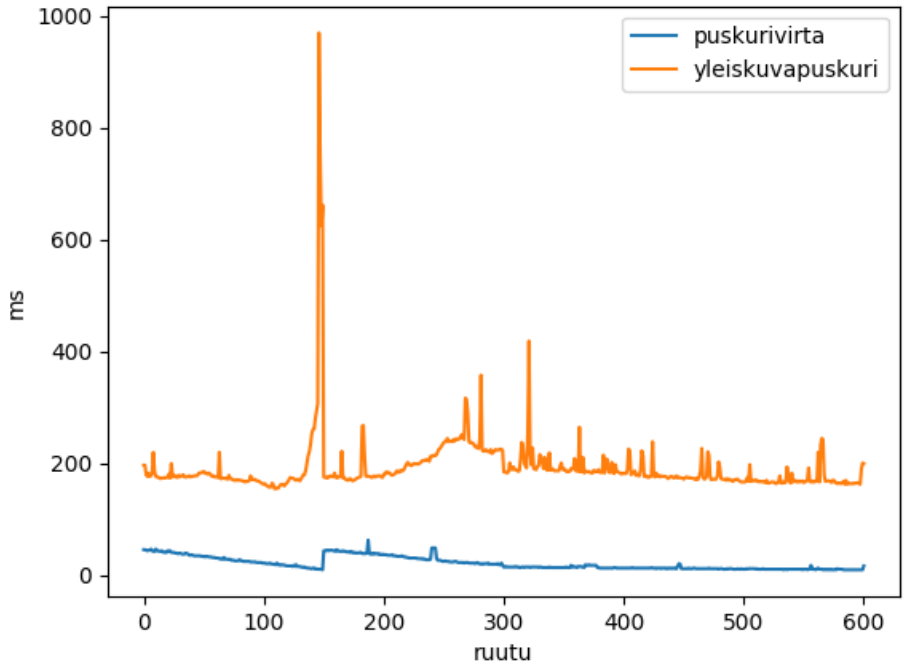
on jonkin verran nopeampaa kuin kompressoitujen kahdeksantavuisten pisteiden. Eron selittää kompression avaamiseen vaadittu laskenta. Jokaisesta kompressoidusta pisteestä etsitään kompressoidun pisteen indeksiä vastaava ruudukon solu ja lasketaan sen keskipiste. Visualisointiajan ero on kuitenkin pieni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



Kuva 19: Ilmanvaihtohuone: kahden miljoonan pisteen piirtämiseen vaadittu aika millisekunteina kahdella eri visualisointialgoritmilla.

Luvussa 4.3 esiteltiin puskurivirran lisäksi yleiskuvapuskuria käyttävä algoritmi, joka pitää osaa pistepilvestä näytönohjaimen muistissa. Kuvassa 19 on mitattu kahden miljoonan pisteen piirtoaikaa ilmanvaihtohuone-pilvessä. Sininen viiva kuvaa piirtoaikaa käytettäessä pelkkää puskurivirtaa ja oranssin viivan kuvaamassa mittauksessa on ensin visualisoitu yleiskuvapuskuri, minkä jälkeen jäljelle jäävät puun solmut on järjestetty kuvaruudulle projisoidun koon mukaan ja piirretty puskurivirralla. Yleiskuvapuskurissa on puun neljä ensimmäistä tasoa, joissa on yhteensä 286 solmua ja niissä 723 834 pistettä.

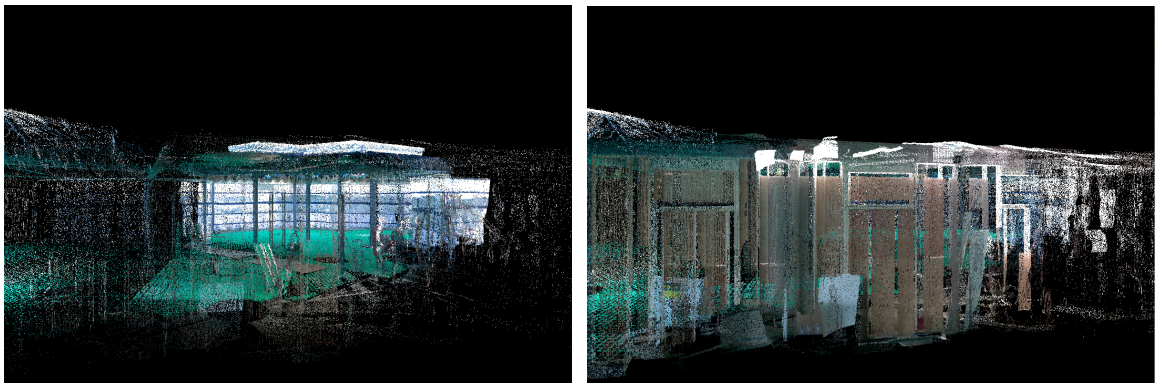
Ilmanvaihtohuoneen kamera-ajo alkaa huoneen reunalta kameran osoittaessa vastakkaiselle seinälle. Kamera liikkuu kohti seinää, jolloin visualisoitavien solmujen määrä vähenee. Kameran saavutettua vastakkaisen seinän noin sadan ruudun jälkeen se kääntyy ympäri osoittamaan huoneen poikki. Näkyvissä olevien solmujen äkkinäinen kasvaminen näkyy jyrkkänä piikkinä kuvassa 19. Tämän jälkeen kamera lähestyy taas seinää ja piirtoaika vähenee. Lopuksi kamera peruuttaa pois päin seinästä ja näkyvillä olevien solmujen kasvava määrä pitkittää ruutujen visualisointia.



Kuva 20: Worksite-pilven kahden miljoonan pisteen piirtämiseen vaadittu aika millisekunteina kahdella eri visualisointialgoritmilla

Kuvassa 20 on tehty vastaavat mittaukset worksite-pilvelle. Oranssin viivan kuvaamassa visualisointitavassa on yleiskuvapuskurissa puun viisi ylintä kerrosta, 1862 solmua ja 481 663 pistettä.

Mittauksista selviää, että pelkän puskurivirran käyttäminen puun visualisoinnissa on selkeästi nopeampaa kuin yleiskuvapuskurin ja puskurivirran yhdistelmällä. Tämä ero se-

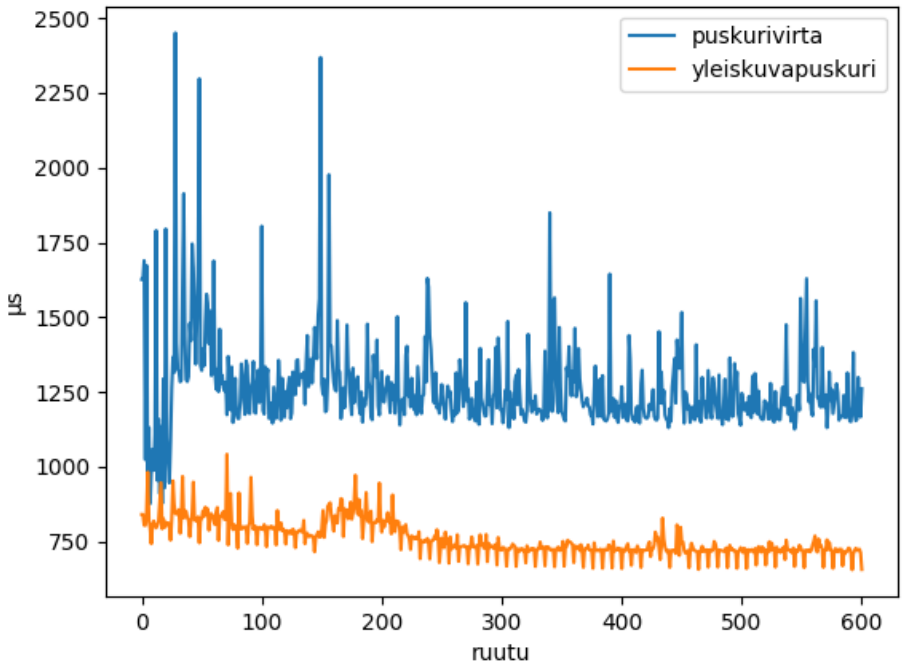


(a) (b)

Kuva 21: Worksite-pilvi visualisoitu kahdella miljoonalla pisteellä käyttäen (a) pelkkää puskurivirtaa ja (b) yleiskuvapuskurin ja puskurivirran yhdistelmällä. Oikeanpuoleisessa kuvassa oktettipuun solmujen järjestämisen ansiosta kameraa lähellä oleva seinä on piirretty korkeammalla pistetiheydellä kuin vasemmassa kuvassa. Pistepilvi on Leica Geosystemsin omaisuutta.

littyy puun solmujen järjestämiseen vaaditulla ajalla. Vaikka järjestäminen vie arvokasta laskenta-aikaa, voidaan sen katsoa parantavan lopputuloksen laatua. Kuvassa 21 vasemmalla puolella näkyy, kuinka puskurivirta on piirtänyt koko näkyvillä olevan pistepilven samalla pistetiheydellä ja taaimmainen seinä näyttää tarkemmalta kuin kameraa lähempänä oleva. Oikeanpuoleisessa kuvassa puun solmut on yleiskuvan piirtämisen jälkeen järjestetty ruudulle projisoidun koon mukaan, minkä seurauksena etualalla oleva seinä näkyy selvästi.

Kameran liikkuessa pistepilven ohi riittää usein piirtää vain karkea yleiskuva pilvestä. Kun yleiskuvan pisteet pidetään näytönohjaimen muistissa, on niiden piirtäminen jokaisella ruudunpäivityksellä nopeaa. Kuvassa 22 on verrattu yleiskuvan piirtämistä kun yleiskuvapuskuri on valmiiksi näytönohjaimen muistissa siihen, kun pisteet ladataan keskusmuistista puskurivirralla näytönohjaimelle. Yleiskuvapuskurin tapauksessa piirtoaika on mitattu piirtokomennon suorittamisesta siihen, että näytönohjain on saanut kaikki pisteet piirrettyä ja merkattua synkronointiobjektin käsitellyksi. Näin kaikki pisteet on varmas-



Kuva 22: Worksite-pilvestä muodostetun puun viiden ylimmän tason visualisointiin käytetty aika mikrosekunteina puskurivirralla ja kun pisteet ovat valmiiksi yleiskuvapuskurissa näytönohjaimen muistissa.

ti piirretty ruudulle ajastimen pysähtyessä. Pisteiden lataaminen levyltä ja kompression purkaminen on näissä mittauksissa jätetty pois.

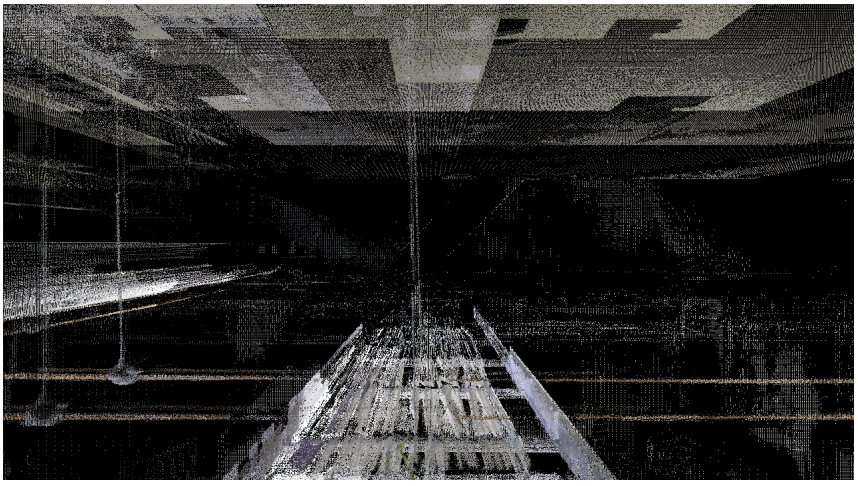
6 Jatkotutkimusaiheita

Luvussa 5 selvisi, että pistedatan kompressoiminen johtaa pienempien tiedostokokojen lisäksi myös oktettipuun rakentamisen nopeutumiseen. Toisaalta kompression purkaminen vie laskenta-aikaa visualisointivaiheessa. Laitossuunnitteluohjelmistossa 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 näytönohjaimen muistiin.

Oktettipuun solmujen järjestäminen ruudulle projisoidun koon mukaan vie runsaasti laskenta-aikaa, minkä johdosta luvussa 4.3 esitetty visualisointialgoritmi suoriutui heikosti suoraviivaiseen puskurivirralla piirtämiseen verrattuna. Voidaan kuitenkin katsoa visualisoidun pistepilven korkeamman tiheyden kameran lähellä olevan tärkeämpää kuin absoluuttinen pisteiden määrä. Solmujen järjestämistä voisi myös nopeuttaa helposti säikeistämällä visualisointialgortimi niin, että yksi säie valikoi puusta solmuja prioriteettijonoon, josta toinen säie ottaa aina suurimman prioriteetin omaavan solmun piirrettäväksi.

Pistebudjetin käyttö pistepilveä visualisoitaessa mahdollistaa interaktiivisen ruudunpäivitysnopeuden, mutta huonontaa piirretyn kuvan laatua. Kuvassa 23 näkyy ilmanvaihtohuoneen katossa ikäviä tarkkuustasojen eroja. Kun puusta piirretään solmuja niiden kuvaruudulle projisoidun koon mukaan eikä taso kerrallaan, voi kuvassa esiintyä suuria tiheyseroja. Tiheyseroja voisi vähentää ja kuvan laatua parantaa implementoimalla esimerkiksi Schützin [33] ehdottaman muokkautuvan pistekoon algoritmin.

Laitossuunnitteluohjelmistossa pistepilven visualisointia voidaan kuitenkin jatkaa monen ruudun ajan, eikä pistebudjetin käyttäminen ole tarpeen. 3d-maisema pistepilvineen ja 2d-grafiikka, kuten kursori ja muut avustimet, voidaan piirtää näytönohjaimelle eri puskureihin. Näin on mahdollista piirtää uudestaan vain kursori, kun käyttäjä liikuttaa hiirtä, minkä jälkeen pistepilven visualisointia voidaan jatkaa siitä mihin viimeksi jäätiin. Oktettipuun solmujen järjestäminen tärkeyden mukaan on silti tarpeen. Kuten kuvasta 21



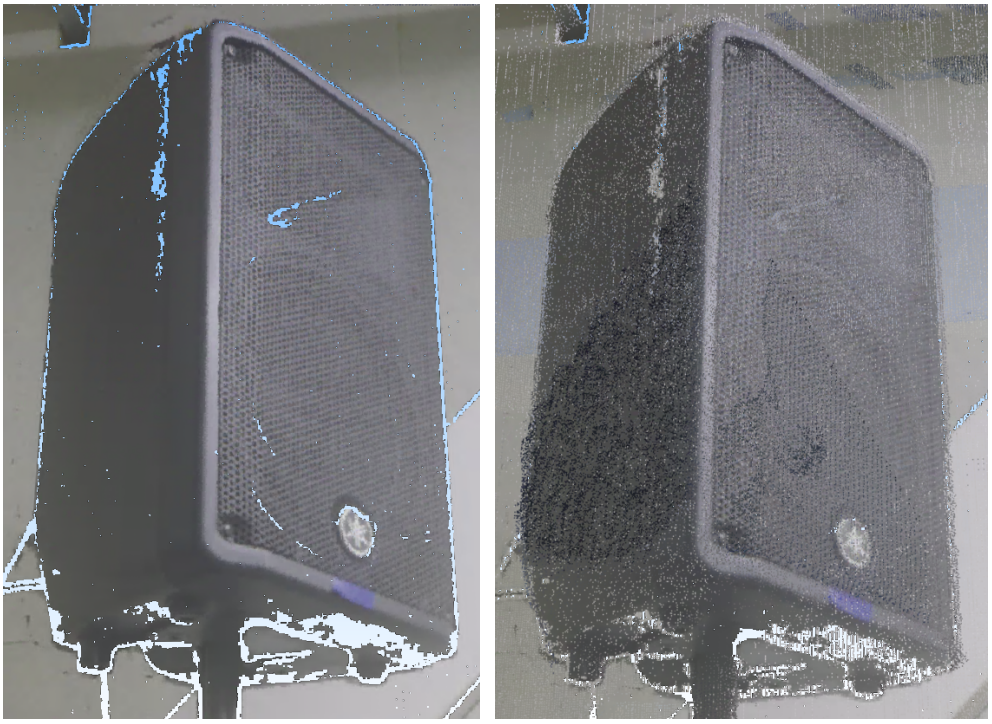
Kuva 23: Kahden miljoonan pisteen budjetin käyttäminen aiheuttaa ilmanvaihtohuonepilvessä häiritseviä tarkkuustasojen välisiä eroja.

näkyy, järjestämisen ansiosta katselupistettä lähellä olevat mielenkiintoiset alueet tarkentuvat nopeammin.

Usealla laserkeilauksella mitattujen pisteiden yhdistäminen yhteen pistepilveen vaikuttaa negatiivisesti etenkin panoraamakuvien laatuun. Saman pinnan pisteet tallentuvat eri värisinä riippuen mistä kohtaa ne on mitattu. Tämä johtuu katselupisteiden välisistä eroista valaistuksessa, ja aiheuttaa kuvassa 24 havainnollistettua ikävää häiriötä. Panoraamakuvaa piirtäessä olisikin hyödyllistä tietää, kuinka kaukaa kuvatasolle projisoitu piste on mitattu. Näin voitaisiin priorisoida pisteitä, jotka ovat lähellä keilaimen sijaintia.

Tutkielmassa esitelty oktettipuu ei suoraan sovellu putkien automaattiseen sovittamiseen pistepilveen luvussa 2.2 esitellyillä tekniikoilla. Oktettipuu toki jakaa pistepilviä pienempiin osiin joita on helpompi käsitellä, mutta ongelmaksi muodostuu pisteiden jakautuminen puun jokaiselle tasolle. Jotta päästäisiin käsiksi tietyn alueen pisteisiin, täytyy käydä läpi kaikki solmut polulla juuresta alueen peittämiin lehtiin. Tehokkaampaa olisi, jos kaikki pisteet olisivat lehtisolmuissa, jolloin tarkasteltavissa solmuissa ei olisi halutun alueen ulkopuolisia pisteitä. Automaattista putkenreititystä varten olisikin ehkä kannattavaa järjestää pistepilvi johonkin toiseen tietorakenteeseen.

Hierarkisen tietorakenteen rakentaminen vie aikaa eikä sen visualisointi ole yhtä nope-



(a) (b)

Kuva 24: Yksityiskohta panoraamakuvasta, joka on muodostettu (a) yhden laserkeilauksen sisältävästä ja (b) useita keilauksia sisältävästä pistepilvestä. Samojen pintojen mitatut värit vaihtelevat keilainten välillä. Pistepilvi on Elomatic Oy:n omaisuutta.

aa kuin pisteiden suoraviivainen piirtäminen. Laitossuunnitteluohjelmistossa käytettävien pistepilvien kokoluokka on harvoin teratavuja, joten muistien kasvaessa voisi olla kannattavaa hierarkisten tietorakenteiden kehittämisen sijaan keskittyä harventamaan pistepilveä niin, että se mahtuu näytönohjaimen muistiin ja käyttää jotakin ei-hierarkista tekniikkaa, joka käyttää näytönohjaimen rinnakkaislaskentatehoa mahdollisimman tehokkaasti.

7 Yhteenveto

Tässä tutkielmassa keskityttiin pistepilvien visualisointitekniikoihin erityisesti laitossuunnitteluohjelmiston tarpeisiin. Pistepilvet ovat laserkeilaimilla mitattuja, suuria pistejoukkoja, jotka kuvastavat mitattavan kohteen pinnanmuotoja. Pistepilviä käytetään muun muassa arkeologiassa, rakentamisessa ja erilaisissa suunnittelutehtävissä. Laitossuunnittelussa pistepilviä käytetään useimmiten muutostöiden yhteydessä. Laserkeilauksella saadaan kohteesta tuotettua kustannustehokkaasti ajantasalla oleva tarkka 3d-malli, joka on välttämätön suurempien muutostöiden onnistumisen kannalta.

Laitossuunnitteluohjelmistossa käytetyt pistepilvet kattavat usein laajoja alueita ja sisältävät niin paljon pisteitä, ettei kaikkia ehditä piirtämään ruudulle jokaisella ruudunpäivityksellä, eikä kokonaisia pistepilviä haluta pitää keskusmuistissa tai näytönohjaimen muistissa. Näihin haasteisiin on vastattu hierarkisilla tietorakenteilla, jotka mahdollistavat pistepilven asteittaisen lataamisen ja visualisoinnin eri tarkkuuksilla.

Luvussa 3 esiteltiin tietorakenteita, joista useat perustuivat avaruutta pienempiin osiin jakaviin puihin, joiden sisäsolmut näytteistävät pistepilveä eri tarkkuuksilla. Qsplat [31] käytti pistejoukkojen rajauspalloista koostuvaa puuta joka visualisoitiin rajauspallojen kokoisilla täplillä. Samaa ajatusta jatkettiin peräkkäispuissa [32][36], jotka hyödynsivät lisäksi näytönohjaimen laskentatehoa. Sisäkkäispistepuut [30][33] mahdollistivat pistepilven asteittaisen tarkentamisen ja jopa pistedatan lataamisen verkon yli. Peräkkäis- ja sisäkkäispistepuut perustuvat avaruuden jakamiseen samankokoisiin palasiin jokaisella puun tasolla. Vaihtoehtoinen lähestymistapa on käyttää kd-puuta [37][38], jonka vahvuutena on puun tasapainoisuus. Näytönohjainten muistien kasvaessa myös ei-hierarkiset tekniikat [34][35] alkavat olla mahdollisia isojenkin pistepilvien visualisoinnissa.

Lähempään tarkasteluun valittiin sisäkkäispistepuut, sillä niiden suorituskyky on todettavissa valmiista toteutuksesta[8] ja ne vaikuttavat vastaavan luvussa 3.6 esitettyihin vaatimuksiin. Sisäkkäispistepuut mahdollistavat eri tarkkuustasojen visualisoinnin käytettä-

[8] www.potree.org/

vissä olevasta piirtoajasta ja laitteistosta riippuen. Sisäkkäispistepuut mahdollistavat myös pistepilven asteittaisen tarkentamisen monen ruudunpäivityksen ajan, mikä sopii laitossuunnitteluohjelmiston tarpeisiin. Usean laserkeilauksen sovittaminen samaan sisäkkäispistepuuhun mahdollistaa pisteiden harventamisen ja globaalin enimmäistiheyden.

Sisäkkäispistepuiden rakenne mahdollistaa luvussa 4.1 esitetyn yksinkertaisen kompressiotekniikan, jossa pisteiden absoluuttiset koordinaatit vaihdetaan indekseihin solmuissa sijaitseviin ruudukoihin. Suhteellisten koordinaattien käyttäminen nopeuttaa lataamisen ja tallentamisen lisäksi myös joitakin laitossuunnitteluohjelmistossa tarvittavia operaatioita, kuten pistepilvien transformaatioita. Myös pienentynyt tallennustilan tarve on huomattava etu laitossuunnitteluohjelmistossa.

Luvussa 4.3 esiteltiin yksinkertainen visualisointialgoritmi sisäkkäispistepuille, joka pitää jokaisella ruudunpäivityksellä piirrettävää karkeaa yleiskuvaa pistepilvestä näytönohjaimen muistissa. Tämän jälkeen loput pisteet voidaan piirtää niiden tärkeysjärjestyksessä, eli oktettipuun solmujen ruudulle projisoidun koon mukaan. Luvussa 5 selvisi, että esitetty visualisointialgoritmi on suoraviivaista verrokkia hitaampi, mutta visuaalinen tulos on parempi samalla määrällä piirrettyjä pisteitä.

Joihinkin operaatioihin tutkielmassa esitellyn kaltaiset sisäkkäispistepuut eivät kuitenkaan ole optimaalisia. Kun halutaan tarkastella tietyn alueen sisältämiä pisteitä, täytyy niitä etsiä sisäkkäispistepuusta koko polulta juuresta lehtiin. Tämä hidastaa esimerkiksi yksittäisten pisteiden tai pistejoukkojen valitsemista kursorilla. Käytännössä suorituskyky ei kärsinyt liikaa, joten voidaan todeta sisäkkäispistepuiden soveltuvan hyvin pistepilvien visualisointiin laitossuunnitteluohjelmistoissa.

Viitteet

[1] M. Levoy ja T. Whitted, *The Use of Points as a Display Primitive*, (1985)

[2] CADMATIC Oy, https://www.cadmatic.com/, viitattu 23.11.2019

[3] T. Qu ja W. Sun, *Usage of 3D Point Cloud Data in BIM (Building Information Modelling): Current Applications and Challenges*, Journal of Civil Engineering and Architecture, vol. 9, (2015)

[4] R. Heikkilä *et al.*, *Siltojen 3D-suunnittelu- ja mittausprosessin kehittäminen ja käyttöönottaminen (Älykäs silta)*, (2005)

[5] Römerstadt Carnuntum, https://www.carnuntum.at/en/science-history/carnuntum-in-roman-times, viitattu 15.11.2019

[6] C. Scheiblauer ja M. Pregesbauer, *Consolidated Visualization of Enormous 3D Scan Point Clouds with Scanopy*, Proceedings of the 16th International Conference on Cultural Heritage and New Technologies, pp. 242–247, (2011)

[7] F. Menna *et al.*, *3D Digitization of a Heritage Masterpiece - a Critical Analysis on Quality Assessment*, ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, vol. XLI-B5, p. 675–683, (2016)

[8] J. Hecht, *Lidar for Self-Driving Cars*, Optics and Photonics News, vol. 29, pp. 26–33, (2018)

[9] J. Huhtanen, *Suomen maasto kartoitetaan ja ilmakuvataan pian niin tarkasti, että Puolustusvoimat salaa osan aineistosta*, Helsingin sanomat, 27.10.2019,

[10] A. Saxena ja B. Sahay, *Computer Aided Engineering Design*, Anamaya Publishers, (2005)

[11] Leica Geosystems AG, Leica RTC360, 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_slider3.ashx, viitattu 5.1.2020

[12] Leica Geosystems AG, Leica RTC360, https://leica-geosystems.com/-/media/files/leicageosystems/products/datasheets/leica-rtc360-ds.ashx, viitattu 6.1.2020,

[13] J. Fabritius, *Terrestrial three-dimensional laser scanning in Aveva PDMS*, Opinnäytetyö, Tampereen ammattikorkeakoulu, (2009)

[14] H. Houshiar, J. Elseberg, D. Borrmann ja A. Nüchter, *A study of projections for key point based registration of panoramic terrestrial 3D laser scan*, Geo-spatial Information Science, vol. 18, pp. 11–31, (2015)

[15] 3DTK, The 3D Toolkit, http://slam6d.sourceforge.net/, viitattu 14.12.2019

[16] P. J. Besl ja N. D. McKay, *A method for registration of 3-D shapes*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 14, pp. 239–256, (1992)

[17] C. Harris ja M. Stephens, *A combined corner and edge detector*, In Proceedings of Fourth Alvey Vision Conference, pp. 147–151, (1988)

[18] E. Rosten ja T. Drummond, *Fusing points and lines for high performance tracking*, Tenth IEEE International Conference on Computer Vision, vol. 2, pp. 1508–1515, (2005)

[19] D. G. Lowe, *Object recognition from local scale-invariant features*, Proceedings of the Seventh IEEE International Conference on Computer Vision, vol. 2, p. 1150, (1999)

[20] M. Weinmann, *Reconstruction and Analysis of 3D Scenes: From Irregularly Distributed 3D Points to Object Classes*, Springer Publishing Company, (2016)

[21] J. Giesen ja F. Cazals, *Delaunay Triangulation Based Surface Reconstruction: Ideas and Algorithms*, (2006)

[22] M. Berger *et al.*, *A Survey of Surface Reconstruction from Point Clouds*, Computer Graphics Forum, vol. 36, (2016)

[23] R. Schnabel, R. Wahl ja R. Klein, *Efficient RANSAC for point-cloud shape detection*, Computer Graphics Forum, vol. 26, pp. 214–226, (2007)

[24] C. M. Huang ja Y.-H. Tseng, *Plane fitting methods of LiDAR point cloud*, 29th Asian Conference on Remote Sensing, (2008)

[25] M. Piipponen, *3D-suunnittelun hyödyntäminen tehdassuunnittelussa*, Opinnäytetyö, Satakunnan ammattikorkeakoulu, (2012)

[26] *AVEVA Laser Modeller, https://www.digitalengineering247.com/pics/pdfs/AVEVA_L51_LaserModeller.PDF, viitattu 12.10.2019*,

[27] M. Ahmed, C. T. Haas ja R. Haas, *Autonomous Modeling of Pipes within Point Clouds*, Proceedings of the 30th ISARC, pp. 1093–1100, (2013)

[28] Y.-J. Liu *et al.*, *Cylinder Detection in Large-Scale Point Cloud of Pipeline Plant*, IEEE transactions on visualization and computer graphics, vol. 19, pp. 1700–1707, (2013)

[29] R. Qiu, Q.-Y. Zhou ja U. Neumann, *Pipe-Run Extraction and Reconstruction from Point Clouds*, Computer Vision – ECCV, pp. 17–30, (2014)

[30] C. Scheiblauer, *Interactions with Gigantic Point Clouds*, Väitöskirja, Institute of Computer Graphics and Algorithms, Vienna University of Technology, (2014)

[31] S. Rusinkiewicz ja M. Levoy, *QSplat: A Multiresolution Point Rendering System for Large Meshes*, Proceedings of SIGGRAPH, (2000)

[32] C. Dachsbacher, C. Vogelgsang ja M. Stamminger, *Sequential point trees*, ACM Transactions on Graphics, vol. 22, (2003)

[33] M. Schütz, *Potree: Rendering Large Point Clouds in Web Browsers*, pro gradu -tutkielma, Institute of Computer Graphics and Algorithms, Vienna University of Technology, (2016)

[34] M. Schütz, K. Krösl ja M. Wimmer, *Real-Time Continuous Level of Detail Rendering of Point Clouds*, 2019 IEEE Conference on Virtual Reality and 3D User Interfaces, pp. 103–110, (2019)

[35] M. Schütz, G. Mandlburger, J. Otepka ja M. Wimmer, *Progressive Real-Time Rendering of One Billion Points Without Hierarchical Acceleration Structures*, (2019)

[36] M. Wimmer ja C. Scheiblauer, *Instant Points: Fast Rendering of Unprocessed Point Clouds*, Proceedings Symposium on Point-Based Graphics, (2006)

[37] R. Richter, S. Discher ja J. Döllner, *Out-of-Core Visualization of Classified 3D Point Clouds*, (2014)

[38] J. Futterlieb, C. Teutsch ja D. Berndt, *Smooth visualization of large point clouds*, IADIS International Journal on Computer Science and Information Systems, vol. 11, pp. 146–158, (2016)

[39] J. Yang, R. Li, Y. Xiao ja Z.-G. Cao, *3D reconstruction from non-uniform point clouds via local hierarchical clustering*, (2017)

[40] J. Davidsson, *Selvitystyö:Massiivisten pistepilviaineistojen hallintajakelun näkökulmasta*, 3point Oy, Lapinlahdenkatu 16 00180 Helsinki, (2019)

[41] M. Wand *et al.*, *Interactive Editing of Large Point Clouds*, Symposium on Point-Based Graphics, pp. 37–45, (2008)

[42] Nvidia, https://www.nvidia.com/en-gb/design-visualization/quadro/rtx-8000/, viitattu 29.11.2019

[43] A. S. Tanenbaum ja H. Bos, *Modern Operating Systems*, neljäs painos, Prentice Hall PressUSA, (2014)

[44] OpenGl, https://www.khronos.org/opengl/wiki/Buffer_Object_Streaming, viitattu 12.12.2019

[45] OpenGl, https://www.khronos.org/opengl/wiki/Sync_Object, viitattu 12.12.2019

[46] T. Akenine-Moller, T. Moller ja E. Haines, *Real-Time Rendering*, toinen painos, A. K. Peters Ltd., (2002)

[47] M. Yllikäinen, *Kolmiulotteisen visualisoinnin erityispiirteet laitossuunnitteluohjelmistoissa*, pro gradu -tutkielma, Informaatioteknologian laitos, Turun yliopisto, (2013)