

# Pistepilvien visualisointi laitossuunnitteluohjelmistossa

Pro Gradu  
Turun yliopisto  
Tulevaisuuden teknologioiden laitos  
Tietojenkäsittelytiede  
2020  
Timo Heinonen  
Tarkastajat:  
P.P.  
H.H.

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

## TURUN YLIOPISTO

Tulevaisuden teknologioiden laitos

Timo Heinonen Pistepilven visualisointi laitossuunnitteluoohjelmistossa

Pro Gradu, 55 s., 3 liites.

Tietojenkäsittelytiede

24. tammikuuta 2020

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

---

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

Pistepilven 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 perehdyytään pistepilven visualisoinnissa käytettyihin hierarkisiin tietorakenteisiin ja todetaan niin kutsuttujen sisäkkäispistepuiden soveltuvan laitossuunnitteluoohjelmiston pistepilvvisualisoijassa käytettäväksi. Lisäksi esitellään sisäkkäispisteille yksinkertainen kompressiotekniikka ja tärkeimpää pisteitä priorisoiva visualisointialgoritmi.

Lopuksi mitataan kompression ja visualisointialgoritmin vaikutusta suorituskykyyn. Pisteiden kompressio pienentää tiedostokokoja ja lyhtentää tietorakenteen rakennusaikaa, mutta kompression purkaminen vaatii laskenta-aikaa visualisointivaiheessa. Esiteltyn visualisointialgoritmi piirtää pisteitä ruudulle hitaanmin kuin suoraviivainen toteutus, mutta katseliaa lähinnä olevat pisteet tulevat piirretynä tarkemmalla resoluutiolla.

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

# Sisältö

<b>1 Johdanto</b>	<b>1</b>
<b>2 Pistepilvet</b>	<b>3</b>
2.1 Pistepilvien hyödyntäminen tietokoneavusteisessa suunnittelussa . . . . .	6
2.2 Pistepilven käsittely . . . . .	7
2.3 Pistepilven visualisoinnin haasteet . . . . .	13
<b>3 Pistepilvien visualisointiin käytetyt tietorakenteet</b>	<b>15</b>
3.1 QSplat . . . . .	15
3.2 Peräkkäispistepuut . . . . .	18
3.3 Sisäkkäispistepuut . . . . .	19
3.4 kd-puut . . . . .	23
3.5 Ei-hierarkiset tekniikat . . . . .	24
3.6 Laitossuunnitteluoohjelmistoon soveltuva tietorakenne . . . . .	25
<b>4 Pistepilven visualisointi sisäkkäispistepuilla</b>	<b>29</b>
4.1 Pistedatan esitysmuoto . . . . .	29
4.2 Tietorakenteen rakentaminen . . . . .	33
4.3 Visualisointi . . . . .	37
4.4 Pisteiden valitseminen . . . . .	41
<b>5 Tietorakenteen arviointi</b>	<b>43</b>
<b>6 Jatkotutkimusaiheita</b>	<b>50</b>
<b>7 Yhteenvetö</b>	<b>52</b>
<b>Viitteet</b>	<b>52</b>

# 1 Johdanto

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

Pistepilviä voidaan tuottaa myös synteettisesti mistä tahansa 3d-mallista. Yksinkertaisimillaan kolmioverkosta voidaan unohtaa kärkipisteiden liitääntäjeto, 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 kärkipisteiden sijaan olisi tehokkaampaa säilyttää muistissa vain yhtä pistettä. Grafiikkakirjastot ja näytönohjaimet on kuitenkin vielä optimoitu kolmioiden käsittelyyn.

Pistepilvillä kuvataan hyvin erikokoisia kohteita, kuten yksittäisiä esineitä tai kokoisia valtioita 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 esikäsittellä ennen kuin niitä voi käyttää suunnittelu- tai katseluohjelmistoissa. Näitä esikäsittelyvaiheita ovat muun muassa rekisteröinti, normaalivektorien etsiminen ja häiriönpoisto. Joitaakin 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 ongeliin 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 laitossunnitteluoohjelmiston tarpeisiin.

Laitossunnitteluoohjelmistoissa käytetään pistepilviä yleensä muutostöiden yhteydessä, kun laitoksesta halutaan saada ajan tasalla oleva 3d-malli. Nämä pistepilvet vaihtelevat kooltaan yhdestä huoneesta kokonaisiin tehdasalueisiin. Laitossunnitteluoohjelmistossa käytettävän pistepilvivisualisoijan on kyettävä pistepilven koosta riippumatta visualisoimaan reaalialkaisella 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.

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

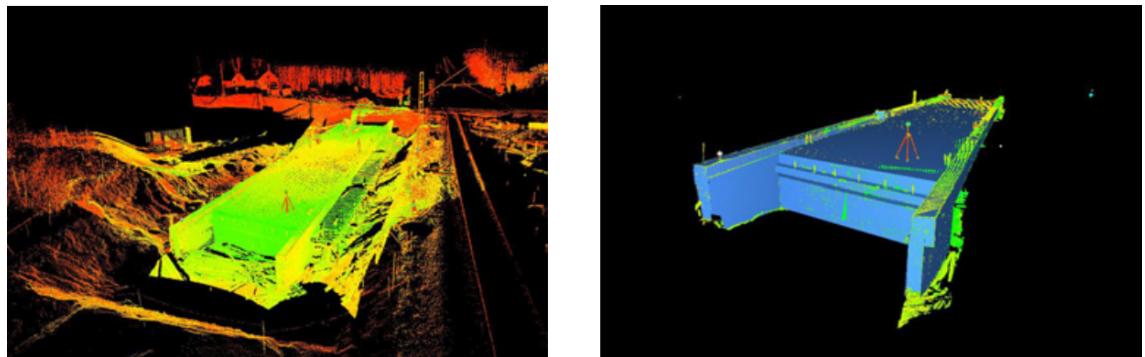
## 2 Pistepilvet

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

Eräs mielenkiintoinen sovelluskohde laserkeilaukselle on siltojen rakentaminen. Silanrakennuksessa haasteita tuottavat tien ja maaston geometrian yhteensovittamisen lisäksi projektin pitkä kesto. Silta on rakennettava osissa ja lopputuloksen on oltava suora, minkä takia siltatyömaalla suoritetaan usein tarkistusmittauksia. Älykäs silta -projektissä tutkittiin tapoja hyödyntää tietotekniikkaa sillanrakentamisessa ja -korjaamisessa, ja havaittiin laserkeilauksen olevan hyvä tapa suorittaa tarkkuusmittauksia. Laserkeilauksella mitattiin pistepilviä sillan kannesta ja rakenteista, jonka jälkeen niistä muodostettiin pintoja 3d-suunnitteluojelmaan. 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 13000 katsojaa. Vuonna 2007 Ala-Itävallan osavaltion hallitus aloitti arkeologiset kaivaukset amfiteatterin alueella, joiden yhteydessä alueesta muodostettiin kattava pistepilvi noin kahdellasadalla laserkeilauksella. Ruudunkaappaus kysesestä pistepilvestä on esitetty kuvassa 2. [5]

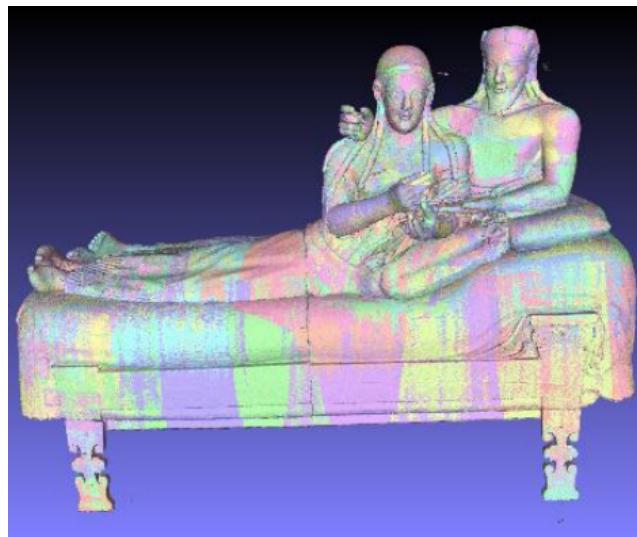
Pistepilviä käytetään historiallisen kulttuuriperinnön säilyttämiseen myös pienemmäs-



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



Kuva 2: Carnuntumin kaupungin amfiteatteri taltioituna laserkeilaiksella. [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 avio-paria rentoutumassa tuonpuoleisessa. Satoihin palasiin hajonnut sarkofagi restauroitiin vuonna 1893 ja se digitoitiin käyttämällä laserkeilausta ja fotogrammetriaa vuonna 2013 taidenäytelyä varten. Sarkofagista keilattu pistepilvi on esitetty kuvassa 3. [7]

Eräs vaativa pistepilvien sovelluskohde on itseohjautuvat kulkuneuvot. Voidakseen navigoida liikenteessä itseohjautuva auto tarvitsee kameroiden ja ultraäänisensoreiden lisäksi katollen laserkeilaimen, jolla voidaan tarkkailla auton etäisyyttä muihin tienkäytäjiin ja esteisiin. Itseohjautuvat autot ovat merkittävä tutkimuskohde myös pistepilven käsitelyn kannalta. Auton katolle asennettavan laserkeilaimen tulisi olla tarkka, nopea ja edullinen, ja sen tuottamaa pistedataa täytyy voida käsitellä reaalialjassa. [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 koteen laajuus tekee pilvistä valtavia. Pistepilviä on käytetty lähinnä metsävarojen kartoittamiseen, mutta Maanmittauslaitos aikoo ryhtyä keräämään pistedataa tarkemmillä laserkeilamilla myös rakennuksista. [9]

Laserkeilauksen kohteena ei ole aina maasto tai eloton rakennelma, vaan myös ihmisiä voidaan tehdä pistepilviä. Esimerkiksi moottoripyöräkypärien suunnittelija voi käyttää hyväkseen ihmisen päätä kuvaavia pistepilviä selvittääkseen sopivan muodon kypärän sisukselle. Kypärän ulkokuori voidaan sen jälkeen mallintaa suunnitteluoohjelmistolla niin, että se täytyy tarvittavat turvallisuusvaatimukset. Pistepilvillä on paikkansa myös lääketieteessä. Proteesin valmistaja voi esimerkiksi käyttää potilaan terveestä jalasta mitattua pistepilveä suunnitellessaan proteesia, jotta se muistuttaisi mahdollisimman paljon amputoitua jalkaa. Myös kirurgit voivat käyttää leikkauksen suunnittelussa hyväkseen elimistä laserkeilauksella muodostettuja 3d-malleja. [10]

## **2.1 Pistepilvien hyödyntäminen tietokoneavusteisessa suunnittelussa**

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

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

Laitossuunnittelussa yleisempi ongelma on 3d-mallin vanhentuminen tai sen puuttuminen kokonaan. Vaikka laitosta alunperin suunniteltaessa siitä olisi tehty 3d-malli, laitteistojen sommittelua saatetaan muuttaa ilman, että samoja muutoksia tehdään 3d-malliin. Kun 3d-mallia halutaan taas hyödyntää, voi olla kustannustehokkaampaa luoda laitoksesta laserkeilaimella pistepilvi kuin mallintaa tehdyt muutokset suunnitteluoohjelmalla. [11]

Jos laitokseen halutaan vaikkapa uusi vesiputki, voidaan sen mahtuminen varmistaa reitittämällä putki suunnitteluoohjelmistolla ja tarkastamalla, osuko se pistepilveen. Jos taas vanhasta laitoksesta halutaan luoda ajantasalla oleva, solidigeometriaa sisältävä 3d-malli, voi suunnittelija mallintaa laitosta pistepilvien avulla. Pistepilven päälle on helppo sijoittaa suunnitteluoohjelman putkistoja ja laitteita oikeille paikoilleen. Markkinoilla on myös ohjelmistoja, joiden luvataan tuottavan pistepilvestä automaattisesti älykäs 3d-malli komponenttitietoineen ilman aikaavievää päälemallinnusta [12].

Kuten sillanrakennuksessa, myös laitoksia rakentaessa on joskus syytä suorittaa tarkustusmittauksia ja verrata niitä alkuperäiseen 3d-malliin. Pistepilvet ovat tähänkin tarkoitukseen sopivia. Nykyaiosten laserkeilainten tuottamat pistepilvet ovat niin tarkkoja, että niistä voi havaita esimerkiksi putkien roikkumisen ja lämpölaajenemisen [11].

Tietokoneavusteisen suunnittelun ohjelmisto käsittelee pistepilviä eri tavoin riippuen suunnittelutyön luonteesta. Kappalemallinnuksessa visualisoidaan usein yksittäisiä osia, joita kuvaavat pistepilvet ovat hyvin tiiviitää ja usein kaikki pistet ovat kerralla näkyvisiä. Laitossuunnittelussa pistepilvet ovat taas hyvin laajoja ja voivat kuvata esimerkiksi kokonaista tehdasalueita. Näissä sovelluksissa on tärkeää pistepilven nopea rajaaminen niin, että vain näkymän sisältävät pistet visualisoidaan. Usein on tarpeen myös käyttää eri tarkkuuksia pistepilven eri osille; kaukana katsojasta sijaitsevista pistestä piirretään vain osa [13].

## 2.2 Pistepilvien käsiteily

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



Kuva 4: Kulkuaikatekniikkaan perustuva Leica RTC360 -laserkeilain. [14]

keilauksia.

Nykyaikaisella laserkeilaimella saadaan muodostettua hyvin tarkka ja tiheä pistepilvi nopeasti. Esimerkiksi kuvassa 4 näkyvä Leica Geosystemsin RTC360 -keilaimen luotaan mittavaan jopa kaksi miljoonaa pistettä sekunnissa ja kiertävän täyden ympyrän alle kahdessa minuutissa. Keilaimen lisäksi laitteessa on kamera, jolla saadaan määritettyä pisteille oikeat värit. [15]

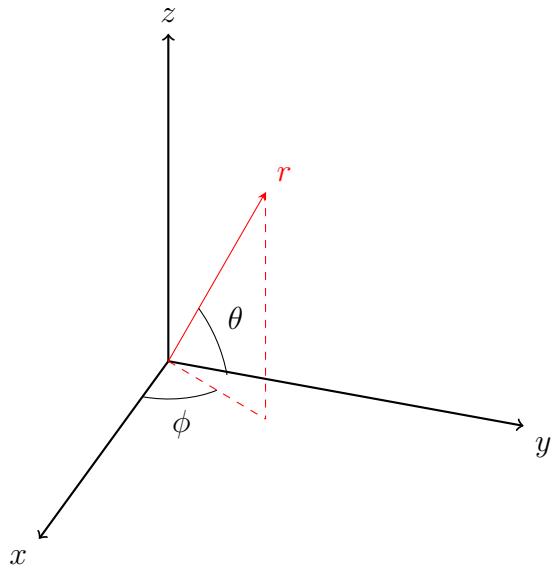
Laserkeilaimien toimintaperiaatteissa on eroja. Kaksi yleisintä toimintaperiaatteita ovat kulkuaikatekniikka (engl. *Time-Of-Flight*) ja vaihesiirtotekniikka (engl. *phase shift*). Kulkuaikatekniikkassa pisteen etäisyys keilaimesta selviää ajasta, joka kuluu laserpurskeen lähetettämisenstä sen heijastuksen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

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

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

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

missä  $n$  on havainnon täysien aaltojen määrä,  $\lambda$  on signaalin aallonpituuus ja  $\Phi$  on lähtevän



Kuva 5: Pallokoordinaatisto. Pisteen sijainti avaruudessa ilmaistaan korotuskulmalla  $\theta$ , atsimuuttikulmalla  $\phi$  ja säteellä  $r$ .

ja palaavan signaalin vaihe-ero. [16]

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,  $\theta$  on korotuskulma ja  $\phi$  atsimuuttikulma. Pallokoordinaatista on havainnollistettu kuvassa 5.

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

Yksinkertaisin tapa visualisoida pistepilvi on muodostaa siitä kaksiulotteinen kuva. Pilvestä voidaan luoda panoramakuva projisoimalla laserkeilaimen ympäröimät pisteet



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

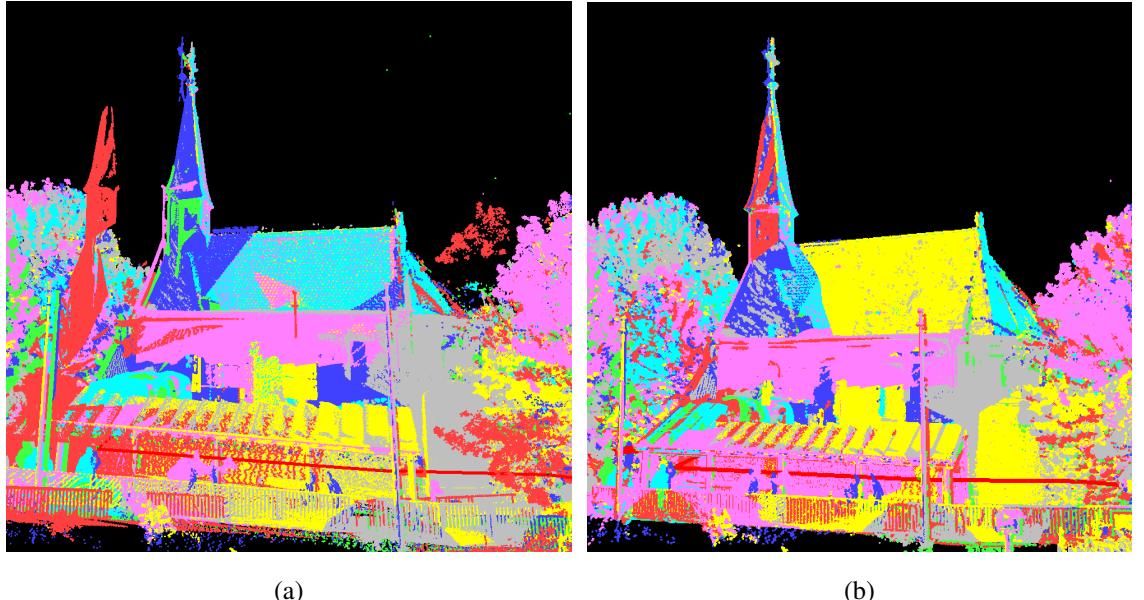
kaksiulotteiselle kuvatasolle.<sup>1</sup> Kuvakoosta riippuen voidaan pistepilvestä karsia huomattava määrä pisteitä, joiden koko olisi liian pieni kuvatasolle projisoituna. Panoramakuvat toimivat siis myös pistedatan pakkausalgoritmina. Panoramakuvan pikseleille voidaan määrittää värit joko sen perusteella, kuinka paljon valoa heijastuu takaisin niitä vastaavista pilven pisteistä, kuinka kaukana pisteet ovat keilaimesta, tai kuinka korkealla pisteet ovat maanpinnasta. Panoramakuvaan on helppo soveltaa erilaisia kuvankäsittelyalgoritmeja, kuten piirteentunnistusta (engl. *feature detection*). Kuva 6 on esimerkki panoramavasta, 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 merkkikuviota, 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. Esimerkik-

---

<sup>1</sup>Projektiot vaikuttavat suuresti panoramakuvan laatuun. Hyvän yleiskatsauksen erilaisiin projektioihin antaa [17]. Kuva 6 on käytetty tasavälistä lieriöprojektiota (engl. *equiangular projection*).



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

si 3DTK-kirjastolla käyttäjä voi valita kahdesta pilvestä samaa aluetta kuvaavia pisteitä, joiden avulla pilvien yhteensovitus onnistuu automaattisesti [18].

Pistepilviä voidaan rekisteroidä myös ilman merkkikuvioita tai käyttäjän apua. Iteratiivinen lähimmän pisteiden algoritmi (engl. *iterative closest point, ICP*) sovitaa pistepilven toiseen etsimällä rotaation ja translaation, jolla pilvien välinen virhe saadaan minimoitua. Virheen laskemista varten täytyy määritellä pilvistä toisiaan vastaavat pisteet. Yksinkertaisimillaan pistettä vastaavaksi pisteeksi valitaan toisesta pilvestä se, jonka euklidinen etäisyys edellä mainittuun on pienin. Iteratiivisen algoritmin kunkin transformaation hyvyyss lasketaan kaikkien vastaavien pisteiden välisten etäisyyksien muodostamasta virheestä. Algoritmin iteroiminen voidaan lopettaa, kun jokin ennaltamääritetty raja virheelle on alitettu. Kuvassa 7 on ICP-algoritmilla sovitettu yhteen kuusi pistepilveä. [19]

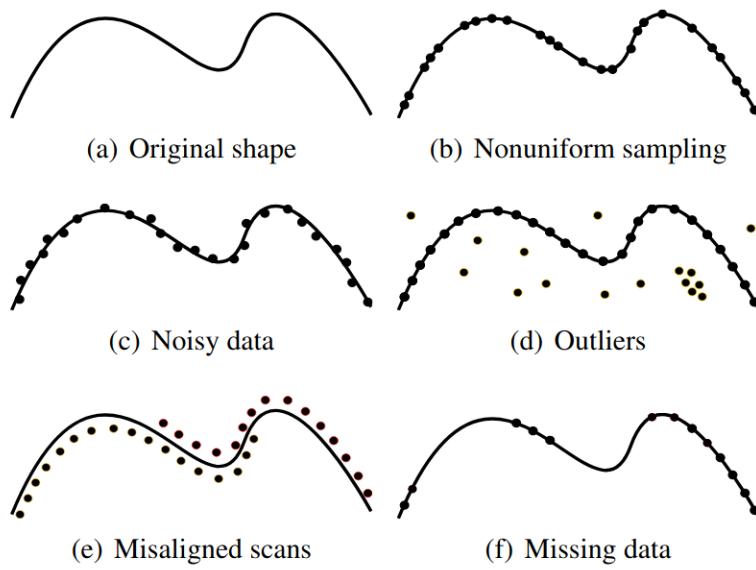
ICP-algoritmi tarvitsee kuitenkin käyttäjältä hyvän alkuarvauksen, jotta virhe supensi ja pilvien sovittaminen onnistuisi. Se on myös erittäin raskas suorittaa isoille pistepilville. Yleensä ICP-algoritmia käytetäänkin karkeamman rekisteröintialgoritmin suo-

rittamisen jälkeiseen hienosäätmiseen. Karkeampi pilvien yhteensovittaminen tehdään yleensä etsimällä kahdesta pistepilvestä muodostetuista panoramakuuvista toisiaan vastaavia avainpisteitä (engl. *keypoints*). Näiden avainpisteiden sijainneista saadaan harva joukko pisteitä, joiden yhteensovittaminen on paljon helpompaa kuin kokonaisten pistepilven. Suosittuja tekniikoita avainpisteiden löytämiseen ovat esimerkiksi Harrisin nurkat [20] sekä FAST ja SIFT -piirteet [21][22]. [23]

Joissakin sovelluksissa halutaan luoda pistedataasta kohteen pintoja kuvaava polygoniverkko, jotta visualisointi olisi nopeampaa nykyaisilla grafiikkakirjastoilla ja visuaalinen lopputulos parempi. Yksinkertainen tekniikka luoda tiivis kolmiointi pistepilvestä on Delaunayn kolmiointi. Delaunayn kolmiointi perustuu Voronoin diagrammiin (engl. *Voronoi diagram*), joka jakaa pisteitä sisältävän tason tai avaruuden konvekseihin Voronoin soluihin. Voronoin solu kattaa sen alueen, jossa etäisyys solua vastaavaan pisteeseen on pienempi kuin muihin pisteisiin. Kahden solun välillä on Voronoin jana, josta etäisyys kahteen pisteeseen on sama, ja kolmen janan leikkauspisteessä on Voronoin kärki, josta etäisyys kolmeen pisteeseen on yhtä suuri. Delaunayn kolmiointi on Voronoin diagrammin duaaligraafi. Kolmiointi luodaan Voronoin diagrammista siten, että pisteen välillä on kaari, mikäli niitä vastaavat Voronoin solut jakavat Voronoin janan. [24]

Oikeat, laserkeilaimella taltioidut pistepilvet sisältävät kuitenkin usein erilaisia virheitä, kuten kuvassa 8 esitellyt epätasainen keilaus, keilaimen häiriö, muukalaispisteet (engl. *outlier*), ongelmat rekisteröinnissä ja puuttuva data. Virheiden johdosta Delaunayn kolmiointi ei ole kovin tehokas algoritmi. Useissa pintojen rekonstruointialgoritmeissa tehdään joitakin oletuksia pistepilven ominaisuuksista. Etenkin tietokoneavusteisessa suunnittelussa keilauksen kohteet muodostuvat yksinkertaisista geometrisista primitiiveistä, kuten tasoista ja sylinteriistä. Esimerkiksi RANSAC-algoritmi [26] sovittelee primitiivejä satunnaistetusti pistepilveen ja arvioi niiden sopivuutta. [25]

Joskus on hyödyllistä tietää pistepilven esittämien pintojen normaalivektorit. Tasainen pinta on helppo sijoittaa pistepilven päälle, jos jollakin alueella on tiheästi pisteitä,



Kuva 8: Kaarevasta pinnasta (a) keilatussa pistepilvissä esiintyviä mahdollisia virheitä: (b) epätasainen pistetihesys, (c) häiriötä pisteiden sijainneissa, (d) muukalaispisteitä, (e) epätarkka rekisteröinti ja (f) puuttuvaa dataa. [25]

joiden normaalivektorit osoittavat samaan suuntaan. Laserkeilaimen epätarkkuudesta tai vaikkapa tuulen heiluttamista puiden lehdistä johtuen on pistepilvissä usein muukalaispisteitä. Yksinkertainen tapa havaita ja poistaa muukalaispisteitä on vertailla pisteiden normaalivektoreita niiden naapuruston normaaleihin ja etsiä poikkeavuuksia.

Pisteen  $p$  normaalivektori voidaan selvittää pääkomponenttianalyysillä (engl. *principal component analysis, PCA*). Ensinnäkin 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 kuvailee pisteen  $p$  normaalialtaa. [27]

### 2.3 Pistepilvien visualisoinnin haasteet

Suurin haaste pistepilvien käsittelyssä ja visualisoinnissa on niiden koko. Nykyään käytettävät laserkeilain, kuten aiemmin esitetty Leica Geosystems RTC360, tuottaa satoja miljooonia 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 neli-tavuisella liukuluvulla ja värit RGB-muodossa kolmella tavulla ja lisätään perään vielä yksi täytetavu, voidaan yksi pilven piste esittää 16:lla tavulla. Miljardin pisteen pilvi olisi siis kooltaan 16 gigatavua mahdollisen metatavan 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 muiston algoritmeja (engl. *out-of-core algorithm*), jotka lataavat pisteitä levyltä muistiin isoissa palasissa, mikä on huomattavasti nopeampaa kuin yksittäisten pisteiden lataaminen. [28]

Pistepilvien koon vuoksi ei ole realistista olettaa, että kaikki pisteet voitaisiin visualisoida reaalialjassa.<sup>2</sup> 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 säänöllisiin 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 mahdolistaavat pistepilven asettaisen tarkentamisen karkeasta yleiskuvasta yksityiskohtiin. Usein interaktiivisissa ohjelmoissa korkean ruudunpäivitystaajuuden ylläpitäminen vaatii sitä, että pistepilvi piirretään matalimmalla tarkkuudella esimerkiksi käyttäjän pyörättäessä näkymää. Toisaalta kun kamera pysähtyy paikalleen, voidaan visualointiin käyttää enemmän aikaa ja pistepilveä tarkentaa.

---

<sup>2</sup>Miellyttävän käytökokemuksen takaamiseksi pitäisi ruutu päivittää vähintään kymmenen kertaa sekunnissa.

### 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 kulkissa tarkkuus kasvaa. Hierarkiasta voidaan joko valita tarkkuustaso, joka takaa nopean ruudunpäivityksen tai tarkeata kuvaan inkrementaaliseksi, kunnes visualisointiaika loppuu kesken tai katselupiste siirtyy ja pistepilvi täytyy piirtää uudestaan toisesta kuvakulmasta.

Esitellään seuraavaksi muutama kiinnostava hierarkinen tietorakenne. Aihealueen pioneerityönä pidetään Rusinkiewiczin ja Levoyn vuonna 2000 julkaisemaa Qsplatia [29], joka onkin antanut paljon vaikuttavia uudemmille tietorakenteille. Dachsbacher et al. esittelivät peräkkäispistepuut [30], joiden avulla pistepilviä voidaan piirtää hyvin nopeasti näytönohjaimella. Viime vuosina pistepilvien visualisoinnin tutkimuksen kirkkainta kärkeä on edustanut Wienin teknillisen yliopiston tietokonegrafiikan tutkimusyksikkö. Tämän tutkielman päälähteinä käytetään Claus Scheiblauerin ja Markus Schützin julkaisuja sisäkkäispistepuista [28],[31]. Schütz et al. ovat esitelleet myös joitakin mielenkiintoisia, ei-hierarkisia tekniikoita [32], jotka kuitenkin rajoittuvat näytönohjaimen muistiin mahdutuviin pistepilviin.

Luvun lopussa listataan laitossuunnitteluoohjelmiston asettamia vaatimuksia pistepilvivisualisoijan käyttämälle tietorakenteelle ja todetaan sisäkkäispistepuiden vaikuttavan lupaavilta myös laitossuunnitteluoohjelmistossa käytettäväksi.

#### 3.1 QSplat

Yksi ensimmäisistä pistedatan visualisointiin käytetyistä hierarkisista tietorakenteista on Rusinkiewiczin ja Levoyn esittämä QSplat, joka on kehitetty kolmioverkon visualisointiin pisteiden avulla. Tietorakenne muodostaminen vaatii, että mallin kolmioiden normaa-

livekitorit tunnetaan, joten se ei suoraan sovella raa' an pistepilvidatan käsittelyyn.<sup>3</sup> QSplatissa on käytetty kuitenkin monia kiinnostavia tekniikoita, joita voi hyödyntää pistepilvidatan käsittelyssä. [29]

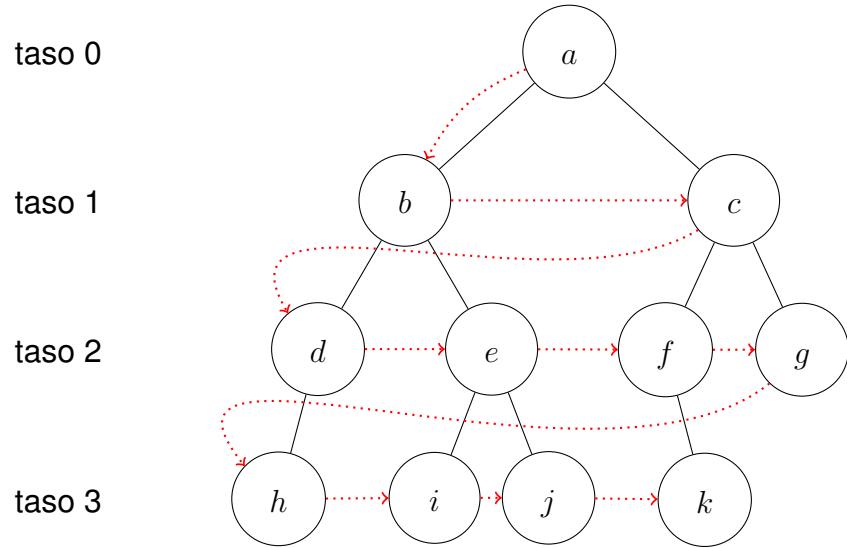
QSplat perustuu puurakenteeseen, jonka solmuissa on rajauspalloja (engl. *bounding sphere*). Pallot jakavat avaruutta rekursiivisesti pienempiin osiin siten, että juuren pallo sisältää kaikki mallin kolmiot ja jokainen sisäsolmu jakaa avaruuden keskimäärin neljään osaan. Puun latva saavutetaan, kun avaruuden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

Puurakenne tallennetaan levylle leveysjärjestysessä (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 teknikoihin, voidaan täplien piirtäminen aloittaa heti, kun tarpeeksi puun solmuja on ladattu levyltä muisiin. Puuta käydään läpi kunnes solmujen rajauspallot ovat niin pieniä, ettei niiden kohdalle enää kannata piirtää täpliä. Puun rakennetta on havainnollistettu kuvassa 9. [29]

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 hakutau-

---

<sup>3</sup>Itse asiassa Rusinkiewicz ja Levoy käyttivät laserkeilatusta pistepilvestä muodostettua kolmioverkkoa, jonka tietorakenne esitti yksinkertaisemmalla pistedatana.



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

lua, voidaan pallon sijainti esittää vain 13:lla bitillä, kun normaali liukulukuesitys vaatii vähintään 16 tavua. [29]

QSplat onnistui piirtämään 1,5-2,5 miljoonaa pistettä sekunnissa, mikä on sen aikaisella laitteistolla erinomainen tulos [29]. Kuten sanottu, se ei sellaisenaan kuitenkaan soveltu laserkeilattujen pistepilvien käsittelyyn. Pistepilvien pisteiden normaaleja ei yleensä tiedetä, joten ne pitäisi esiprosessointivaiheessa selvittää esimerkiksi luvussa 2.2 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 tasolla on keinotekoisia, monia kolmioita kuvaavia palloja. QSplat tarjoaa kuitenkin monia teknikoita, joita pistepilviä käsittelevässä tietorakenteessa voidaan hyödyntää, kuten hierarkinen rakenne ja koordinaattien suhteellinen esitystapa.

### 3.2 Peräkkäispistepuut

Dachsbacher et al. esittelevät niin kutstutun peräkkäispistepuun (engl. *sequential point tree*), joka yrittää hyödyntää QSplatin hierarkian lisäksi näytönohjaimen laskentatehoa. Dachsbacher et al. litistävät hierarkian yksiulotteksi taulukoksi, joka voidaan ladata näytönohjaimen muistiin. visualisointivaiheessa näytönohjaimen riittää vain valita mitkä osat taulukosta pirretään ruudulle. Peräkkäispistepuut siirtävät siis työtä näytönohjaimelle ja jättävät suorittimen vapaaksi muuta laskentaa varten. [30]

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. Visualointiin käytetään rajauspallojen kokoisia täpliä joiden väri määräytyy puun haaran alempien solmujen värien keskiarvolla. [30]

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  $\epsilon$  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. Nämä näytönohjaimen täytyy käydä taulukkoa läpi vain siihen asti, kun katselupisteen etäisyys  $r$  ylittää käsiteltävän pisteen  $r_{max}$ -arvon. [30]

Peräkkäispistepuut voidaan visualisoida nopeasti näytönohjaimen tehokkaan käytön ansiosta, mutta niissä on myös heikkouksia. Tietorakenteen vaatimuksena on, että kaikki data mahtuu näytönohjaimen muistiin. Nämä 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 esittävät parannuksia peräkkäispistepuihin muistioptimoidulla peräkkäispistepuilla (engl. *memory optimized sequential point tree, MOSPT*). Uusien täplien luomisen sijaan puun sisäsolmuissa valitaan lapsisolmuista edustaja, joka parhaiten kuva sisäsolmesta alkavaa haaraa. Haaran solmujen väreistä laksetaan keskiarvo ja edustajasolmuksi valitaan sellainen, jonka väri on lähinnä keskiarvoa. Olemassa olevien solmujen käyttäminen tarkkuustasojen muodostamiseen on tärkeä oivallus, sillä käsitteläässä massiivisia pistepilviä tulisi välttää ylimääräisen datan luomista. [33]

Wimmer ja Scheiblauer 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  $r_{max}$  arvoa, vaan riittää tietää, mistä pistetaulukon indeksistä alkaa mikäkin hierarkian taso. Tämän jälkeen taulukosta piirretään pisteitä kunnes saavutetaan taso, jolla rajauspallot projisoidisivat alle pikselin kokoisiksi kuvaruudulle. [33]

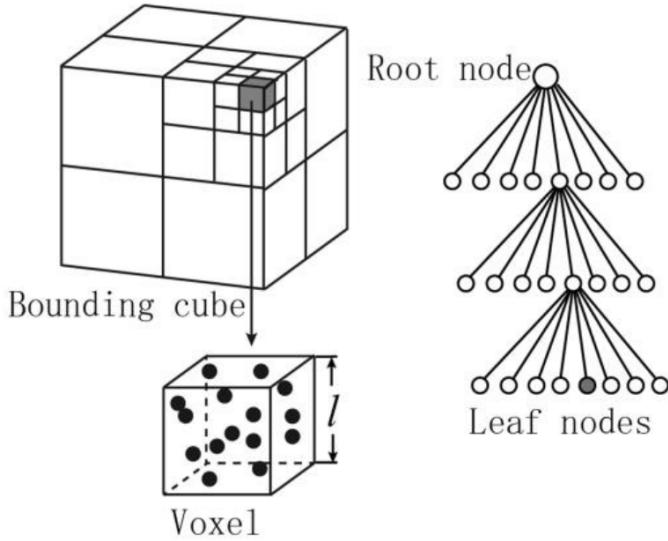
### 3.3 Sisäkkäispistepuut

Wimmer ja Scheiblauer kritisoivat muistioptimoituja peräkkäispistepuita siitä, että ne eivät tue näkökentän ulkopuolisten pisteiden tehokasta karsimista ja siitä, ettei muistioptimointi yksinään riittänyt poistamaan tarvetta ulkoisen muistin algoritmeille. Ratkaisuksi he esittivät sisäkkäisiä oktettipuita (engl. *nested octree*). Oktettipuu<sup>4</sup> 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 tehokkaiseen läpikäymiseen ja näkökentän ulkopuolistenalueiden karsimiseen. Ulomman puun jokainen solmu sisältää yh-

---

<sup>4</sup>Tätä suomennosta käytää esimerkiksi Davidsson [35]. Vaihtoehtoinen suomennos on kahdeksanpuu.



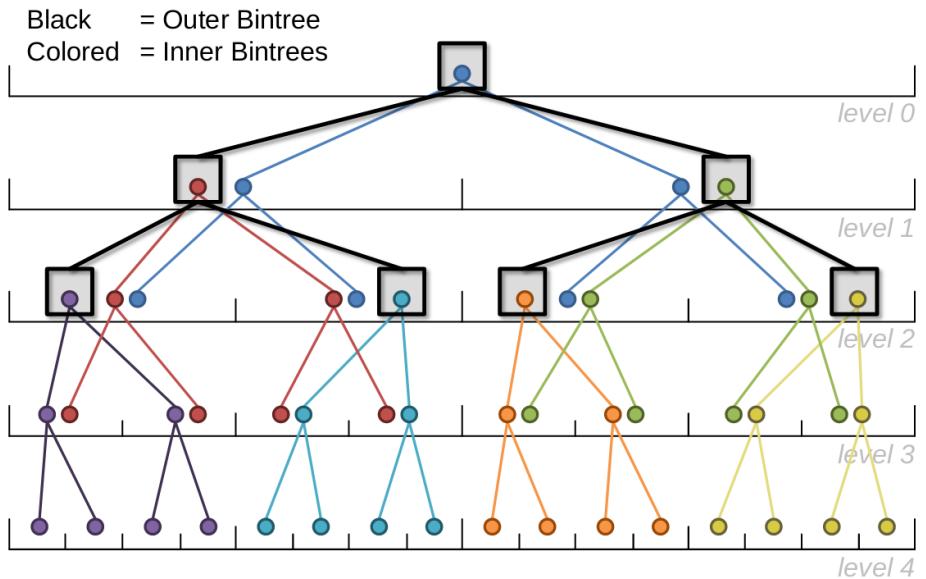
Kuva 10: Oktettipuun juurisolmu sisältää kaikki pisteet sisältävän rajauslaatikon. Jokainen sisäsolmu jakaa rajauslaatikkonsa kahdeksaan osaan. [34]

den sisemmän oktettipuun, joka vastaa samaa avaruuden osaa, kuin ulkoisen puun solmu. Pisteet sijoitetaan sisempiin puihin, yksi jokaiseen solmuun. [33]

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 tasolla. Tarkkuustasojen muodostumista on havainnollistettu kuvassa 11. Puut tallennetaan levylle tarkkuus-taso kerrallaan, mikä mahdollistaa ulkoisen muiston algoritmien käytön. Visualisoitaessa tarvitsee levyltä lukea pisteitä vain haluttuun tarkkuustasoon asti, eikä loppuja pisteitä tarvitse ladata muistiin. [33]

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

MNO:n rakenne eroaa sisäkkäisistä oktettipuista siten, että sisemmät puut korvataan



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

säännöllisillä kolmiulotteisilla ruudukoilla, joihin pistetet tallennetaan. MNO:n rakentaminen alkaa juurisolmesta, joka vastaa kaikki pistetet peittävästä avaruutta. Solmun sisältämä ruudukko jakaa solmua kuvaavan avaruuden osan  $128^3 = 2097152$  soluun. Pistetet lisäään puuhun yksi kerrallaan niin, että jokaiseen ruudukon soluun mahtuu vain yksi piste. Jos solu on varattu, sijoitetaan piste ylimääräiseen taulukkoon odottamaan, että vastaavia pistetet kertyy tarpeeksi, jotta olisi järkevä luoda uusia solmuja puuhun. Kun ennalta-määritty vähimmäismäärä pistetet on kertynyt ylimääräisten pistetiden taulukkoon, luo-daan ruudukon sisältävälle solulle lapsisolmuja ja sijoitetaan ylimääräiset pistetet niihin. Ruudukkoon sijoitettavien pistetiden määrälle on hyvä asettaa myös yläraja. [28]

Jokainen tietorakenteen solmu tallennetaan omaan tiedostoonsa levylle, josta niitä ladataan muistiin visualisointivaiheessa tarvittaessa. visualisointialgoritmien kuuluu käyttää-jän asettama pistebudjetti, joka asettaa ylärajan yhdessä ruudunpäivityksessä piirrettävien pistetiden määrälle.<sup>5</sup> Tätä rajaa säätämällä käyttäjä saa jonkinlaisen kontrollin ruudunpäi-

<sup>5</sup>Scheiblauer testasi pistepilvivisualisoijaansa asettamalla rajan vain sataantuhanteen pisteeseen.

vitystaajuuden suhteen. [28]

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

Scheiblauer oli ottanut MNO:ta kehittääseen vaikutteita Michael Wandin et al. esittämästä oktettipuusta, jonka sisäsolmuihin kuuluu myös ruudukko. Pisteet tallennetaan kuitenkin ruudukon sijaan lehtisolmuihin joihin mahtuu kuhunkin enintään satatuhatta pistettä. Sisäsolmuissa pidetään kopioita yhdestä niiden läpi kulkeneesta pisteestä ja tarkkustasot muodostetaan näistä sisäsolmujen pisteistä. Wandin et al. tietorakenne käyttää MNO:n tapaan ulkoista muistia ja mahdollistaa tehokkaat lisäys- ja poisto-operaatiot. [36]

Markus Schütz jatkoi Wimmerin ja Scheiblauerin työtä esittelemällä opinnäytetyösseen 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 ennaltamääärätyyn rajaan saakka kaikki pisteet, kunnes ne muutetaan sisäsolmuiksi ja liian lähekkäin olevat pisteet jaetaan uusien lapsisolmujen kesken. [31]

Potree käyttää ulkoista muistia tehokkaasti ja pystyy käsittelemään jopa 640 miljardia pistettä sisältäviä pistepilviä.<sup>6</sup> Rakennusvaiheessa oktettipuun solmuja tallennetaan tasai-

---

<sup>6</sup>Kyseinen pistepilvi (Actueel Hoogtebestand Nederland, ANH2, <http://ahn2.pointclouds.nl/>) kuvaaa koko Alankomaiden valtiota ja se vaatii 7,68 teratavua tallennustilaata. Potreen tietorakenteessa

sin väliajoin levylle, jottei muisti täytyisi. 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. [31]

Potreen visualisointialgoritmi priorisoi niitä hierarkian solmuja, jotka ovat lähellä kat selupistettä ja joiden kuvaruudulle projisoitu koko on suurin. Visualisoinnin suorituskykyä voidaan säädellä Scheiblauerin toteutuksen mukaisesti käyttäjän asettamalla piste budgetilla. Schütz on kehittänyt Potreehen myös näytönohjaimella ajettavan algoritmin mukautuvaan pisteiden koon määrittämiseen; pistepilven harvemmissa osissa piirretään pisteet suurempina, jottei reikiä esiintyisi. [31]

### 3.4 kd-puut

Suosittujen oktettipuiden lisäksi on pistepilvien visualisoinnissa käytetty kd-puita (engl. *k-dimensional tree, kd-tree*)<sup>7</sup>. 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 et al. kehittivät massiivisten ulkoilmakeilausten visualisointiin kd-puuhun perustuvan pistepilvisualisoijan, 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 objektiluopistepilvi jakautui 13:lle tasolle ja 38:aan miljoonaan solmuun.

---

<sup>7</sup>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. [32]

kalla oli oma rajansa muistinkäytölle. [37]

Futterlieb et al. 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]

### 3.5 Ei-hierarkiset tekniikat

Pistepilvistä voi muodostaa tarkkuustasoja myös ilman hierarkista tietorakennetta. Markus Schütz et al. [32] esittivät virtuaalitodellisuuslaseille suunnatun pistepilvivisualisointia, 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. [32]

Jatkuvat tarkkuustasot ratkaisevat hierarkisia tietorakenteita käytettäessä usein esiintyvän ongelman diskreettien tarkkuustasojen näkyvistä rajoista. Piirretystä 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. [39] 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 kuva tarkennetaan seuraavilla ruuduilla samalla kun näytönohjaimen muistiin ladataan lisää pisteitä. Puskurien täyttäminen satunnaisessa järjestyksessä saa aikaan kuvan miellyttävän tarkentumisen.

Nämä kaksoi 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 [40]. Laitossuunnitteluoohjelmistossa käytettävien pistepilven kokoluokka on harvoin teratavuja, joten voisi olla kannattavaa hierarkisten tietorakenteiden kehittämisen sijaan keskittyä harventamaan pistepilveä niin, että se mahtuu näytönohjaimen muistiin ja käyttää jotakin tekniikkaa, joka käyttää näytönohjaimen rinnakkaislaskentatehoa mahdollisimman tehokkaasti.

### **3.6 Laitossuunnitteluoohjelmistoon soveltuva tietorakenne**

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

Kun laitoksesta halutaan luoda ajantasalla oleva 3d-malli pistepilven avulla, täytyy se mallintaa suunnitteluoohjelmiston käyttämäksi geometriaksi pistepilveä mukailleen. Lattiat ja seinät on tasoina helppo asettaa paikalleen, kuten myös suunnitteluoohjelmiston komponenttikirjastosta löytyvät laitteet. Suurin työ on yleensä putkistoissa, ilmakanavissa ja kaapeliradoissa. Useat suunnitteluoohjelmistot tarjoavat jonkinasteista automatisointia etenkin putkien reititykseen pistepilven päälle. Ohjelmisto voi automaattisesti tunnistaa pilvestä

sylinterit ja asettaa niiden päälle sopivia putkisto-osia. Vaihtoehtoisesti käyttäjä voi valita pilvestä muutamia pisteitä ja ohjelmisto laskee niiden perusteella putken pituuden ja halkaisijan ja asettaa oikean osan paikalleen. Mallinnustyö ja etenkin automaattiset muodonmuuttusalgoritmit asettavat ohjelmistolle vaatimuksen tarkkuudesta. Laitossuunniteluohjelmistossa käytetään yleensä millimetrejä perusyksikköinä, joten pistepilvessä ei saisi esiintyä senttimetriien virheitä.

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 vaa-tima pieni siivu. Myös pistepilveä voidaan rajata niin, että siitä näkyy vain tarpeellinen osa. Pistepilviä visualisoivan ohjelmiston tulisi siis kyettä rajaamaan pilveä toistuvasti ja nopeasti. Käyttökokemus olisi paras, jos käyttäjä pystyisi hiirellä interaktiivisesti määrit-tämään 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 mil-jonien pisteen joukosta etsitään juuri kurSORin alla oleva piste. Yksittäisten pisteen hakeminen pilvestä täytyy siis olla nopeaa.

Toinen yleinen pistepilvien käyttökohde on 3d-mallin katselu joko laitossuunnitte-luohjelmistossa tai erityisessä mallinkatseluohjelmistossa. Etenkin suunnitteluprojektien esimiehet haluavat usein tarkastella suunnittelijoiden luomaa 3d-mallia helposti ja no-peasti. Luonnollisesti malliin kuuluvat pistepilvet tulevat myös näkyä katselijalle. Tä-

mä saattaa tuottaa haasteita ohjelmiston kannalta, sillä katseluohjelmistojen käyttäjillä on käytettävissä harvoin yhtä järeää laitteistoa, kuin suunnittelijoiden työasemat. Mal-linkatseluohjelmistossa 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.

Tässä tutkielmassa kehitetään laitossuunnitteluoohjelmistolle optimoitu hierarkinen tie-torakenne pistepilven 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 tihästi näytteistettyjä osia.
4. Käyttäjän on voitava määrittää pilvestä alueita, joiden sisältävien tai ulkopuolelle jäävien pisteiden ominaisuuksia, kuten näkyvyyttä tai väriä, voidaan muuttaa.
5. Pilvestä on voitava nopeasti ja tarkasti valita yksittäisiä pisteitä.
6. Pistepilvessä ei saa esiintyä yli millimetrin suuruisia virheitä.

Luvussa 3.3 esiteltyn Potree on osoittautunut massiivisten pistepilven interaktiivisen visualisoinnin olevan mahdollista jopa verkkoselaimessa, jossa etenkin tiedonsiirtonopeus rajoittaa visualisoinnin suorituskykyä. Tarkastellaan siis sisäkkäispistepuiden soveltuuutta laitossuunnitteluoohjelmistoon.

Oktettipuun läpikäyminen taso kerrallaan muodostaa tehokkaasti tarkkuustasot, joten vaatimus 1 on helppo tyydyttää. Pisteiden asettelu sisäkkäisten oktettipuiden solmuihin

mahdollistaa myös vaatimuksen 2 mukaisesti ulkoisen muistin käyttämisen. Scheiblauerin muokattavien sisäkkäisten oktettipuiden jokainen solmu sisältää ruudukon, johon pistetet sijoitetaan. Mitä syvemmällä tasolla solmu on, sitä pienempiä ruudukon solut ovat. Vaatimuksen 3 esittämä pilven harvennus onnistuu asettamalla puulle enimmäissyyvyys ruudukon koon mukaan ja hylkäämällä lehtisolmuissa kaikki pistetet, jotka tulisi lisätynksi jo varattuun soluun.

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

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

## 4 Pistepilven visualisointi sisäkkäispistepuilla

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

### 4.1 Pistedatan esitysmuoto

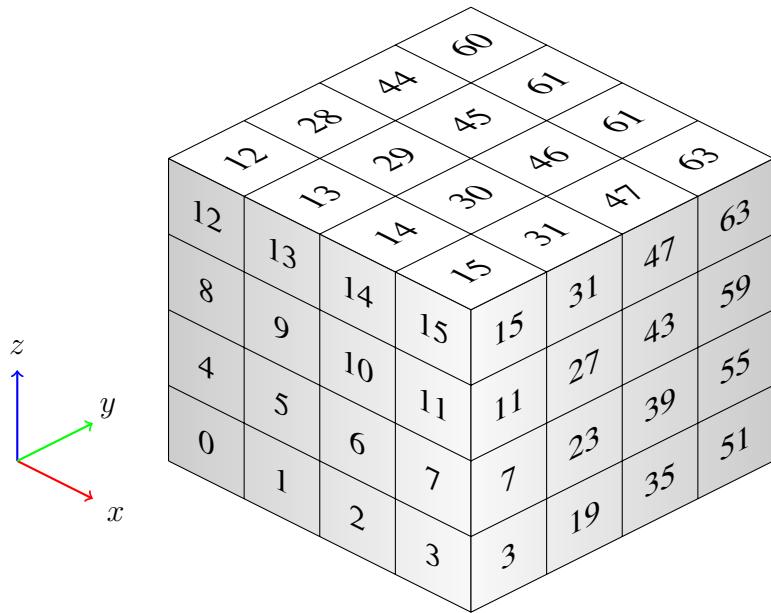
Pistepilvet sisältävät usein satoja miljoonia tai jopa miljardeja pisteitä, mikä johtaa luonnollisesti isoihin tiedostokokoihin. Yleensä hierarkiatiedon osuus tiedoston sisällöstä on varsin pieni, joten tiedostokokoa saadaan pienennettyä parhaiten tiivistämällä itse pisteiden esitysmuotoa. Pisteistä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

```
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.<sup>8</sup> Muuttamalla pisteiden esitysmuotoa voidaan pistepilviä tiivistää ja joitakin ope-

---

<sup>8</sup>Pistepilvisovelluksen käyttäjän rahapussin koosta riippuen tämän kokoinen pilvi mahtuisi vielä keskusmuistiin ja jopa näytönohjaimen muistiin [40].



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

raatioita nopeuttaa.

Puun rakennusvaiheessa pisteet lisätään jokaisessa solmussa olevaan kolmiulotteiseen ruudukkoon, jonka jokaiseen soluun mahtuu vain yksi piste. Kun piste lisätään ruudukon soluun, tallennetaan solun järjestysnumero hajautustauluun, josta voidaan jatkossa nopeasti tarkastaa, onko kyseinen solu varattu. Mahdollinen numerointi ruudukolle on esitetty kuvassa 13. 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 pisteen absoluuttinen sijainti voidaan unohtaa ja käyttää sijainnin tallentamiseen pisteen suhteellista sijaintia ruudukossa. Yksinkertaisimillaan 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 välttisi enää 8 gigatavua muistia.

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

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

```

struct Point {

    unsigned char dx;

    unsigned char dy;

    unsigned char dz;

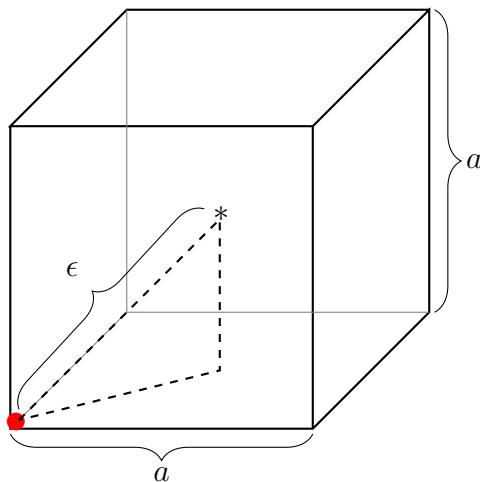
    unsigned char intensity;

}

```

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

Kun pisteen tarkka sijainti unohdetaan ja se ilmaistaan suhteessa ruudukkoon, syntyy pistepilveen virheitä. Suurinta mahdollista virhettä on havainnollistettu kuvassa 14, kun piste visualisoidaan solun keskipisteenä, vaikka se oikeasti sijaitsisi aivan sen nurkassa.



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

Jos oletetaan, että ruudukon solut ovat kuutioita, saadaan enimmäisvirhe laskettua helposti Pythagoraan lauseella muotoon

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

Vaatimus 6 esitti virheen enimmäissuuruudeksi yhtä millimetriä. Oktettipuu 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 lisättyt pisteet täyttävät vaatimuksen 6. Tämä ei tarkoita sitä, että puu voisi sisältää pisteitä vain tasolle 11 ja sitä syvemmälle. Pisteitä voi hyväksyä suuriinkin soluihin, jos ne ovat sattuvat osumaan tarpeeksi lähelle sen keskipistettä. Käytännössä suurissa pistepilvissä on pisteitä yleensä niin tiheästi, että puun ylimillekin tasolle tulee tallennetuksi pisteitä.

Pisteiden sijainnin suhteellinen esitysmuoto nopeuttaa myös pistepilvien käsittelyä.

Laitossuunnittelussa pistepilveä joudutaan usein sovellukseen latauksen jälkeen liikuttamaan ja skaalaamaan, jotta se istuisi hyvin 3d-malliin. Jos pisteisiin olisi tallennettu absoluuttinen sijainti, jouduttaisiin pistepilveä tallennettaessa uudelleen kirjoittamaan kaikki pisteet käyttäjän määrittämällä transformaatiomatriisilla kerrottuna. Edellä kuvatulla esitysmuodolla transformaatio täytyy suorittaa vain oktettiipun solmujen rajauslaatikoilla, joista tiivistetyt pisteiden sijainti johdetaan. Tämä johtaa merkittävään parannukseen käytökokemuksessa, sillä satojen miljoonien pisteiden kirjoittaminen levylle kestää useita minuutteja.

## 4.2 Tietorakenteen rakentaminen

Edellä esitellyn oktettiipun rakentaminen suoritetaan kahdessa vaiheessa. Ensin käydään kaikki syötepisteet läpi ja selvitetään niille rajauslaatikko. Tämän jälkeen luodaan tyhjä juurisolmu, jonka rajauslaatikkona toimii äsknen laskettu, kaikki pisteet sisältävä 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 määälle kannattaa asettaa yläraja, jotta saavutetaisiin 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, ylittiäisikö uusi lapsisolmu puun enimmäissyyvyyden. Kun ennaltamäärätylle pohjatasolle hyväksytään kaikki pisteet, kunhan niitä vastaava ruudukon solu on vapaa, saadaan pistepilvelle tehokas ja globaali enimmäistiheys. Tällä tavalla pilveä saadaan harvnettua tehokkaasti ja vaatimus 3 voidaan tyydyttää.

Tietorakennetta tallennettaessa kirjoitetaan tiedostoon ensin tarvittavat tiedot puun

---

**Algoritmi 1:** RakennaOktettipuu
 

---

**Syöte :** Joukko pistepilviä  $P$

**Tuloste:** Pisteet sisältävän oktettipuun juurisolmu  $s$

```

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

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

17 return  $s$ 

```

---

solmuista ja pisteet vasta niiden jälkeen. Jokainen solmu sisältää tiedon siitä, kuinka monta pistettä siihen kuuluu, sekä ensimmäisen pisteen indeksin pistetaulukossa. Tämä mahdollistaa sen, että pistepilveä käsiteltäessä luetaan tiedostosta ensin vain kevyt hierarkia, jonka jälkeen vain tarvittavat pisteet voidaan lukea muistiin. Tiedoston rakennetta on havainnollistettu kuvassa 15. Pisteiden lukumäärän ja ensimmäisen pisteen indeksin lisäksi jokaisesta solmusta tallennetaan rajauslaatikko ja sijaintikoodi, joka kertoo sen sijainnin puussa. Sijaintikoodin numerot kertovat reitin juurisolmusta kyseiseen solmuun. Esimer-

---

**Algoritmi 2:** LisääPisteSolmuun
 

---

**Syöte :** Puun solmu  $s$ ,

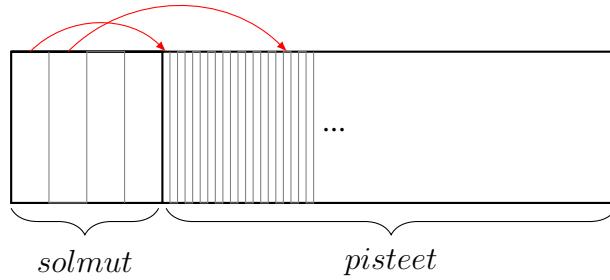
piste  $p$ , jolla on sijainti  $(x, y, z)$  ja väri  $(r, g, b)$

```

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

```

---



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

kaksi 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 tallennustilaan  $33+d$  tavua, missä  $d$  on solmun syvyys puussa.

Edellä kuvattu rakennusalgoritmi pitää koko oktettipiun muistissa ennen kuin pis-

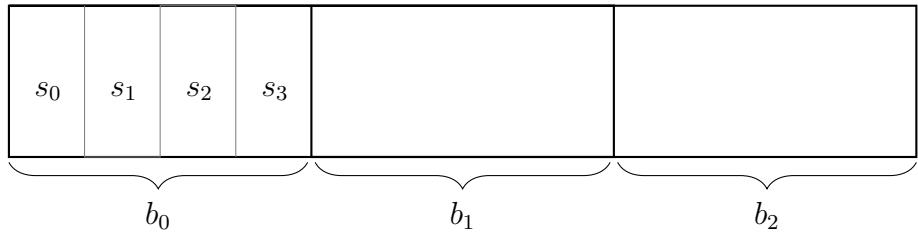
teet kirjoitetaan levylle. Massiivisten pistepilvien tapauksessa keskusmuisti saattaa täytyä vaikka pilveä olisi harvennettu ja pisteitä kompressoitu. Muisin täytyessä käyttöjärjestelmä alkaa pitää osaa prosessin muistiavaruudesta kiintolevyllä sivutuksen (engl. *paging*) avulla [41]. Jatkotutkimuksen aiheeksi jää selvittää, onko nopeampaa antaa käyttöjärjestelmän siirtää sivuja kiintolevylle, vai esimerkiksi tallentaa ensin pisteitä väliaikaiseen tiedostoon niiden lisäysjärjestykseen, josta 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 laitossuunnitteluoohjelmistossa visualisoida muitakin grafiikkaa. Näytönohjaimen muistia voidaan säestää käytämällä niin kutsuttua puskurivirtaa (engl. *buffer streaming*). Yleinen ongelma grafiikan piirtämisessä on se, että näytönohjain prosessoii dataa nopeammin kuin suoritin ehtii sillä 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 puskuria ilman, että sen tarvitsisi odottaa lainkaan hidasta tiedonsiirtoa. [42]

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

Tietorakenteen visualisointia testataessa puskurivirta ei kuitenkaan tuottanut tarpeeksi miellyttävää visuaalista loppululosta. Etenkin pilveä pyörityläessä ja sen läpi liikuttaessa



Kuva 16: 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 tarkoituksesta on vähentää tiedonsiirrosta aiheutuvaa latenssia.

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 pistetä näkyviin solmuihin, mutta näkyvyystarkastelun suorittaminen ja puskarin muokkaaminen jokaisella ruudunpäivityksellä osoittautui liian aikaavieväksi.

Yleiskuvan visualisoinnin jälkeen jäljelle jäävät solmut järjestetään niiden kuvaruudulle projisoidun koon mukaan, koska pistepilvi 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 täytövuorossa olevaan puskurivirran osioon. Puskuriosion täyttyessä vaihdetaan täytettäväksi seuraava osio. Puskurivirtaa käyttäessä täytyy varmistaa, ettei sama osio ole sekä suorittimen kirjoitettavana, että näytönohjaimen luettavana. Tämä tapahtuu lähetämällä näytönohjaimelle jokaisen puskuriosion täytymisen jälkeen synkronointiobjekti (engl. *sync object*). Kun uutta osiota otetaan kirjoitettavaksi, tarkastetaan, että näytönohjain on merkannut kyseisen osion synkronointiobjektiin

---

**Algoritmi 3:** PiirräSisäkkäispistepuu
 

---

**Syöte :** Sisäkkäispistepuu  $P$

```

1  $b_{yleiskuva} \leftarrow$  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  $s \in P$  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   $S \leftarrow$  näkyvissä olevat puun solmut, joita ei lisätty puskuriin  $b_{yleiskuva}$ 
12  Järjestä  $S$  ruudulle projisoidun koon mukaan
13  PiirräPuskurivirta( $S$ )

```

---

käsitlelyksi. [43]

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. Ajatuksesta on visualisoida vain tärkeimmät objektit ja jättää kaukana olevat tai pienet objektit käsittämättä, jotta ruudunpäivitystaajuus pysyy interaktiivisenä. 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.

---

**Algoritmi 4:** PiirräPuskurivirta
 

---

**Syöte :** Sisäkkäispistepuun solmujoukko  $S$

```

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

```

---

[44] esittävät kuvaruudulle projisoidun objektiin koon arviolle kaavaa

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

jossa  $n$  on katselupisteiden etäisyys kuvatasosta,  $r$  objektiin rajauspallon säde ja  $\mathbf{c}$  keskipiste,  $\mathbf{d}$  on normalisoitu katsomissuunta, ja  $\mathbf{v}$  katselupiste. [13]

Mikko Yllikäinen huomautti pro gradu -tutkielmassaan [13], ettei tarkkaa arvioita objektiin koosta kannata selvittää, jos halutaan selville vain suuruusjärjestys. Yllikäinen yksinkertaistaa kaavan muotoon

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

Yllikäinen nimittää täällä kaavalla muodostetu suuruusjärjestyksen käyttämistä yksityiskohtien karsinnassa kontribuutiokarsinnaksi (engl. *contribution culling*). On huomattavaa,

että tällainen karsinta ei ole mahdollista paralleliprojektiomaisemissa, joissa katseluetäisyys ei vaikuta objektien kokoon. [13]

#### 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 piste ammutaan kamerasta säde kursorin projisoidun sijainnin läpi pistepilveen. Nyt pisteitä tarvitsee etsiä vain niistä oktettipuun solmuista, joihin säde osuu. Yksinkertaisimmillaan voidaan valita sädettä lähinnä oleva piste. Tällöin valituksi saattaisi kuitenkin tulla muukalaispiste, joka sattuu olemaan juuri kameran edessä. Yksinkertainen tapa välttää muukalaispisteiden valitsemista olisi hyvä sellaiset puun solmut, joissa säde osuu vain yhteen tai muutamaan pisteeseen. Toinen vaihtoehto olisi kerätä useita kandidaattipisteitä ja hylätä varmuuden vuoksi muutama kursoria lähinnä oleva piste.

Oktettipuun rakenne auttaa myös vaatimuksen 4 tyydyttämisessä. Jos pistepilvestä halutaan piilottaa tai korostaa tiettyä osaa, voi käyttäjä valita maisemasta laatikon ja muokata sen sisältämien pisteiden näkyvyyttä. Oktettipuuta käytettäessä ei jokaisen pisteen sisältymistä valintalaatikoihin tarvitse selvittää, vaan riittää tarkastaa, onko solmun rajauslaatikko valintalaatikon sisällä. Vain siinä tapauksessa, että rajauslaatikko on vain osittain valintalaatikon sisällä, tarvitsee tarkastus tehdä kaikille pisteille. Valintalaatikoita voidaan pitää muistissa visualointivaiheessa 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, kuvaavat valintaoktettipuua valittujen pisteiden asuttamaa avaruuden osaa. Valintaoktettipuuta käy-

tetään esimerkiksi pisteiden piilottamiseen näkymästä. Pisteitä piirrettäessä tarkastetaan, osuuko sen sijainti valintaoktettipuun solmuihin ja päätetään sen perusteella, hylätäänkö piste vai ei. [28]

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

## 5 Tietorakenteen arvointi

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

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

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

Taulukoissa 1, 2 ja 3 on mitattu testipilvien rakentamiseen, tallentamiseen ja pisteiden läpikäyntiin kuluva aika käyttäen kolmea luvussa 4.1 käytettyä pisteiden esitysmuota: maailmakoordinaatit ja väri (16 tavua), ruudukon solun indeksi ja väri (8 tavua), sekä solun suhteelliset koordinaatit ja laserkeilaimeen takaisin heijastuneen valon intensiteetti

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

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

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

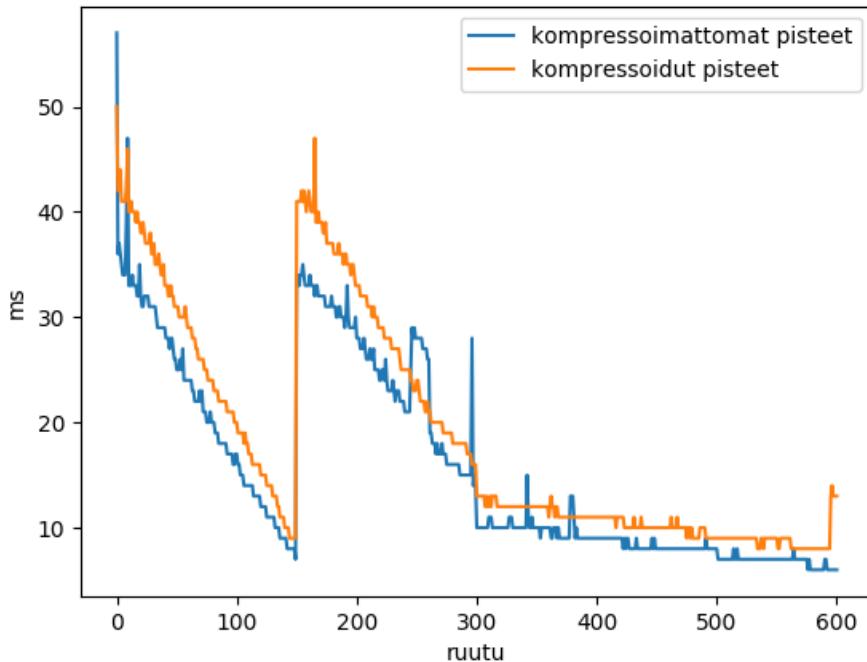
Taulukko 3: Pumput-pilvestä muodostetun oktettipuun rakentaminen ja pisteiden läpikäyminen

(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 tавuja 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 506366789 pistettä 267641 solmussa kahdeksassa tasossa. *Worksite-pilvi* on Leica Geosystems testidataa, joka sisältää 7 keilausta, joiden 53881180 pistettä jakautuu 543105 solmuun yhdeksälle puun tasolle.

Arviodaan ensin luvussa 4.1 esitellyn kompression vaikutusta pistepilven visuali-



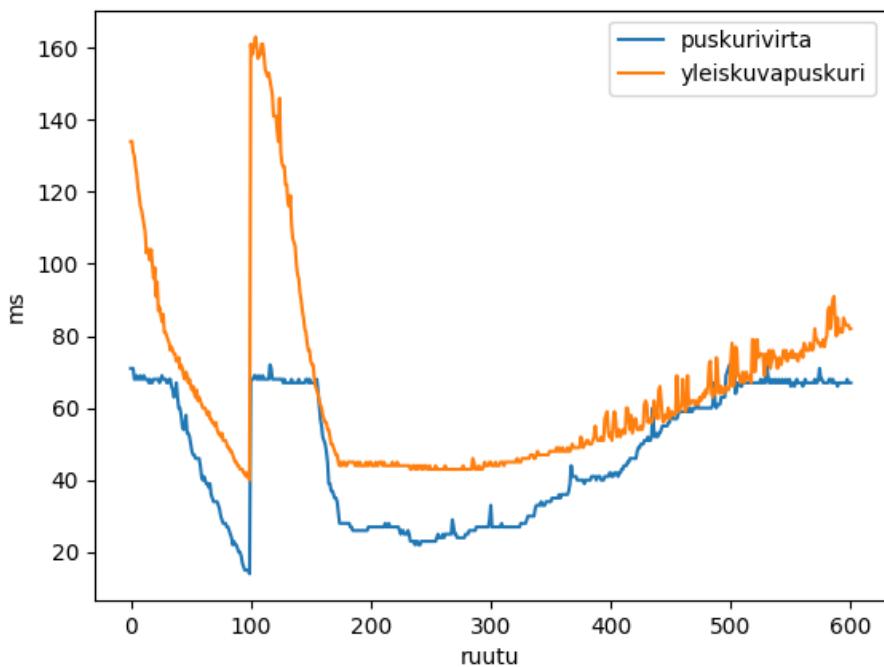
Kuva 17: Worksite-pilven kahden miljoonan pisteen piirtämiseen vaadittu aika milisekunteina käyttäen kompressoimattomia 16:n tavun pisteitä ja kahdeksaan tavaan kompressoituja pisteitä.

sointiaikaan. Kuvassa 17 on esitetty kaavio kahden miljoonan pisteen piirtämisen vaatimasta ajasta kamera-ajon jokaisella ruudunpäivityksellä. Sininen viiva kuvailee piirtoaikaa kompressoimattomilla pisteillä ja oranssi viiva värit säilyttäävällä kompressiolla. Visualisoinnissa on käytetty luvussa 4.3 esitelyä 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 17 ruudun visualisointiajan laskiessa. Näkymäkartton sisällä olevien solmujen määrä kasvaa äkkinäisesti noin 150:nnen ruudun kohdalla, jolloin kamera käännyy 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

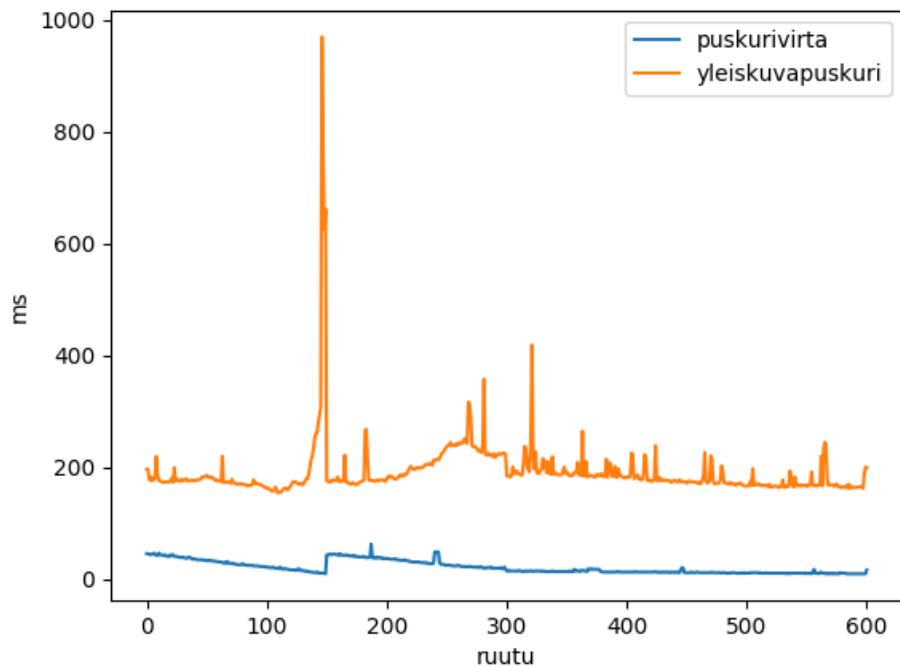
on jonkin verran nopeampaa kuin kompressoitujen kahdeksantavuisten pisteen. Eron selittää kompression avamiseen vaadittu laskenta. Jokaisesta kompressoidusta pistestä etsitään kompressoidun pisteen indeksiä vastaava ruudukon solu ja lasketaan sen keskipiste. Visualisointiajan ero on kuitenkin pieni ja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



Kuva 18: 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 18 on mitattu kahden miljoonan pisteen piirtoaikaa ilmanvaihtohuone-pilvessä. Sininen viiva kuvaaa piirtoaikaa käytettäessä pelkkää puskurivirtaa ja oranassin 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 723834 pistettä.

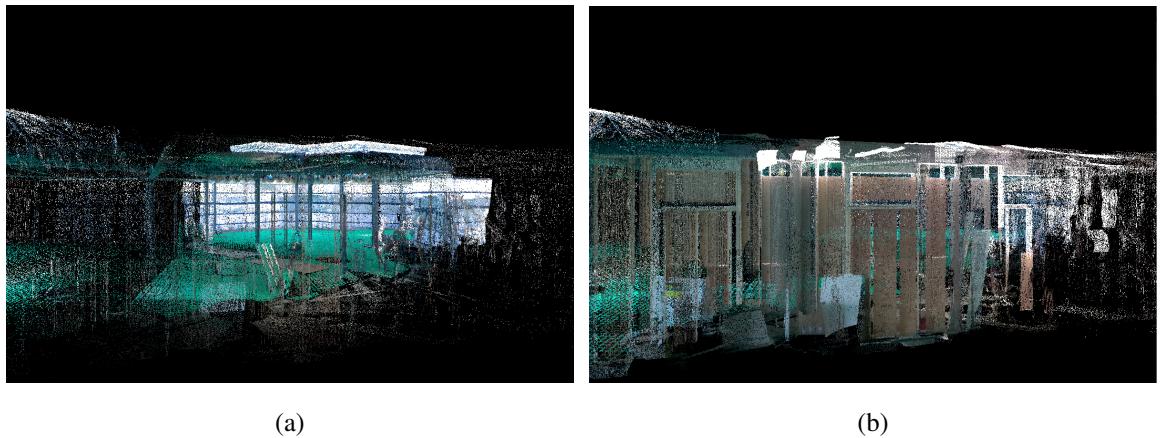
Ilmanvaihtohuoneen kamera-ajo alkaa huoneen reunalta kameran osoittaessa vastakkaiselleseinälle. Kamera liikkuu kohti vastakkaisesta seinää, jolloin visualisoitavien solmujen määrä vähenee. Kameran saavutettua vastakkaisen seinän noin sadan ruudun jälkeen se käännyy ympäri osoittamaan huoneen poikki.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



Kuva 19: Worksite-pilven kahden miljoonan pisteen piirtämiseen vaadittu aika millisekunteina kahdella eri visualisointialgoritmilla

Kuvassa 19 on tehty vastaavat mittaukset worksite-pilvelle. Oranssin viivan kuvaamassa visualisointitavassa on yleiskuvapuskurissa puun viisi ylintä kerrosta, 1862 solmua ja 481663 pistettä.

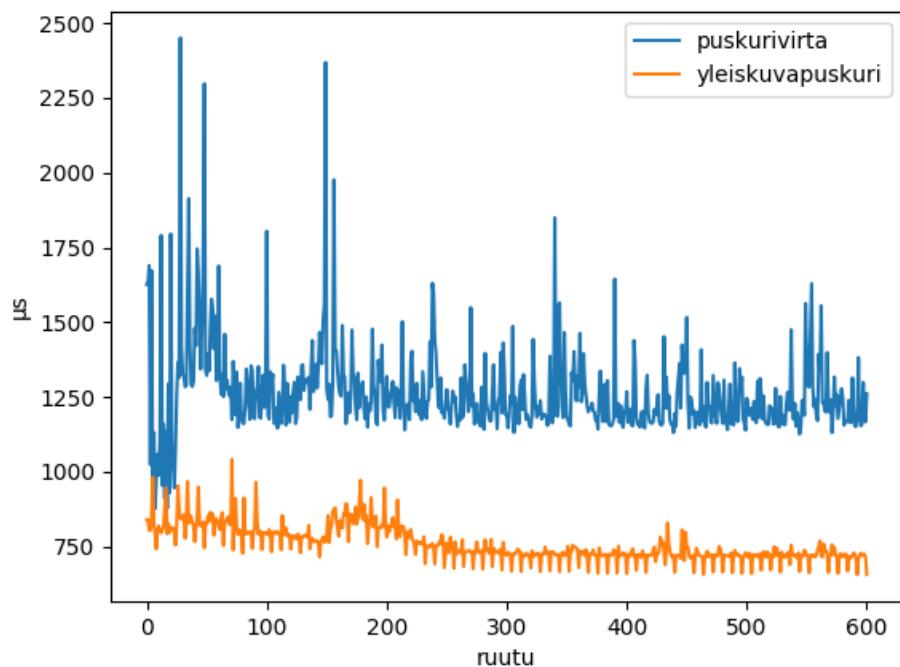
Mittauksista selviää, että pelkän puskurivirran käyttäminen puun visualisoinnissa on selkeästi nopeampaa kuin yleiskuvapuskuri ja puskurivirran yhdistelmällä. Tämä ero se-



Kuva 20: Worksite-pilvi visualisoitu kahdella miljoonalla pisteellä käyttäen (a) pelkkää puskurivirtaa ja (b) yleiskuvapuskurin ja puskurivirran yhdistelmää. Oikeanpuoleisessa kuvassa oktettiipuun solmujen järjestämisen ansiosta kameraa lähellä oleva seinä on piirretty korkeammalla pistetihedyllä 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 20 vasemmalla puolella näkyy, kuinka puskurivirta on piirtänyt koko näkyvillä olevan pistepilven samalla pistetihedyllä ja taimmainen seinä näyttää tarkemmalta kuin kameraa lähempiä 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 pisteteet pidetään näytönohjaimen muistissa, on niiden piirtäminen jokaisella ruudunpäivityksellä nopeaa. Kuvassa 21 on verrattu yleiskuvan piirtämistä kun yleiskuvapuskuri on valmiaksi näytönohjaimen muistissa siihen, kun pisteteet ladataan keskusmuistista puskurivirralla näytönohjaimelle. Yleiskuvapuskurin tapauksessa piirtoaika on mitattu piirtokomennon suorittamisesta siihen, että näytönohjain on saanut kaikki pisteteet piirrettyä ja merkattua synkronointiobjektiin käsitellyksi. Näin kaikki pisteteet on varmas-



Kuva 21: Worksite-pilvestä muodostetun puun viiden ylimmän tason visualisointiin käytetty aika mikrosekunteina puskurivirralla ja kun pisteet ovat valmiiksi yleiskuvapuskuissa 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 runsaasti laskenta-aikaa visualisointivaiheessa. Laitossuunnittelohjelmistossa tallennustilan tehokas käyttö ja pistepilven saaminen nopeasti katsottavaksi ovat tärkeää, joten olisi syytä tutkia, saadaanko kompression purkamista nopeutettua. Yksi mahdollisuus oli siäkeistää pisteiden lataamista niin, että yksi säie lukee pisteitä levyltä, toinen purkaa niiden kompressiota ja kolmas kopioi niitä näytönohjaimelle.

Oktettipuun solmujen järjestäminen ruudulle projisoidun koon mukaan vei 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 prioriteetijoonaan, josta toinen säie ottaa aina suurimman prioriteetin omaavan solmun piirrettäväksi.

Pistebudjetin käyttö pistepilveä visualisoitessa mahdollistaa interaktiivisen ruudunpäivitysnopeuden, mutta huonontaa piirretyn kuvan laatua. Kuvassa 22 näkyy ilmanvaihtooneen 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 [31] ehdottaman muokkautuvan pistekoon algoritmin.

Laitossuunnittelohjelmistossa 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 puskuereihin. Näin on mahdollista piirtää uudestaan vain kursori, kun käyttäjä liikuttaa hiirtä, minkä jälkeen pistepilven visualisointia voidaan jatkaa siitä mihin viimeksi jäätii. Oktettipuun solmujen järjestäminen tärkeyden mukaan on silti tarpeen. Kuten kuvasta 20



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

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

Mallinnustyössä pistepilvi täytyy usein piirtää useaan maisemaan samanaikaisesti. Käytännössä joudutaan harvoin tilanteeseen, jossa kamerat liikkuvat ja pistepilviä joudutaan päivittämään monessa maisemassa samaan aikaan. Kuitenkin olisi suotavaa, jos visualisoija piirtäisi ensin pistepilven yleiskuvan jokaiseen maisemaan, minkä jälkeen se kävisi maisemia läpi ja antaisi kullekin pieniin aikasiivun pistepilven tarkentamiseen. Samoin usean pistepilven tapauksessa ei saisi käydä niin, että yksi pilvi piirretään täyneen tarkkuuteensa ennenkuin toista on edes aloitettu.

## 7 Yhteenvetö

## 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 9, 1269 (2015)
- [4] H. Rauno *et al.*, *Siltojen 3D-suunnittelun- ja mittausprosessin kehittäminen ja käytöönottaminen (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. Pre gesbauer, *Consolidated Visualization of Enormous 3D Scan Point Clouds with Scanopy*, Proceedings of the 16th International Conference on Cultural Heritage and New Technologies 242 (2011)
- [7] F. Menna *et al.*, *3D DIGITIZATION OF AN HERITAGE MASTERPIECE - A CRITICAL ANALYSIS ON QUALITY ASSESSMENT*, ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences XLI-B5, 675 (2016)
- [8] J. Hecht, *Lidar for Self-Driving Cars*, Optics and Photonics News 29, 26 (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 F-230, Lado Sarai, New Delhi-110 030, India, 2005)
- [11] M. Piipponen, *3D-suunnittelun hyödyntäminen tehdassuunnittelussa*, Opinnäytetyö, Satakunnan ammattikorkeakoulu, 2012
- [12] AVEVA Laser Modeller, [https://www.digitalengineering247.com/pics/pdfs/AVEVA\\_L51\\_LaserModeller.pdf](https://www.digitalengineering247.com/pics/pdfs/AVEVA_L51_LaserModeller.pdf), viitattu 12.10.2019,
- [13] M. Yllikäinen, *Kolmiulotteisen visualisoinnin erityispiirteet laitossuunnitteluhjelmistoissa*, pro gradu -tutkielma, Informaatioteknologian laitos, Turun yliopisto, 2013
- [14] Leica Geosystems AG, [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](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,
- [15] Leica Geosystems AG, *Leica RTC360 Data Sheet*, <https://leica-geosystems.com/-/media/files/leicageosystems/products/datasheets/leica-rtc360-ds.ashx>, viitattu 6.1.2020, (2018)

- [16] J. Fabritius, *Terrestrial three-dimensional laser scanning in Aveva PDMS*, Opinnäytettyö, Tampereen ammattikorkeakoulu, 2009
- [17] H. Houshiar, J. Elseberg, D. Borrmann ja A. Nuchter, *A study of projections for key point based registration of panoramic terrestrial 3D laser scan*, Geo-spatial Information Science 18, (2015)
- [18] 3DTK, The 3D Toolkit, <http://slam6d.sourceforge.net/>, viitattu 14.12.2019,
- [19] P. J. Besl ja N. D. McKay, *A method for registration of 3-D shapes*, IEEE Transactions on Pattern Analysis and Machine Intelligence 14, 239 (1992)
- [20] C. Harris ja M. Stephens, *A combined corner and edge detector*, In Proc. of Fourth Alvey Vision Conference 147 (1988)
- [21] E. Rosten ja T. Drummond, *Fusing points and lines for high performance tracking*, 2, 1508 (2005)
- [22] D. G. Lowe, *Object recognition from local scale-invariant features*, 2, 1150 (1999)
- [23] M. Weinmann, *Reconstruction and Analysis of 3D Scenes: From Irregularly Distributed 3D Points to Object Classes* (Springer Publishing Company, Incorporated, 2016)
- [24] J. Giesen ja F. Cazals, *Delaunay Triangulation Based Surface Reconstruction: Ideas and Algorithms*, (2006)
- [25] M. Berger *et al.*, *A Survey of Surface Reconstruction from Point Clouds*, Computer Graphics Forum (2016)
- [26] R. Schnabel, R. Wahl ja R. Klein, *Efficient RANSAC for point-cloud shape detection*, Comput. Graph. Forum 26, 214 (2007)
- [27] C. M. Huang ja Y.-H. Tseng, *Plane fitting methods of LiDAR point cloud*, 1925 (2008)
- [28] C. Scheiblauer, *Interactions with Gigantic Point Clouds*, Väitöskirja, Institute of Computer Graphics and Algorithms, Vienna University of Technology, 2014
- [29] S. Rusinkiewicz ja M. Levoy, *QSplat: A Multiresolution Point Rendering System for Large Meshes*, Proceedings of SIGGRAPH 2000, (2001)
- [30] C. Dachsbacher, C. Vogelsgang ja M. Stamminger, *Sequential point trees*, ACM Transactions on Graphics 22, 657 (2003)
- [31] M. Schütz, *Potree: Rendering Large Point Clouds in Web Browsers*, pro gradu -tutkielma, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, 2016

- [32] 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 103 (2019)
- [33] M. Wimmer ja C. Scheiblauer, *Instant Points: Fast Rendering of Unprocessed Point Clouds*, Proceedings Symposium on Point-Based Graphics 2006 129 (2006)
- [34] J. Yang, R. Li, Y. Xiao ja Z.-G. Cao, *3D reconstruction from non-uniform point clouds via local hierarchical clustering*, (2017)
- [35] J. Davidsson, *Selvitystyö:Massiivisten pistepilviaineistojen Hallintajakelun näkökulmasta*, 3point Oy, Lapinlahdenkatu 16 00180 Helsinki (2019)
- [36] M. Wand *et al.*, *Interactive Editing of Large Point Clouds*, Symposium on Point-Based Graphics 2007 : Eurographics / IEEE VGTC Symposium Proceedings, Eurographics Association, 37-46 (2007) (2008)
- [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*, Smooth visualization of large point clouds, IADIS International Journal on Computer Science and Information Systems 11, 146 (2016)
- [39] M. Schütz, G. Mandlburger, J. Otepka ja M. Wimmer, *Progressive Real-Time Rendering of One Billion Points Without Hierarchical Acceleration Structures*, , 2019
- [40] Nvidia, <https://www.nvidia.com/en-gb/design-visualization/quadro/rtx-8000/>, viitattu 29.11.2019,
- [41] A. S. Tanenbaum ja H. Bos, *Modern Operating Systems*, 4th ed. (Prentice Hall PressUSA, 2014)
- [42] OpenGl, [https://www.khronos.org/opengl/wiki/Buffer\\_Object\\_Streaming](https://www.khronos.org/opengl/wiki/Buffer_Object_Streaming), viitattu 12.12.2019,
- [43] OpenGl, [https://www.khronos.org/opengl/wiki/Sync\\_Object](https://www.khronos.org/opengl/wiki/Sync_Object), viitattu 12.12.2019,
- [44] T. Akenine-Moller, T. Moller ja E. Haines, *Real-Time Rendering*, 2nd ed. (A. K. Peters, Ltd.Natick, MA, USA, 2002)