**Tuukka Varjus**

# Laitteistokiihdytteisen säteenseurannan hyödyntäminen reaaliaikaisessa renderöinnissä

Tietotekniikan pro gradu -tutkielma

22. toukokuuta 2023

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

**Tekijä:** Tuukka Varjus

**Yhteystiedot:** tuukkavarjus@gmail.com

**Ohjaajat:** Sanna Mönkölä ja Tuomo Rossi

**Työn nimi:** Laitteistokiihdytteisen säteenseurannan hyödyntäminen reaaliaikaisessa renderöinnissä

**Title in English:** Hardware accelerated ray tracing in real-time rendering

**Työ:** Pro gradu -tutkielma

**Opintosuunta:** Ohjelmisto- ja tietoliikennetekniikka

**Sivumäärä:** 167+0

**Tiivistelmä:** NVIDIA:n esitellessä maailman ensimmäiset laitteistokiihdytteistä säteenseurantaa tukevat Turing-arkkitehtuurin näytönohjaimet vuonna 2018, mahdollisti se säteenseurannan hyödyntämisen reaaliaikaisessa renderöinnissä. Säteenseurannan ollessa tunnetusti laskennallisesti hyvinkin raskas verrattuna esimerkiksi rasterointiin, on tutkielman tarkoitus selvittää missä määrin ja millä menetelmin ja tavoin säteenseurantaa kyetään hyödyntämään reaaliaikaisessa renderöinnissä ja mitä haasteita siihen liittyy. Säteenseurannan suorituskyvyn esittelemiseksi tutkielman empiirisessä osiossa arvioitiin säteenseurannan suorituskykyä ja siihen vaikuttavia tekijöitä NVIDIA:n Turing-arkkitehtuurin näytönohjaimella. Varsinaisia menetelmiä ja tapoja laitteistokiihdytteisen säteenseurannan hyödyntämiseen ja siihen liittyviä haasteita tarkasteltiin, niin nykyisissä videopeleissä perustuen pääsääntöisesti GDC konferenssitallenteisiin, kuin myös kirjallisuuskatsauksen avulla. Videopeleissä säteenseurannan laskennallinen vaativuus näkyy rasteroinnin säilymisenä päätekniikkana renderöinnissä säteenseurannan täydentäessä valittujen valon ilmenemismuotojen kuvaamista. Kirjallisuuskatsauksessa tutkimuksen havaittiin pääsääntöisesti keskittyvän näytteiden tehokkaaseen hyödyntämiseen ja häiriönpoistoon, joihin lukeutuvia menetelmiä tutkielmassa tarkemmin esitellään. Merkittävimpinä havaittuina haasteina voidaan pitää monivalo-ongelmaa, sekä menetelmien ajallista vakautta ja harhattomuutta.

**Avainsanat:** säteenseuranta, polunseuranta, Turing-arkkitehtuuri, Monte Carlo -integrointi,

häiriönpoisto, Vulkan API

**Abstract:** Since 2018 when NVIDIA introduced the world's first GPUs supporting hardware accelerated ray tracing, ray tracing has become available in real-time rendering. As ray tracing is known for its computational complexity, the purpose of this study is to introduce methods and ways to utilize ray tracing in real-time rendering, as well related notable challenges. To introduce the performance of hardware accelerated ray tracing and to study performance affecting factors more closely, empirical study using NVIDIA Turing architecture GPU was conducted. The methods and ways to utilize ray tracing in real-time rendering and related challenges were explored through literature review and through studying ray tracing usage in current video games mainly by reviewing GDC presentations. In reviewed video games the computational complexity of ray tracing was seen as usage of rasterization as primary rendering technique, whereas ray tracing was used for complementing selected effects. In literature review the research around real-time ray tracing was seen as mostly focusing on efficient sampling and denoising techniques, which are introduced in detail in this study. The most notable challenges detected in real-time ray tracing are many-lights problem and temporal stability and unbiasedness of used techniques.

**Keywords:** ray tracing, path tracing, Turing architecture, Monte Carlo integration, denoising, Vulkan API

# Kuviot

# Taulukot

# Sisällys

# 1 Johdanto

Säteenseurannan menetelmä on tunnettu pitkään ja menetelmän historia ulottuukin pitkälle aikaa ennen tietokoneita. Menetelmän katsotaan syntyneen jo 1500-luvulla ja sen keksijänä pidetään renessanssiajan taidemaalari Albrecht Düreria (Haines ja Akenine-Möller 2019). Myös tietokonegrafiikassa menetelmällä on suhteellisen pitkä historia Arthur Appelin (1968) ollessa ensimmäinen, joka kuvaili säteenseurannan[1] käytön tietokonegrafiikassa. Merkittävimpänä yhtälönä säteenseurannan kannalta voidaan pitää Kajiyan (1986) esittelemää valon kulkeutumista kuvaavaa renderöintiyhtälöä (engl. *rendering equation*, *light transport equation*). Samassa artikkelissa yhtälön ratkaisemiseksi esitellään säteenseurantaa ja Monte Carlo -integrointia yhdistävä polunseurantamenetelmä (engl. *path tracing*). Tässä tutkielmassa säteenseurantaa suureksi osaksi käsitellään juurikin polunseurantamenetelmän kautta.

Säteenseurannan etuna muihin renderöintitekniikoihin voidaan pitää sen yleiskäyttöisyyttä ja selkeyttä, sillä säteenseuranta luonnostaan soveltuu useimpien valon ilmenemismuotojen kuvaamiseen. Kun säteenseurantaa käytetään osana polunseurantamenetelmää, kyetään sillä tuottamaan fotorealistisia kuvia. Näistä syistä ei-reaaliaikaisessa renderöinnissä kuten animaatiossa, säteenseurantaa on hyödynnetty jo pidemmän aikaa Pixarin Bugs Life animaation (1998) ollessa varhaisimpia, jossa osittain hyödynnettiin säteenseurantaa (Christensen ja Jarosz 2016). Ensimmäinen polunseurannalla toteutettu täysimittainen elokuva oli vuonna 2006 ilmestynyt Monster house (Christensen ja Jarosz 2016). Nykyisin elokuvatuotannossa polunseurantamenetelmä onkin korvannut muita renderöinti menetelmiä. Tästä yhtenä esimerkkinä voidaan pitää Pixarin RenderMan[2] (Christensen ym. 2018) renderöintiohjelmiston luopumista Reyes[3] tyylisestä renderöinnistä siirtyen puhtaasti polunseurannalla toteutettuun renderöintiin vuonna 2016 RenderMan 21 julkaisun myötä (“RenderMan 21 Documenta-

1. Arthur Appel esitti säteenseurannan ei-rekursiivisena, mikä osassa myöhempää kirjallisuutta tunnetaan säteensuuntaamisena (engl. *ray casting*).

2. Pixarin RenderMan renderöintiohjelmistolla on toteutettu muun muassa ensimmäinen täysimittainen tietokoneella luotu (engl. *computer-generated*, *CG*) animaatio Toy Story vuonna 1995.

3. Reyes (Render Everything You Ever Saw) (Cook, Carpenter ja Catmull 1987) on Lucasfilm Ltd:llä kehitetty renderöinti arkkitehtuuri nopeuttamaan tietokonegrafiikan tuottamista sen aikaisessa elokuvatuotannossa, jossa tietokonegrafiikka käytettiin pääsääntöisesti lyhyisiin osuuksiin elokuvissa.

tion” 2016).

Myös reaaliaikaisessa renderöinnissä säteenseurannan yleiskäyttöisyys ja selkeys ovat herättäneet alan piireissä kiinnostusta. Tämä huolimatta siitä, että reaaliaikaisessa renderöinnissä yleisesti käytetyt rasterointipohjaiset menetelmät kykenevätkin tuottamaan visuaalisesti näyttävän approksimaation useimmille valon ilmenemismuodoille. Säteenseurannan laajempaa käyttöä reaaliaikaisessa renderöinnissä on kuitenkin rajoittanut sen laskennallinen vaativuus. Säteenseurannan ja erityisesti reaaliaikaisen säteenseurannan kenties merkittävin läpimurto tapahtui vuonna 2018 NVIDIA:n esitellessä maailman ensimmäiset laitteistokiihdytteistä säteenseurantaa tukevat Turing-arkkitehtuurin näytönohjaimet (Corporation 2018). Läpimurtoa edesauttoi NVIDIA:n pitkä kehitystyö OptiX säteenseuranta API:n parissa. Vulkan API:lle täysi tuki laitteistokiihdytteiselle säteenseurannalle tuli loppuvuodesta 2020 (Sjöholm, Jukarainen ja Aalto 2021, s. 214). Säteenseurannan laitteistokiihdytyksen myötä useimmat pelimoottorit ovat alkaneet tukemaan säteenseurantaa (“NVIDIA RTX: List Of All Games, Engines And Applications Featuring GeForce RTX-Powered Technology” 2022), mikä näkyy säteenseurannan hyödyntämisenä videopeleissä. Vuonna 2020 julkaistujen Xbox Series X ja Playstation 5 -pelikonsolien sisältämien laitteistokiihdytteistä säteenseurantaa tukevien AMD:n RDNA2-arkkitehtuurin näytönohjaimien myötä säteenseuranta on tullut mahdolliseksi entistä laajemmalle yleisölle.

Laitteistokiihdytyksen tuki, säteenseurannan pitkä historia ja entistä laajempi tavoitettavuus tekevät säteenseurannasta mielenkiintoisen tutkimuskohteen. Lisäksi riittävästi aikaa on ehtinyt kulua laitteistokiihdytyksen tuen julkaisemisesta, jotta aihe on kerännyt paljon uutta tutkimustietoa. Tämän tutkielman tarkoituksena on esitellä niitä menetelmiä, joiden kautta säteenseurantaa kyetään hyödyntämään reaaliaikaisessa renderöinnissä, sekä teknologia säteenseurannan taustalla, vastaamalla tutkimuskysymykseen: "Minkälaisten menetelmien avulla säteenseurantaa kyetään hyödyntämään reaaliaikaisessa renderöinnissä ja mitä haasteita siihen liittyy?". Lisäksi tutkielman tarkoituksena on esitellä laitteistokiihdytteisen säteenseurannan suorituskykyä ja siihen vaikuttavia tekijöitä laitteistokiihdytteisen säteenseurannan keskiössä olevan kiihdytysrakenteen (engl. *acceleration structure*) osalta vastaten tutkimuskysymykseen "Mikä on laitteistokiihdytteisen säteenseurannan suorituskyky ja mitkä tekijät siihen vaikuttavat?".

Loppuosa tutkielmasta jakautuu seitsemään lukuun. Luvussa 2 esitellään lyhyesti tutkielman kannalta merkittävät valon fysikaaliset ominaisuudet ja suureet, sekä keskeisimmät tietokonegrafiikassa esiintyvät käsitteet ja tapa, jolla valo tietokonegrafiikassa kuvataan. Luvussa 3 esitellään lyhyesti erityisesti reaaliaikaisessa renderöinnissä valon kuvaamiseen laajasti käytetty renderöintitekniikka: rasterointitekniikka, sekä rasterointipohjaisia menetelmiä. Lisäksi luvussa esitellään säteenseurantatekniikka, sekä rasterointitekniikkan ja säteenseurantatekniikan eroavaisuuksia. Luvussa 4 esitellään säteenseurantatekniikkaan perustuva, tutkielman kannalta merkittävä menetelmä, polunseurantamenetelmä. Luvussa 5 puolestaan syvennytään laitteistokiihdytteiseen säteenseurantaan tarkastellen sitä mahdollistavaa laitteistoa ja ohjelmointirajapintaa keskittyen NVIDIA:n Turing-arkkitehtuuriin ja Vulkan API:in. Luvussa 6 tutkitaan laitteistokiihdytteisen säteenseurannan suorituskykyä, sekä kiihdytysrakenteen ominaisuuksien vaikutusta suorituskykyyn Vulkan API:lla toteutetussa koeympäristössä. Luvussa 7 tarkastellaan miten nykyisissä videopeleissä hyödynnetään säteenseurantaa. Luvussa 8 puolestaan esitellään reaaliaikaiseen renderöintiin tarkoitettuja säteenseurantaa hyödyntäviä menetelmiä kirjallisuuskatsauksen muodossa. Lopuksi luvussa 9 esitellään johtopäätökset ja pohditaan säteenseurannan tulevaisuutta reaaliaikaisessa renderöinnissä.

# 2 Mitä valo on?

Usein näkee mainittavan säteenseurannan perustuvan valon fysikaaliseen käyttäytymiseen. Tämä lieneekin oikeutettua, jos verrataan säteenseurantaa perinteisempiin reaaliaikaisissa renderöinnissä käytettäviin menetelmiin ja tarkastellaan valoa sädeoptiikkaan eli geometrisen valo-opin (engl. *geometrical optics*) näkökulmasta. Vaikka tietokonegrafiikan sovelluksissa tarkoitus ei yleensä olekaan valon täsmällinen fysikaalinen mallintaminen, vaan riittävän hyvän approksimaation tuottaminen valon käyttäytymisestä, ovat valon fysiikan käsitteet ja lainalaisuudet pohjana useimmissa tietokonegrafiikan menetelmissä. Tämä tekeekin valo-opin tuntemisesta tärkeän osan niin nykyisten menetelmien ymmärtämistä, kuin myös uusien menetelmien kehittämistä. Tässä luvussa esitellään vain pintaraapaisu valon fysiikasta. Kattavasti valon fysiikkaa esitellään muun muassa kirjassa (Roychoudhuri, Kracklauer ja Creath 2008).

Fysiikassa valo määritellään sähkömagneettiseksi säteilyksi, joka puolestaan on säteilyn etenemissuuntaa vastaan kohtisuorasti kaikkiin suuntiin tapahtuvaa värähtelyä sähkö- ja magneettikentissä. Sähkömagneettisen säteilyn ja siten valon ominaisuuksien kuvaamiseen ei olemassa yksiselitteistä teoriaa, vaan valon voidaan katsoa sisältävän, sekä aalto-, että hiukkasominaisuuksia. Kvanttimekaniikassa tämä valon, sekä yleisemmin aineen ja sähkömagneettisen säteilyn ja täten sähkömagneettisen säteilyn alkeishiukkasen, fotonin (engl. *photon*), moniominaisuus tunnetaan nimellä aalto-hiukkasdualismi (engl. *wave–particle duality*). Historian saatossa molemmat, sekä aalto- että hiukkasmalli ovat keränneet taaksensa useita tunnettuja fyysikkoja ja löydöksiä. Aalto-ominaisuuksilla kyetään esimerkiksi selittämään valon interferenssi, diffraktio ja polarisoituminen, mutta ei valosähköistä ilmiötä, joka puolestaan kyetään selittämään valon hiukkasominaisuuksilla.

Riippumatta siitä, ajatellaanko valoa ja fotoneja aaltoina vai hiukkasina, voidaan fotonien kuljettamalle energialle johtaa yhteys havaittaviin väreihin: yksittäisen fotonin kuljettama energia on kääntäen verrannollinen fotonien muodostaman aallon pituuteen. Eri aineilla puolestaan on erilainen kyky absorboida ja heijastaa eri aallonpituuksia. Esimerkiksi, jos jokin aine heijastaa punaisen valon aallonpituutta ja absorboi muita valon aallonpituuksia, havainnoidaan kyseinen aine punaisena. Ihmisen silmä kykenee havaitsemaan elektromagneettista

säteilyä noin[1] aallonpituuksilla 380-780 nm (Dutre ym. 2006, 19)(Keating 2002, 5), mikä vastaa fotonin energiaa 3.3-1.6eV.

Se miten ihminen aistii valoa verkkokalvon eli retinan aistinsolujen: sauva- ja tappisolujen avulla ja miten valon aiheuttama valoärsyke muuntuu signaaliksi, joka kulkeutuu näköhermoa pitkin talamuksen ulommalle polvitumakkeelle ja sieltä primääriselle näköaivokuorelle ja miten meille lopulta muodostuu oikeinpäin oleva kuva havaitsemastamme maailmasta on monimutkainen ja yksityiskohdiltaan tuntematon prosessi (Keating 2002). Fysiologian ja havaintopsykologian keskittyessä fotoneiden aiheuttamiin ärsykkeisiin silmässä ja niistä syntyviin signaaleihin ja kvanttimekaniikan pyrkiessä selittämään fotonien ominaisuudet ja käyttäytyminen hyvinkin yksityiskohtaisesti, on tietokonegrafiikan näkökulmasta mielekkäämpää tarkastella valoa suhteellisen yksinkertaistettuna.

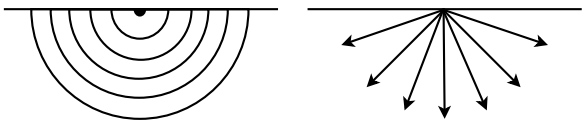
Sen sijaan, että valon ominaisuuksia kuvattaisiin molekyyli- tai hiukkastasolla, antaa aliluvussa 2.1 esiteltävä sädeoptiikka riittävän kuvauksen valon käyttäytymisestä reaaliaikaisen tietokonegrafiikan ja säteenseurannan näkökulmasta. Siinä, missä sädeoptiikka kuvaa valon käyttäytymistä geometrisesti, esitellään aliluvussa 2.2 valolle laskennassa käytettävät radiometrian suureet. Aliluvussa 2.3 esitellään radiometriasta johdettava funktio kuvaamaan erilaisten aineiden pinnoilla tapahtuvaa valon heijastumista. Aliluvussa 2.4 puolestaan esitellään tapa, jolla valon ominaisuuksia kuvataan tietokonegrafiikassa, sekä siihen liittyvää termistöä.

## 2.1 Sädeoptiikka

Geometrisesti valon eteneminen voidaan kuvata aaltorintamana tai säteinä. Kun valon etenemistä kuvataan valonsäteinä, puhutaan sädeoptiikasta. Vaikka sädeoptiikan avulla ei kyetäkään kuvaamaan valon aalto-ominaisuuksia kuten diffraktiota ja interferenssiä, on sädeoptiikka usein riittävä[2] geometrinen tulkinta valolle tietokonegrafiikan yhteydessä. Sädeoptiikassa valonsäteet osoittavat aaltorintaman normaalin suuntaan eli valon etenemissuunnan

1. Havaitseminen on yksilöllistä ja eri lähteissä näkee käytettävän eri raja-arvoja näkyvän valon aallonpituuksille.

2. Valon aalto-ominaisuudet kyetään yleensä havaitsemaan vain, kun valo on vaikutuksissa suhteellisen pienten kappaleiden kanssa kuten esimerkiksi Youngin kaksoisrakokokeessa (Mobley 2005).



Kuvio 1. Pistemäisen valolähteen synnyttämä valo esitettyä aaltorintamana ja säteinä.

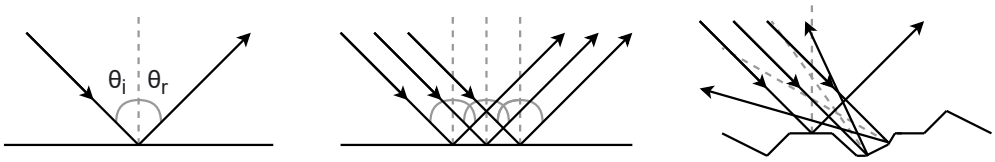
mukaisesti, mikä on havainnollistettu kuviossa 1. Siinä, missä sädeoptiikan avulla voidaan tarkastella erilaisten linssien ja peilien toimintaa, esitellään tässä aliluvussa säteenseurannan kannalta kaksi merkittävää sädeoptiikan perustavanlaatuista lakia: heijastus- ja taittumislaki.

Heijastus- ja taittumislakien esittelemiseksi tässä luvussa väliaineiden oletetaan olevan isotrooppisia ja homogeenisiä, jos ei erikseen mainita, jolloin väliaineessa matkaavan säteen oletetaan kulkevan lineaarisesti ja kohdatessaan toisen homogeenisen aineen rajapinnan valonsäde joko heijastuu, absorboituu tai läpäisee aineen. Heijastuslaki määrittelee pinnalle saapuvan valon tulokulman $\theta_i$ ja heijastuskulman $\theta_r$ yhtäsuuruuden

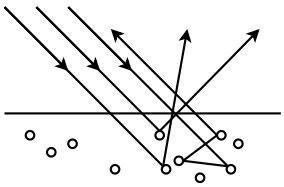
$$\theta_i = \theta_r, \tag{2.1}$$

missä tulokulma $\theta_i$ ja heijastuskulma $\theta_r$ ovat pinnalle saapuvan valonsäteen ja siitä heijastuvan valonsäteen ja pinnan normaalin väliset kulmat. Heijastuslailla voidaan selittää diffuusi heijastukset (engl. *diffuse reflection*) karheille pinnoille, kun aineen pinnan epätasaisuudet ovat valon aallonpituutta suurempia. Tällöin joukko toisiaan lähellä matkaavia säteitä, jotka saapuvat aineen pintaan samasta tulokulmasta, voivat pinnan kohdatessaan heijastua täysin eri suuntiin heijastuslain mukaisesti peilaten pinnan paikallista normaalia. Heijastuslaki, peiliheijastus ja diffuusi heijastus ovat esitetty kuviossa 2. Heterogeenisella väliaineella syy diffuusi heijastukselle on usein väliaineen pinnalla tapahtuva sironta (engl. *subsurface scattering*), joka on seurausta siitä, että valonsäde heijastuu aineen epätasaisuuksista. Pinnalla tapahtuva sironta esitetään kuviossa 3.

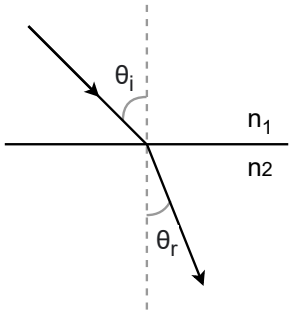
Siinä, missä valoa läpäisemätön aine absorboi valoa ja aiheuttaa fotonin energian muuntumisen aineen sisäenergiaksi, valoa läpäisevät väliaineet aiheuttavat valon taittumisen väliaineiden rajapinnalla. Snellin laki eli taittumislaki määrittää taittumiskulman suuruuden valon




Kuvio 2. Heijastumislaki (vasemmalla) ja sen toteutuminen peiliheijastuksille (keskellä) ja diffuusi heijastuksille (oikealla).



Kuvio 3. Pinnalla tapahtuva sironta.




Kuvio 4. Taittumislaki.

tulokulman ja väliaineiden taitekertoimien[3] avulla. Taittumislain mukaisesti väliaineiden, joiden taitekertoimet ovat $n_1$ ja $n_2$, rajapinnalle saapuvan valon tulokulman $\theta_i$ ja taittumiskulman $\theta_r$ välillä pätee

$$n_1 \sin(\theta_i) = n_2 \sin(\theta_r). \tag{2.2}$$

Tällöin, mikäli valo etenee taitekertoimeltaan alhaisemmasta väliaineesta taitekertoimeltaan korkeampaan väliaineeseen, valon säde taittuu kohti rajapinnan normaalia. Jos puolestaan valo etenee taitekertoimeltaan korkeammasta väliaineesta alhaisempaan väliaineeseen, valo taittuu poispäin rajapinnan normaalista. Taittumislaki esitetään kuviossa 4. Ilman, veden ja erilaisten lasien taitekertoimet 588 nm aallonpituudelle ovat 1,0003, 1,33 ja noin 1,50-1,90 vastaavassa järjestyksessä (Keating 2002, 8).

On hyvä huomata, että valo ei juurikaan koskaan taitu kokonaisuudessaan, vaan vähintään

3. Taitekerroin kertoo aineen kyvyn taittaa valoa. Jossain yhteyksissä puhutaan optisesta tiheydestä, joka määräytyy taitekertoimen mukaan, eli mitä suurempi taitekerroin väliaineella on, sitä optisesti tiheämpää se on.

osa siitä heijastuu. Heijastussuhde saapuvalle valolle saadaan Fresnel kertoimien avulla (engl. *Fresnel coefficients*), joihin vaikuttavat valon tulokulman lisäksi aineiden taitekertoimet ja valon polarisaatio. Tietokonegrafiikassa Fresnel kertoimien ratkaisemisen sijasta usein käytetään Schlickin approksimaatiota (engl. *Schlick's approximation*) (Schlick 1994) heijastussuhteen approksimoimiseksi, mistä syystä Fresnelin kertoimia ei tutkielmassa esitetä. Säteenseurannan kannalta mielenkiintoisempaa on yksi heijastussuhteen erikoistapauksista: kokonaisheijastuminen (engl. *total internal reflection*). Kokonaisheijastuminen tapahtuu tulokulman ylittäessä niin kutsutun kriittisen kulman, $\theta_c$, jolloin sen sijaan, että valo taittuisi kahden väliaineen rajapinnalla, valo heijastuu kokonaisuudessaan. Kriittinen kulma saadaan määriteltyä taittumislain avulla, kun yhtälöön sijoitetaan $\theta_r = 90°$:

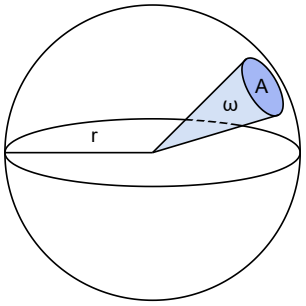
$$\theta_c = \arcsin\left(\frac{n_2}{n_1}\right). \tag{2.3}$$

## 2.2 Valon fysikaaliset suureet

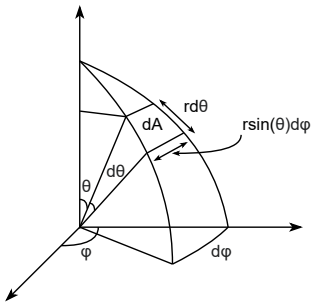
Valoa voidaan mitata sekä radiometrian, että fotometrian keinoin. Siinä, missä fotometria keskittyy mittaamaan valoa rajattuna näkyvän valon spektrille ja ihmisen näkökyvyn huomioiden, keskittyy radiometria absoluuttisesti mittaamaan elektromagneettista säteilyä sisältäen näkyvän valon spektrin. Myös tietokonegrafiikan kirjallisuudessa useimmiten[4] näkee käytettävän radiometrian suureita fotometrian suureiden sijasta. Täten myös tässä tutkielmassa pitäydytään radiometrian suureissa. Tämän luvun tarkoituksena on lähteisiin (Yliantila ja Jokela 2009) ja (Dutre ym. 2006) perustuen esitellä säteenseurannan kannalta oleellisimmat optisen radiometrian suureet ja käsitteet, joita tässä tutkielmassa tullaan tarvitsemaan esimerkiksi luvussa 2.4 esiteltävän renderöintiyhtälön 2.23 ymmärtämiseksi.

Säteenseurannan kannalta oleellisimmat radiometrian suureet ovat irradianssi (engl. *irradiance*) ja radianssi (engl. *radiance*), joita tarvitaan seuraavassa luvussa esiteltävän kaksisuuntaisen heijastusjakaumafunktion (engl. *bidirectional reflectance distribution function*, BRDF) esittelemiseksi. Ennen näiden suureiden esittelemistä, määritellään ensiksi kuviossa

4. Poikkeuksena esimerkiksi artikkelit (Schied ym. 2017) ja (Marrs ym. 2018), joissa käytetään fotometrian luminanssi suuretta radiometrian radianssi suureen sijasta.




Kuvio 5. Avaruuskulma $\omega$.




Kuvio 6. Pinta-ala-alkio pallokoordinaatistossa, missä $\varphi$ on atsimuuttikulma ja $\theta$ on napakulma.

5 esitetty avaruuskulma $\omega$ (engl. *solid angle*), joka tarvitaan radianssin määrittelemiseksi.

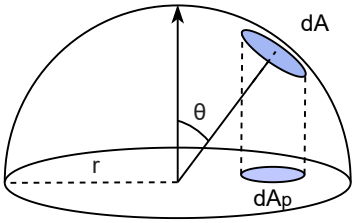
Avaruuskulma voidaan mieltää tasokulman kaksiulotteiseksi vastineeksi. Avaruuskulma kuvaa pallon keskipisteestä avautuvan kartiopinnan rajaaman pallon pinta-alan suhteessa pallon säteen neliöön [5]:

$$\omega = \frac{A}{r^2}. \tag{2.4}$$

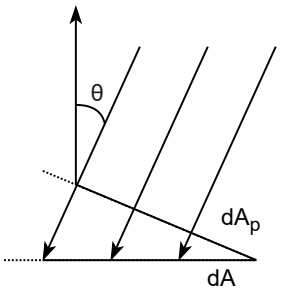
Avaruuskulman SI-järjestelmän mukainen yksikkö on steradiaani (*sr*) ja pallon pinta-alan $A = 4\pi r^2$ mukaisesti täysi avaruuskulma on $4\pi$ steradiaania. Koska säteenseurannassa käsitellään säteitä, joiden poikkipintainen pinta-ala voidaan kuvitella infinitesimaaliseksi[6], säteenseurannan kannalta onkin mielekkäämpää käsitellä yksittäisiä infitesimaalisia pinta-ala-alkiota $dA$ ja täten infitesimaalisia avaruuskulmia $d\omega$. Tästä seuraa, että pallon pinta-ala-alkio voidaan kuvitella tasaisena pintana, jonka normaali osoittaa kohti pallon keskipistettä. Tämä antaa puolestaan mahdollisuuden esittää pinta-ala-alkion pinta-ala $dA$ pallokoordinaateissa, kuten on havainnollistettu kuviossa 6. Tästä seuraa, että

$$dA = rd\theta \cdot \sin(\theta) rd\varphi = r^2 \sin(\theta) d\theta d\varphi. \tag{2.5}$$

5. Toisin sanoen avaruuskulma on pinta-ala yksikköpallolla, sillä $A_{yksikko} = \frac{A}{1^2}$.
6. Äärettömän pieni.




Kuvio 7. Projektioavaruuskulma.




Kuvio 8. Radiometristä sädettä vasten projisoitu pinta-ala.

Tällöin infitesimaalinen avaruuskulma (engl. *differential solid angle*) on

$$d\omega = \frac{r^2 \sin(\theta) d\theta d\varphi}{r^2} = \sin(\theta) d\theta d\varphi, \tag{2.6}$$

jota käyttäen voidaan puolipallon suuruinen avaruuskulma esittää integraalina

$$\int_{\varphi=0}^{2\pi} \int_{\theta=0}^{\frac{\pi}{2}} \sin(\theta) d\theta d\varphi = 2\pi. \tag{2.7}$$

Avaruuskulmaa ja sitä vastaavaa pallon pinta-alaa useammin tietokonegrafiikassa käsitellään projektioavaruuskulmaa $\Omega$[7] ja projektiopinta-alaa $A_p$, jotka esitetään kuviossa 7. Projektioavaruuskulma on puoliavaruuden[8] rajoittamalle ympyrälle projisoidun pinta-alan suhde kyseisen ympyrän säteeseen:

$$d\Omega = \frac{dA_p}{r^2} = \frac{\cos(\theta) dA}{r^2} = \cos(\theta) d\omega. \tag{2.8}$$

Tällöin enimmäisprojektioavaruuskulmaksi puoliavaruudesta saadaan ympyrän pinta-ala $\pi$ steradiaania[9]. Radianssi ja irradianssi ovat riippuvaisia säteilytehosta $\Phi$ (engl. *radiant flux*).

7. Tässä luvussa $\Omega$ symbolilla merkitään projektioavaruuskulmaa ja $\omega$ symbolilla avaruuskulmaa kuten myös Ylianttilan ja Jokelan (Ylianttila ja Jokela 2009) kirjassa.

8. Tutkielmassa käytetään termiä puoliavaruus tarkoittaen pinta-ala-alkion normaalin suuntaista puolipalloa. Näin ollen, puoliavaruus sisältää kaikki avaruuskulmat, joista pinta-ala-alkio on nähtävissä.

9. $\int_{\varphi=0}^{2\pi} \int_{\theta=0}^{\frac{\pi}{2}} \cos(\theta)\sin(\theta) d\theta d\varphi = 2\pi(\frac{\sin^2(\frac{\pi}{2})}{2} - \frac{\sin^2(0)}{2}) = 2\pi(\frac{1}{2} - 0) = \pi.$

Säteilyteho on energian määrä $Q$[10] tietyssä ajassa:

$$\Phi = \frac{dQ}{dt}. \tag{2.9}$$

Säteilytehon yksikkö on $W$. Säteilytehosta voidaan pinta-ala-alkiolle johtaa irradianssi $E$, joka kuvaa pinta-ala-alkiolle puoliavaruudesta saapuvaa säteilytehoa

$$E = \frac{d\Phi}{dA}, \tag{2.10}$$

kun taas eksitanssi (engl. *radient exitence*) $M$ kuvaa pinnalta puoliavaruuteen lähtevää säteilytehoa

$$M = \frac{d\Phi}{dA}. \tag{2.11}$$

Näin ollen irradianssin, sekä eksitanssin yksikkö on $\frac{W}{m^2}$. Radianssi $L$ puolestaan kuvaa radiometrisessä säteessä[11] kulkevaa säteilytehoa, jolloin radianssi on säteilyteho avaruuskulmaa ja projisoitua pinta-alaa kohden. Pinta-ala-alkiolle radianssi voidaan täten esittää projektiopinta-alaa käyttäen muodossa

$$L = \frac{d^2\Phi}{dA_p d\omega} = \frac{d^2\Phi}{\cos(\theta) dA d\omega}, \tag{2.12}$$

missä $\cos(\theta)dA$ on radiometrisen säteen kulkusuuntaa vasten projisoitu pinta-ala, mikä on havainnollistettu kuviossa 8. Tästä seuraa, että radianssin yksikkö on $\frac{W}{m^2 sr}$. Radianssin avulla pinta-ala-alkion irradianssi puoliavaruudesta voidaan esittää integraalina

$$E = \int_{2\pi} L\cos(\theta) d\omega \tag{2.13}$$

ja pinta-ala-alkion eksitanssi integraalina

10. Energian määrä $Q = \frac{hc}{\lambda}$, missä $h$ on Planckin vakio, $c$ on valonnopeus ja $\lambda$ on valon aallonpituus.
11. Radiometrinen säde määritellään säteilynä, joka kulkee kahden infinitesimaalisen pinta-ala-alkion välillä.

$$M = \int_{2\pi} L\cos(\theta)d\omega. \tag{2.14}$$

## 2.3 Kaksisuuntainen heijastusjakaumafunktio

Radiometrian suureista voidaan johtaa tietokonegrafiikan kannalta merkittävä funktio: kaksisuuntainen heijastusjakaumafunktio (engl. *bidirectional reflectance distribution function*, *BRDF*). Kaksisuuntainen heijastusjakaumafunktio kertoo pinta-ala-alkiolle tulokulmasta $\omega_i$ saapuvan irradianssin ja pinnalta puoliavaruuteen heijastuskulmassa $\omega_r$ heijastuvan radianssin suhteen. Kaksisuuntaisessa heijastusjakaumafunktiossa oletetaan, että piste johon valo saapuu ja piste, josta valo heijastuu ovat samoja[12][13], jolloin kaksisuuntainen heijastusjakaumafunktio voidaan esittää kaksiulotteisena muodossa

$$f_r(\omega_i, \omega_r) = \frac{dL(\omega_r)}{dE(\omega_i)} = \frac{dL(\omega_r)}{L(\omega_i)\cos(\theta)d\omega_i}, \tag{2.15}$$

missä $\theta$ on valon tulokulman ja pinta-ala-alkion normaalin välinen kulma. Molemmat infitesimaaliset avaruuskulmat tulokulma $\omega_i$ ja heijastuskulma $\omega_r$ voidaan esittää myös pallokoordinaatistossa atsimuuttikulman ja napakulman avulla, jolloin kaksisuuntaisen heijastusjakaumafunktio esitetään neliulotteisena funktiona. Funktion arvon riippuessa atsimuuttikulmasta, on kaksisuuntainen heijastusjakaumafunktio anisotrooppinen. Puolestaan, jos funktion arvo ei ole riippuvainen atsimuuttikulmasta, on kaksisuuntainen heijastusjakaumafunktio isotrooppinen. Kaksisuuntaisen heijastusjakaumafunktion määritelmän 2.15 mukaisesti kokonaisuudessaan heijastettu radianssi avaruuskulmalle $\omega_r$ voidaan esittää muodossa

$$L(\omega_r) = \int_{2\pi} f_r(\omega_i, \omega_r)L(\omega_i)\cos(\theta)d\omega_i. \tag{2.16}$$

12. Jos valo heijastuu eri pisteestä, kuin mihin se saapuu, on käytetään kaksisuuntaista pinnan sironta heijastus funktiota (engl. *bidirectional surface scattering reflectance distribution function*, *BSSRDF*) heijastuksen määrittelemiseen, josta kaksisuuntainen heijastusjakaumafunktio on erityistapaus (Dutre ym. 2006, 32).

13. Tästä seuraan, että esimerkiksi kuviossa 3 esitettyä pinnalla tapahtuvaa sirontaa ei tarkasti kyetä mallintamaan, mikä reaaliaikaisessa tietokonegrafiikassa ei useimmiten ole tarpeellistakaan, vaan kaksisuuntainen heijastusjakaumafunktio antaa riittävän approksimaation valon heijastumisesta aineen pinnalta.

Kaksisuuntaisella heijastusjakaumafunktiolla on muutama tärkeä ominaisuus, joista yksi on vastavuoroisuus (engl. *Helmholtz reciprocity*)

$$f_r(\omega_i, \omega_r) = f_r(\omega_r, \omega_i). \tag{2.17}$$

Vastavuoroisuuden ohella toinen tärkeä ominaisuus on energian säilyminen (engl. *energy conservation*), mikä tarkoittaa, että pinta-ala-alkion heijastama energia puoliavaruuteen ei voi ylittää sille saapuvaa energiaa. Hyödyntämällä eksitanssin määritelmää 2.14 ja kaksisuuntaisen heijastusjakaumafunktion määritelmää 2.15, voidaan kokonaiseksitanssi esittää muodossa

$$M = \int_{2\pi}\int_{2\pi} f_r(\omega_i, \omega_r) L(\omega_i) \cos(\theta_r) \cos(\theta_i) d\omega_r d\omega_i. \tag{2.18}$$

Tällöin energian säilymisen mukaisesti kaksisuuntaiselle heijastusjakaumafunktiolle pätee

$$\frac{\int_{2\pi}\int_{2\pi} f_r(\omega_i, \omega_r) L(\omega_i) \cos(\theta_r) \cos(\theta_i) d\omega_r d\omega_i}{\int_{2\pi} L(\omega_i) \cos(\theta_i) d\omega_i} \leq 1. \tag{2.19}$$

Kun tiedetään, että kaksisuuntainen heijastusjakaumafunktio on riippumaton eri avaruuskulmista saapuvasta radianssista, voidaan valita sopiva delta-funktio[14] esittämään saapuvan radianssin jakaumaa[15], jolloin energian säilyminen voidaan esittää monessa lähteessä esitetyssä muodossa

$$\forall \alpha : \int_{2\pi} f_r(\alpha, \omega_r) \cos(\theta_r) d\omega_r \leq 1, \tag{2.20}$$

missä $\alpha = \omega_i$.

14. Nimestään huolimatta delta-funktio ei ole funktio.

15. Intuitiivisesti yksinkertaistaen voidaan jakauma ajatella infitesimaalisena avaruuskulmana saapuvalle radianssille.

## 2.4 Valo tietokonegrafiikassa

Siinä, missä valon fysikaalinen käyttäytyminen voi olla hyvinkin monimutkaista, ei tietokonegrafiikassa sen täsmällinen mallintaminen useinkaan ole tarkoituksenmukaista, vaan riittävän[16] hyvän approksimaation tuottaminen, mikä varsinkin reaaliaikaisissa sovelluksissa vaatii usein suuriakin yksinkertaistuksia. Yksinkertaistukset ulottuvat niin valon ilmenemismuotojen kuten valon tuottamiin varjoihin ja valon heijastumisen mallintamiseen, mukaan lukien käytettyihin kaksisuuntaisiin heijastusjakaumafunktioihin, kuin myös itse valon käsitteeseen, jota tyypillisesti käsitellään osissa eikä kokonaisena ilmiönä. Tässä aliluvussa esitellään lyhyesti tietokonegrafiikassa käytetyt yleisimmät approksimaatiot valon suhteen. Varsinaisia tietokonegrafiikassa käytettyjä menetelmiä valon kuvaamiseen käsitellään luvussa 3.

Siinä, missä reaalimaailmassa aineiden kaksisuuntaiset heijastusjakaumafunktiot voivat olla hyvinkin monimutkaisia, käytetään tietokonegrafiikassa usein hyvinkin pelkistettyjä kaksisuuntaisia heijastusjakaumafunktiota jakaen heijastuksen diffuusi- ja spekulaari-termeihin. Myös energian säilymisen noudattaminen tietokonegrafiikassa käytetyissä kaksisuuntaisissa heijastusjakaumafunktiossa ei ole välttämätöntä[17], mutta esimerkiksi luvussa 4 esiteltävässä polunseurantamenetelmässä energian säilyminen tulee huomioida. Yksinkertaisin kaksisuuntainen heijastusjakaumafunktio on Lambertin diffuusi, missä radianssi on kaikkiin suuntiin vakio. Kun aineen heijastuvuutta eli albedoa merkitään $\rho_d$:lla, voidaan Lambertin diffuusi esittää muodossa

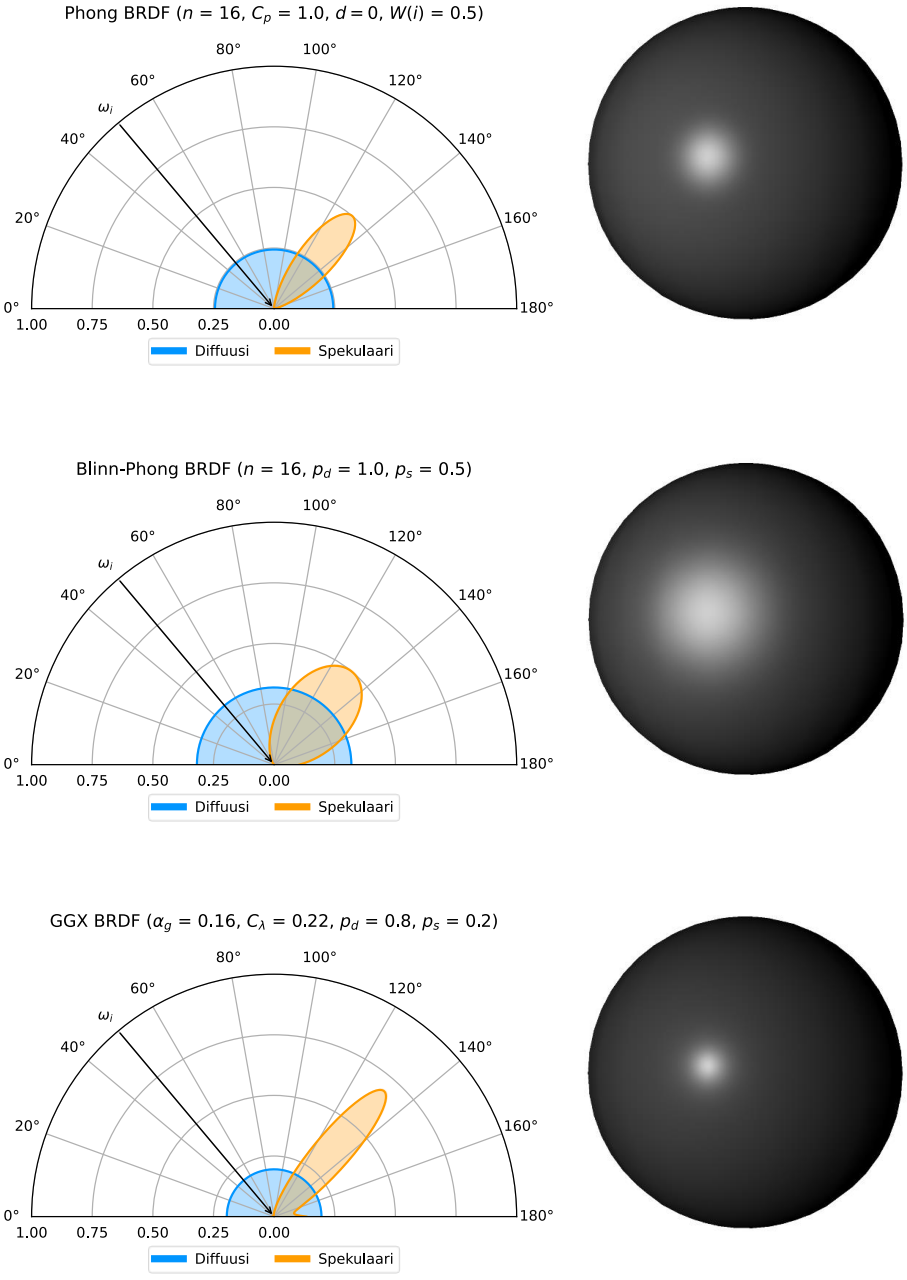
$$f_r(\omega_i, \omega_r) = \frac{\rho_d}{\pi}, \tag{2.21}$$

missä termi $\frac{1}{\pi}$ tarvitaan energian säilymiseksi[18]. Joitakin tietokonegrafiikassa tyypillisiä kaksisuuntaisia heijastusjakaumafunktiota esitetään kuviossa 9. Kuviossa esitettyä Phongin kaksisuuntaista heijastusjakaumafunktiota ei pidä sekoittaa kuviossa 10 esitettyyn pikselikohtaiseen interpolointi menetelmään, joka jossain lähteissä Phongin varjostuksena (engl. *Phong*

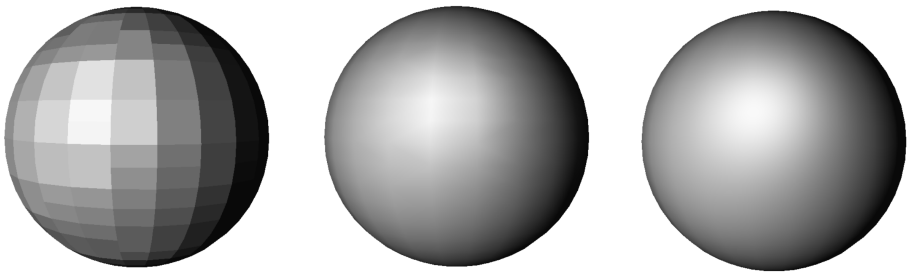
16. Paljon riippuu sovelluskohteesta, sekä mallintamiseen käytettävästä ajasta, mikä on riittävää.

17. Esimerkiksi kuviossa 9 esitetyt Phongin ja Blinn-Phong kaksisuuntaiset heijastusjakaumafunktiot eivät noudata energian säilymistä kuten eivät niiden alkuperäiset kuvauksetkaan.

18. Energian säilymiseksi radianssi tulee jakaa enimmäisprojektioavaruuskulmalla.




Kuvio 9. Tunnettuja tietokonegrafiikassa käytettyjä isotrooppisia kaksisuuntaisia heijastusjakaumafunktioita kuvattuna 2D tasolla: Phong (Phong 1975), Blinn-Phong (Blinn 1977) ja GGX (Walter ym. 2007). Yksittäisen valonsäteen tulokulma on 50°. GGX kaksisuuntaisessa heijastusjakaumafunktiossa on käytetty Schlickin approksimaatiota (Schlick 1994) Fresnel-termille.



Kuvio 10. Valaistuksen interpolointi: per primitiivi, per verteksi ja per pikseli.

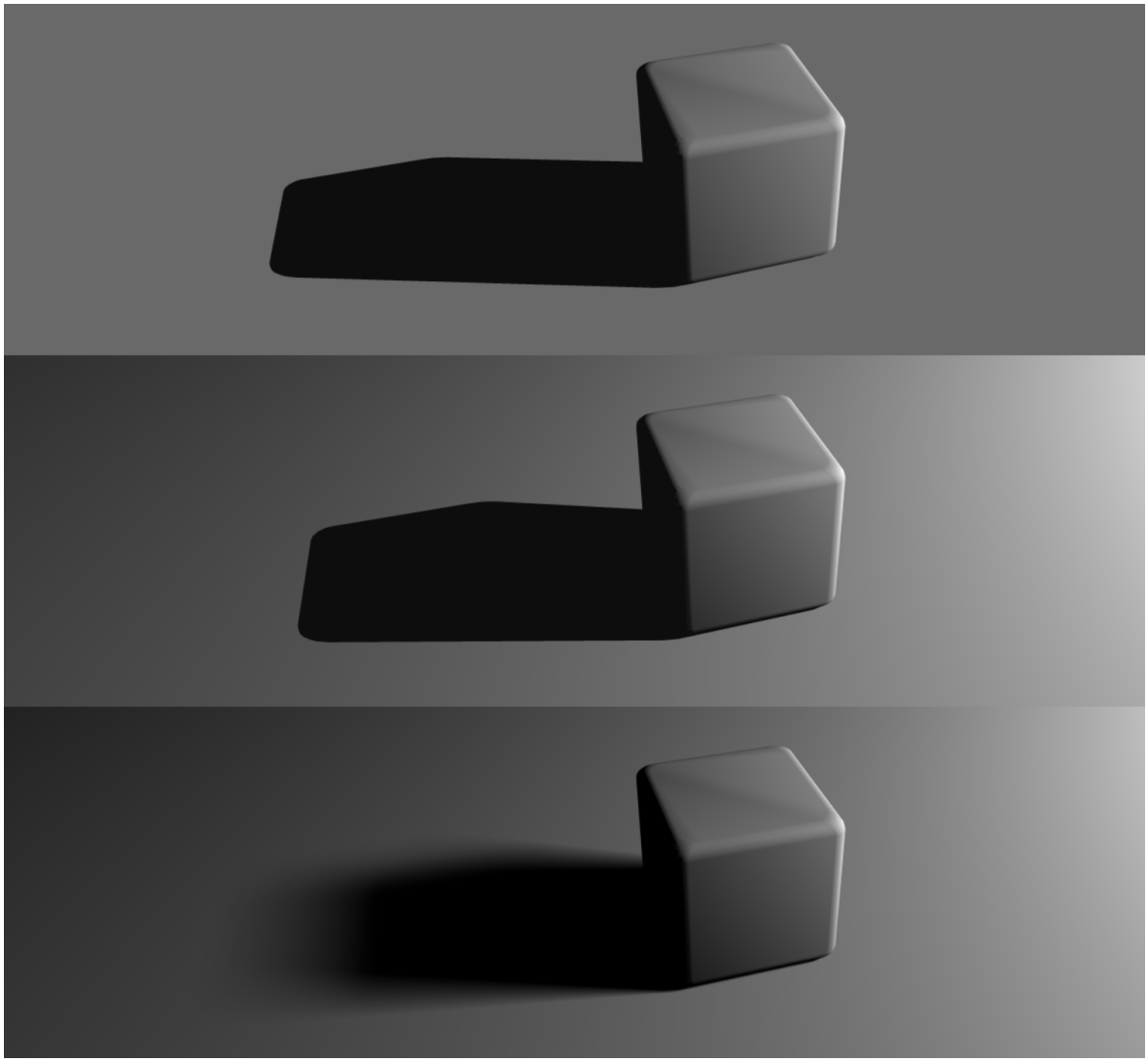
*shading*) tunnetaan. Luvun 7 aineiston videopeleissä paljon käytetty kaksisuuntainen heijastusjakaumafunktio on GGX. Kattavasti tietokonegrafiikassa käytettyjä kaksisuuntaisia heijastusjakaumafunktioita on esitelty ja kategorisoitu artikkelissa (Soldado ja Almagro 2012).

Myös siinä, missä reaalimaailman valonlähteillä on aina jokin pinta-ala, voidaan tietokonegrafiikassa pinta-alallisen valonlähteen (engl. *area light*, *polygonal light*) lisäksi käyttää laskennallisesti yksinkertaisempia suuntavaloja (engl. *directional light*) ja pistevaloja (engl. *point light*)[19]. Toisaalta pinta-ala-alkion ja valonlähteen välisen etäisyyden ollessa hyvin suuri suhteessa valonlähteen pinta-alaan, voidaan pinta-alallista valonlähdettä perustellusti approksimoida pistevalolla, kun taas valonlähteen pinta-alan ollessa hyvin suuri suhteessa pinta-ala-alkion ja valonlähteen välisen etäisyyteen, voidaan valonlähdettä perustellusti approksimoida suuntavalona[20]. Kuviossa 11 esitetään suuntavalon, pistevalon ja pinta-alallisen valonlähteen vaikutus varjostukseen. Pehmeät varjot (engl. *soft shadows*) ovat seurausta laskennallisesti vaativista pinta-alallisista valonlähteistä, mistä syystä pehmeiden varjojenapproksimointiin reaaliaikaisissa sovelluksissa on kehitetty useita menetelmiä. Myös valonlähteen staattisuus saattaa vaikuttaa siihen, miten valonlähdettä käsitellään sovelluksessa ja usein lienee tarvetta erotella staattiset ja dynaamiset valonlähteet toisistaan.

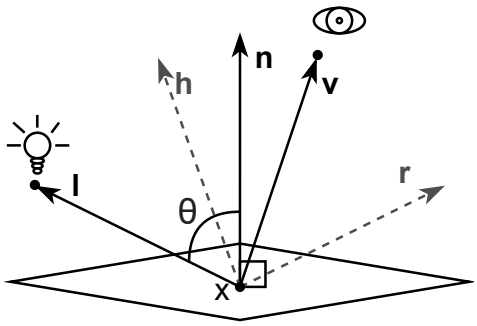
Valon kuvaamisen taso tietokonegrafiikassa jaetaan suoraan (engl. *direct*) ja epäsuoraan (engl.

19. Suunta- ja pistevalot ovat reaaliaikaisissa sovelluksissa laajalti käytettyjä, kun ei-reaaliaikaisessa renderöinnissä puolestaan useimmin käytetään geometrisia ja pallomaisia valonlähteitä (Peters ja Dachsbacher 2019).

20. Suuntavalossa valon suunta ei ole riippuvainen tarkasteltavasta pisteestä, vaan valo saapuu kuhunkin pisteeseen samasta suunnasta.



Kuvio 11. Erilaisista valonlähteistä aiheutuva varjo: suuntavalo (ylhäällä), pistevalo (keskellä) ja pinta-alallinen valonlähde (alhaalla). Pinta-alallisesta valonlähteestä aiheutuva pehmeä varjo on saavutettu näytteistämällä pinta-alallista valonlähdettä 256 näytteellä pikseliä kohden. Pehmeässä varjossa varjon umbra (täysvarjo) ja penumbra (puolivarjo) ovat selvästi havaittavissa.

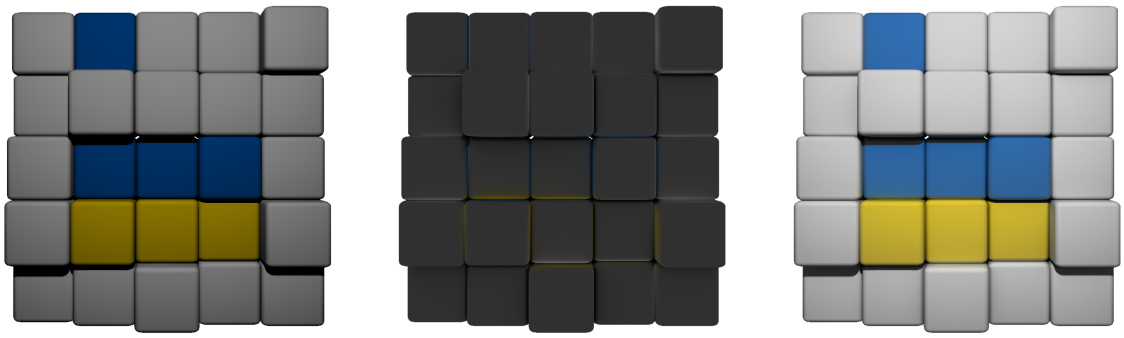



Kuvio 12. Aliluvussa käytetty vektori notaatio, missä **l** on mikä tahansa pinta-ala-alkion $x$ ja sen normaalin **n** suuntaisen puoliavaruuden sisältämä suuntavektori, **v** on pinta-ala-alkion $x$ ja havainnoitsijan välinen suuntavektori, **r** on heijastuslain mukainen suuntavektori $l$:lle ja **h** on vektoreiden **l** ja **v** välinen niin kutsuttu puolivektori (engl. *halfway vector*).

*indirect*) valoon. Suoralla valolla tarkoitetaan suoraan valonlähteiltä pinta-ala-alkiolle saapuvaa valoa, kun epäsuoralla valolla puolestaan tarkoitetaan pinta-ala-alkiolle, muilta kuin valoa emittoivilta pinta-ala-alkiolta, saapuvaa valoa eli heijastunutta valoa. Kun pinta-ala-alkiolle lasketaan suoran valaistuksen vaikutus, ratkaistaan tällöin paikallinen valaistus (engl. *local illumination*). Kun suoran valaistuksen vaikutuksen lisäksi pinta-ala-alkiolle lasketaan epäsuoran valaistuksen vaikutus, ratkaistaan tällöin globaali valaistus (engl. *global illumination*). Paikallisen ja globaalin valaistuksen ero esitetään kuviossa 13. Koska valoa heijastavia eli epäsuoria valonlähteitä on yleisesti valtava määrä verrattuna valoa emittoiviin valonlähteisiin, on globaalin valaistuksen tarkka approksimointi merkittävä haaste reaaliaikaisessa tietokonegrafiikassa.

Kajiya (1986) esitteli niin kutsutun renderöintiyhtälön, joka on yksi tietokonegrafiikan merkittävimmistä yhtälöistä. Renderöintiyhtälön ratkaisemista[21] voidaan pitää realistisen tietokonegrafiikan tavoitteena. Renderöintiyhtälö sisältää edellisessä luvussa esitellyn yhtälön 2.16, mikä tunnetaan myös nimellä heijastavuus yhtälö (engl. *reflectance equation*). Tyypillisesti tietokonegrafiikassa ei suoraan käsitellä avaruuskulmia, vaan vektoreita, jolloin infitesimaalinen avaruuskulma korvataan suuntavektorilla. Tällöin kuvion 12 notaatiota käyttäen heijastavuus yhtälö infitesimaalisille avaruuskulmille voidaan esittää muodossa

21. Analyyttisesti renderöintiyhtälöä ei käytännön tilanteissa kyetä ratkaisemaan, vaan sille pyritään tuottamaan riittävän tarkka approksimaatio.



Kuvio 13. Suora valaistus (vasemmalla), epäsuora valaistus (keskellä) ja näiden kahden yhdistelmä (oikealla). Epäsuoran valaistuksen kuvaamista varten säteenseurannalla on kerätty puoliavaruudesta 1024 näytettä pikseliä kohden.

$$L_r(\mathbf{v}) = \int_{2\pi} f_r(x, -\mathbf{l}, \mathbf{v}) L(\mathbf{l}) \mathbf{n} \cdot \mathbf{l} d\mathbf{l}, \tag{2.22}$$

jolloin valon tulokulma $\omega_i$ korvautuu valaistavan pisteen puoliavaruuteen osoittamalla suuntavektorilla $\mathbf{l}$ ja heijastuskulma $\omega_r$ korvautuu valaistavasta pisteestä katsottuna havainnoitsijan suuntavektorilla $\mathbf{v}$. Lisäksi pinta-ala-alkion normaalin ja valon suuntavektorin välinen kulman kosini $cos(\theta)$ voidaan esittää näiden kahden vektorin pistetulona. Merkitään lisäksi pinta-ala-alkion sijaintia eksplisiittisesti muuttujalla $x$. Kun heijastavuus yhtälöön lisätään pinta-ala-alkion emittoima valo $L_e$ saadaan muodostettua renderöintiyhtälö

$$L(\mathbf{v}) = L_e(\mathbf{v}) + \int_{2\pi} f_r(x, -\mathbf{l}, \mathbf{v}) L(\mathbf{l}) \mathbf{n} \cdot \mathbf{l} d\mathbf{l}. \tag{2.23}$$

Näin ollen pinta-ala-alkion $x$ havainnoitsijan suuntaan $\mathbf{v}$ tuottama radianssi on riippuvainen sekä pinta-ala-alkion emittoimasta valosta, että pinta-ala-alkion heijastamasta valosta. Tärkeä huomio renderöintiyhtälöstä, kuin myös heijastavuus yhtälöstä, on yhtälöiden rekursiivisuus: pinta-ala-alkion tuottama radianssi havainnoitsijan suuntaan on riippuvainen pinta-ala alkiolle saapuvasta radianssista eli irradianssista, mikä puolestaan on riippuvainen muiden pinta-ala alkioiden tuottamasta radianssista kohti ratkaistavaa pinta-ala-alkiota. Rekursiivisuudesta seuraa se, että renderöintiyhtälöä ei kyetä analyyttisesti ratkaisemaan käytännön tilanteissa, mistä syystä yhtälölle pyritään tuottamaan riittävän tarkka approksimaatio. Tun-

nettuja menetelmiä renderöintiyhtälön approksimointiin ovat radiosititeetti ja polunseuranta, joista jälkimmäinen esitellään tarkemmin luvussa 4.

Siinä, missä renderöintiyhtälö esitetään emittoidun ja heijastetun valon summana, voidaan heijastavuus yhtälö esittää suoran ja epäsuoran valaistuksen summana: $L_r = L_{suora} + L_{epasuora}$ (Dutre ym. 2006, 44). Tyypillisesti tietokonegrafiikassa virtuaalimaailman valonlähteet tunnetaan, jolloin suora valaistus voidaan esittää muodossa

$$L_{suora}(\mathbf{v}) = \int_A f_r(x, -\mathbf{l_y}, \mathbf{v}) L_e(\mathbf{l_y}) V(x,y) G(x,y) dA_y, \tag{2.24}$$

missä $A$ käsittää kaikki virtuaalimaailman valonlähteiden pinta-ala-alkiot[22], $y \in A$ ja $\mathbf{l_y}$ on suuntavektori $\overrightarrow{xy}$. Termi $V(x,y)$ on niin kutsuttu näkyvyystermi, joka saa arvokseen yksi, jos pinta-ala-alkiot $x$ ja $y$ näkevät toisensa, muuten nolla[23]. Termi $G(x,y)$ puolestaan on niin kutsuttu geometriatermi, joka riippuu siitä, missä kulmassa valonlähteen pinta-ala-alkio ja ratkaistava pinta-ala-alkio näkevät toisensa eli näiden kahden pinta-ala-alkion välisestä projektioavaruuskulmasta. Geometriatermi voidaan esittää, kun merkitään valonlähteen pinta-ala-alkion normaalia $\mathbf{n_y}$:llä ja pinta-ala-alkion ja valonlähteen pinta-ala-alkion välistä etäisyyttä $\|x-y\|$:llä, muodossa

$$G(x,y) = \frac{(\mathbf{n} \cdot \mathbf{l_y})(\mathbf{n_y} \cdot -\mathbf{l_y})}{\|x-y\|^2}. \tag{2.25}$$

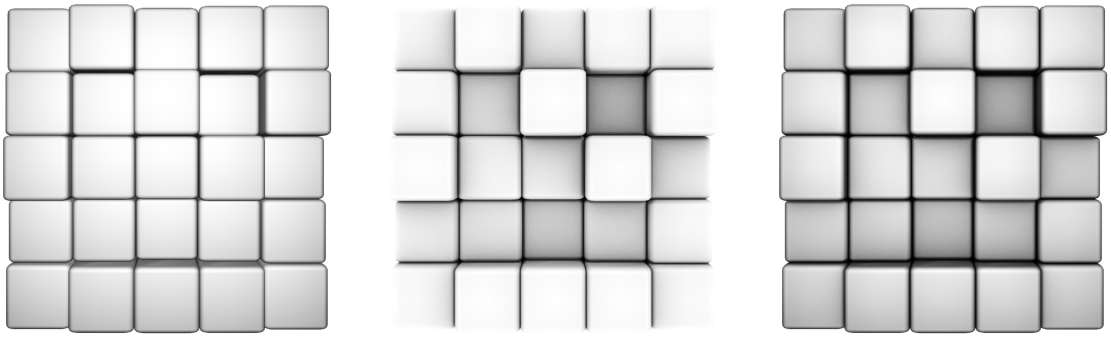
Epäsuora valaistus puolestaan voidaan esittää yhtälönä

$$L_{epasuora}(\mathbf{v}) = \int_{2\pi} f_r(x, -\mathbf{l}, \mathbf{v}) L_i(\mathbf{l}) \mathbf{n} \cdot \mathbf{l} d\mathbf{l}, \tag{2.26}$$

missä $L_i$ merkitsee pinta-ala-alkiolle saapuvaa heijastettua radianssia suunnasta $\mathbf{l}$, eikä siten sisällä kyseisestä suunnasta emittoitua valoa.

22. Virtuaalimaailman koostuessa pelkästään pistemäisistä valonlähteistä, koostuisi $A$ kaikista virtuaalimaailman pistevalojen sijainneista.

23. Toisin sanoen jos pinta-ala-alkiolla ei ole suoraa näkyvyyttä valonlähteen pinta-ala-alkiolle, ei valonlähteeltä saavu pinta-ala-alkiolle suoraa valoa.



Kuvio 14. Suora valaistus (vasemmalla), ympäristön okkluusio (keskellä) ja näiden kahden yhdistelmä (oikealla). Ympäristön okkluusio on tuotettu säteenseurannalla käyttäen 256 sädettä pikseliä kohden.

Globaalin valaistuksen täsmällinen approksimointi epäsuoran valaistuksen[24] myötä on reaaliaikaisessa renderöinnissä on usein haasteellista, mistä syystä epäsuoraa valoa tyypillisesti käsitellään erikseen spekulaari ja diffuusi heijastuksina, jolloin näiden kahden mallintamiseen voidaan käyttää eri menetelmiä tai esimerkiksi voidaan diffuusi heijastukset jättää kokonaan mallintamatta. Laajalti reaaliaikaisessa renderöinnissä käytetty menetelmä globaalin valaistuksen karkeaan approksimaatioon on ympäristön okkluusio (engl. *ambient occlusion*, *AO*) menetelmä. Ympäristön okkluusio menetelmän tarkoituksena on karkeasti arvioida pinta-ala-alkiolle sen normaalin suuntaisesta puoliavaruudesta saapuvaa radianssia. Menetelmässä tämä tapahtuu tarkastelemalla lyhyellä etäisyydellä pinta-ala-alkion näkyvyyttä sen puoliavaruudesta, jolloin näkyvyyttä käytetään arvioimaan pinta-ala-alkiolle saapuvaa radianssia. Säteenseurannalla toteutettu ympäristön okkluusio (engl. *ray-traced ambient occlusion*, *RTAO*) esitetään kuviossa 14. Säteenseurannan ohella ympäristön okkluusio voidaan kuvata myös seuraavassa luvussa esitetyllä rasteroinnilla.

24. Suoran valon ratkaisemiseksi tulee laskea kaikilta pinta-ala-alkiolle näkyviltä virtuaalimaailman valonlähteiltä saapuva radianssi, kun puolestaan epäsuoran valaistuksen laskemiseksi tulee laskea pinta-ala-alkiolle kaikkialta sen puoliavaruudesta saapuva radianssi.

# 3 Renderöintitekniikat

Valon kuvaamiseen tietokonegrafiikassa on vuosien saatossa esitelty muutamia eri tekniikoita[1], joista reaaliaikaisessa renderöinnissä pääsääntöisesti käytetyksi tekniikaksi on muodostunut rasterointitekniikka. Rasterointitekniikan käyttöä reaaliaikaisessa renderöinnissä puoltaa sen sopivuus näytönohjaimella suoritettavaan rinnakkaislaskentaan. Toisaalta rasterointitekniikan huonona puolena voidaan pitää sen heikkoa soveltuvuutta eri valon ilmenemismuotojen kuvaamiseen, mistä syystä reaaliaikaisessa renderöinnissä on syntynyt kiinnostus säteenseurantatekniikkaa kohtaan, joka luonnollisesti soveltuu monien valon ilmenemismuotojen kuvaamiseen. Toisaalta säteenseurantatekniikka on rasterointitekniikkaa huomattavasti laskennallisesti kompleksisempi, mistä syystä säteenseurantatekniikka ei ole rasterointitekniikkaa korvannut kokonaisuudessaan reaaliaikaisessa renderöinnissä, vaan usein näitä kahta tekniikkaa sovelletaan yhdessä, jolloin puhutaan hybriditekniikasta. Tällöin säteenseurantaa käytetään valittujen valon ilmenemismuotojen kuvaamiseen, kun rasterointitekniikkaa puolestaan sovelletaan näkyvien primitiivien määrittämiseen ja muiden kuin säteenseurannalla toteutettujen valon ilmenemismuotojen kuvaamiseen.

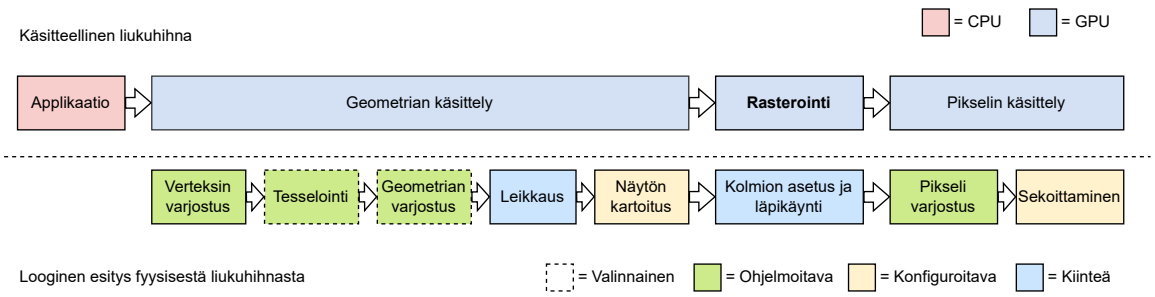
Tämän luvun tarkoituksena on esitellä nämä kaksi renderöintitekniikkaa, sekä tekniikoiden väliset eroavaisuudet. Aliluvussa 3.1 esitellään rasterointitekniikka, sekä tyypillisiä luvun 7 aineiston videopeleissä käytettyjä rasterointi pohjaisia menetelmiä ja tyypillisiä rasteroinnissa käytettyjä laskentaa tehostavia menetelmiä. Aliluvussa 3.2 puolestaan esitellään säteenseurantatekniikan perusteet, sekä tekniikan historia, joka on johtanut luvussa 4 esiteltävän polunseurantamenetelmän kehittymiseen.

## 3.1 Rasterointi

Tietokonegrafiikan yhteydessä rasteroinnilla tarkoitetaan käsitteellisen grafiikkaliukuhihnan vaihetta, joka sijoittuu liukuhihnassa geometrian käsittely ja pikselin käsittely vaiheiden välille. Käsitteellinen, sekä looginen esitys grafiikkaliukuhihnasta on annettu kuviossa 15.

1. Rasterointi ja säteenseurantatekniikoiden lisäksi mainittakoon ei-reaaliaikaisessa renderöinnissä käytetty Reyes renderöinti arkkitehtuuri (Cook, Carpenter ja Catmull 1987).

Loogisella esityksellä tarkoitetaan renderöinti-ohjelmointirajapinnan näkymää grafiikkaliukuhihnaan, mikä voi vaihdella ohjelmistorajapintojen välillä. Rasterointivaiheen tehtävänä on selvittää, missä pikseleissä annettu kolmio on näkyvissä, sekä tuottaa pikseleitä edustavat fragmentit[2] (engl. *fragment*), joille valaistuslaskenta suoritetaan. Tässä tutkielmassa termillä rasterointi viitataan juurikin käsitteellisen liukuhihnan rasterointivaiheeseen, kun rasterointitekniikalla viitataan rasterointiin pohjautuvaan renderöintitekniikkaan ja rasterointipohjaisilla menetelmillä puolestaan viitataan grafiikkaliukuhihnaa hyödyntäviin menetelmiin. Näin ollen tutkielmassa rasterointipohjaisiin menetelmiin luetellaan myös säteenseurantaa hyödyntävät menetelmät, joissa säteenseuranta suoritetaan rasteroinnilla tuotettujen kuvien välillä tai säteenseurantaa käytetään osana ohjelman ajon ulkopuolella tapahtuvaa esilaskentaa. Varsinaisena säteenseurantana tutkielmassa pidetään seuraavassa luvussa esiteltävää säteenseurantaa, jossa säde matkaa virtuaalimaailman sisällä, eikä säteen näkymä siten ole sidottu virtuaalikameran näkymään, kuten on esimerkiksi tässä luvussa esiteltävissä näkymäavaruuden (engl. *screen space*, *SS*) menetelmissä.




Kuvio 15. Grafiikkaliukuhihna kuvattuna käsitteellisenä ja loogisena.

Rasterointipohjaisia menetelmiä on pitkään käytetty reaaliaikaisissa sovelluksissa, mikä on johtanut niin laitteistotason ratkaisujen[3], kuin myös itse algoritmien kehittymiseen. Rasterointitekniikan hyödyntämistä reaaliaikaisessa renderöinnissä puoltaa sen hyvä soveltuvuus rinnakkaislaskentaan, sillä kussakin grafiikkaliukuhihnan vaiheessa primitiivit voidaan käsitellä itsenäisesti. Rasterointiin suunnatut menetelmät voidaan jakaa laskentaa tehostaviin

2. Tutkielmassa käytetään OpenGL ja Vulkan renderöinti-ohjelmointirajapintojen käyttämää termiä fragmentti tarkoittaen pikseliä edustavaa näytettä, joita voi olla yksi tai useampi pikseliä kohden.

3. Esimerkiksi NVIDIA:n näytönohjaimet GeForce3 näytönohjaimesta lähtien tukevat varhaista fragmentin syvyysarvoon perustuvaa hylkäystä (engl. *early Z-rejection*), mikä mahdollistaa muiden primitiivien peittoon jäävien fragmenttien käsittelyn keskeyttämisen (Fernando 2004).

ja valon ilmenemismuotoja mallintaviin menetelmiin. Toisin kuin seuraavassa luvussa esiteltävässä säteenseurannassa, rasteroinnissa ainoastaan näyttöalueelle sijoittuvat primitiivit voivat vaikuttaa lopulliseen kuvaan, jolloin näyttöalueen ulkopuolelle jääviä geometrioita voidaan karsia CPU:lla kameran kuvapyramidin (engl. *frustrum*) ja geometrioille muodostettujen rajaavien tilavuuksien (engl. *bounding volume*) leikkaavuusmäärittelyllä (engl. *CPU culling*). Tämä vähentää näytönohjaimella käsiteltävien primitiivien lukumäärää liukuhihnan leikkausvaiheeseen (engl. *clipping*) asti, kuin myös itse leikkausvaiheessa, jossa suoritetaan laitteistossa toteutettu kuvapuskurin ulkopuolelle jäävien primitiivin osien karsiminen.

Toinen ja laajasti käytetty menetelmä laskennan tehostamiseksi on niin kutsuttu viivästetty varjostus (engl. *deferred shading*), jonka vastakohtana voidaan pitää suoraa varjostusta (engl. *forward shading*). Siinä, missä suorassa varjostuksessa valaistuslaskenta suoritetaan samalla liukuhihnan läpikäynnillä kuin primitiivien näkyvyyksien määrittely, niin viivästetyssä varjostuksessa näkyvyysmäärittely ja valaistuslaskenta suoritetaan erikseen. Tällöin ensimmäisellä liukuhihnan läpikäynnillä, eli primitiivien näkyvyyksien määrittelyssä, todelliset geometriat matkaavat liukuhihnan lävitse samoin kuin suorassa varjostuksessa, mutta sen sijaan, että pikseleille suoritettaisiin varsinainen valaistuslaskenta, muodostetaan niin kutsuttu geometria- eli G-puskuri (engl. *G-buffer*)[4]. G-puskuriin sisällytettäviin näkymän kattaviin tekstuureihin eli kuvapuskureihin tallennetaan tarvittava data valaistuksen, kuin myös muiden visuaalisten tehosteiden tuottamista varten jokaista pikseliä kohden. Hyvinkin pelkistetty[5] G-puskuri esitetään kuviossa 32. Toisella liukuhihnan läpikäynnillä varsinaisten geometrioiden sijasta, liukuhihnan lävitse matkaa kehyspuskurin kattava suorakulmio, jolle suoritetaan valaistuslaskenta G-puskuriin tallennetun datan ja esimerkiksi virtuaalimaailman sisältämien valonlähteiden avulla.

Laskentaa viivästetty varjostus tehostaa takaamalla[6] sen, että jokaiselle pikselille valaistus-

4. Ensimmäisen kerran G-puskurin käyttö kuvailtiin artikkelissa (Saito ja Takahashi 1990).

5. G-puskurin sisältö riippuu aina käyttökohteesta. Monimutkaisempi esimerkki G-puskurista on artikkelissa (Oberberger, Chajdas ja Westermann 2022), jossa G-puskuriin sisällytetään kuvapuskurit pikseleiden syvyysarvoille, kehysten välisille siirtymille eli liikevektoreille (engl. *motion vector*), normaaleille kappaleen omassa ja globaalissa koordinaatistossa, sekä virtuaalimaailman geometrioiden ID:lle.

6. Myös varhaista fragmenttien syvyysarvoon perustuvaa hylkäystä voidaan käyttää samaan tarkoitukseen, jolloin jokaisen fragmentin syvyysarvoa verrataan syvyyspuskuriin (engl. *depth buffer*, *Z-buffer*) *jo* tallennettuun arvoon (Fernando 2004), minkä perusteella fragmentti joko hylätään tai sen käsittelyä jatketaan. Käsittelyn

laskenta suoritetaan vain kerran. Viivästetyn varjostuksen huonot puolet sen sijaan ovat läpinäkyvien materiaalien mallintamisen vaikeutuminen ja se, ettei laitteistossa toteutettua moninäytteistykseen perustuvaa reunanpehmennystä (engl. *multisampling anti-aliasing*, *MSAA*) kyetä järkevästi hyödyntämään (Yang, Liu ja Salvi 2020). Yksinkertainen ja toimiva ratkaisu läpinäkyvien materiaalien mallintamiseen on piirtää läpinäkyvät materiaalit erillisellä liukuhihnan läpikäynnillä. Kykenemättömyys järkevästi hyödyntämään laitteistoon toteutettua moninäytteistykseen perustuvaa reunanpehmennystä on johtanut useiden reunanpehmennysmenetelmien kehittymiseen (Yang, Liu ja Salvi 2020), joista varsinkin videopeleissä laajasti (Patney ym. 2016) käytetty menetelmä on ajallinen reunanpehmennys (engl. *temporal anti-aliasing*, *TAA*).

Siinä, missä moninäytteistykseen perustuvassa reunanpehmennyksessä yhdistetään useampia yksittäisen pikselin sisältä kerättyjä näytteitä eli fragmentteja samalta kehykseltä (engl. *frame*), yhdistetään ajallisessa reunanpehmennysmenetelmässä yksittäisen pikselin sisältä eri kehyksiltä kerättyjä näytteitä. Samoin kuin moninäytteistykseen perustuvassa reunanpehmennyksessä, tulee ajallisessa reunanpehmennyksessä näytteitä poikkeuttaa (engl. *jitter*)[7] pikselin sisällä. Kun virtuaalinäkymä on täysin staattinen eli sekä virtuaalikamera, että virtuaalimaailman geometriat eivät liiku kehyksien välillä, saadaan usean kehyksen jälkeen vastaavat näytteet kuin moninäytteistykseen perustuvassa reunanpehmennyksessä. Näytteiden yhdistämiseen ajallisessa reunanpehmennyksessä voidaan yksittäisille näytteille käyttää erilaisia painokertoimia, kuten esimerkiksi painottaen uudempia näytteitä voimakkaammin. Dynaamisessa näkymässä, jossa virtuaalikamera tai virtuaalimaailman geometriat liikkuvat kehysten välillä, tulee näytteiden yhdistämiseksi dynaamiset pikselit uudelleen projisoida edelliselle kehykselle laskemalla pikseleille kehysten väliset siirtymät[8], joita voidaan nykyisten pikselien sijaintien avulla käyttää aikaisempien näytteiden lukemiseen. Aina dynaamisia pikseleitä ei voida uudelleen projisoida kuten käy esimerkiksi virtuaalikameran kiertyessä kehysten välillä, jolloin näkymän reunoilla sijaitsevia pikseleitä ei kyetä uudelleen projisoimaan, jolloin niille ei kyetä myöskään reunanpehmennystä suorittamaan. Toinen merkittä-

jatkaminen ei takaa sitä, että fragmentti tulee näkymään lopullisessa kuvassa.

7. Yksi tapa poikkeuttamiseen on käyttää Halton sekvenssiä.

8. Kehyksen välinen siirtymä muodostetaan käyttämällä edellisen kehyksen näkymä, projektio ja muunnosmatriiseja projisoimaan nykyinen pikseli edellisen kehyksen näyttökoordinaatistoon, jolloin siirtymä voidaan laskea näyttökoordinaateissa erotuksena nykyisestä kehyksestä.

vä ongelma ajallisessa reunanpehmennyksessä ovat niin kutsutut haamukuvat (engl. *ghosting*), jotka syntyvät kun pikselin sisältö vaihtuu äkillisesti ilman, että aikaisempia virheellisiä näytteitä hylätään, jolloin äkkinäiset muutokset pikselissä tapahtuvat vaiheittain. Kattavasti ajallinen reunanpehmennys ja siihen kohdistettuja parannuksia on esitelty artikkelissa (Yang, Liu ja Salvi 2020).

Ajateltuna valoa aaltoina tai aaltorintamaa vasten kohtisuoraa etenevinä säteinä, voidaan rasterointipohjaisia menetelmiä kutsua "teeskenteleviksi" verrattuna esimerkiksi seuraavassa luvussa esiteltävään säteenseurantatekniikkaan, eikä rasterointitekniikalla sellaisenaan kyetä esimerkiksi renderöintiyhtälöä mielekkäästi approksimoimaan[9]. Tämä onkin johtanut melko laajaan kirjoon valon eri ilmenemismuotoja kuvaavia menetelmiä. Valon luomien varjojen mallintamiseen varjokartat (engl. *shadow maps*)[10] ja sen muunnelmat ovat yleisesti käytettyjä menetelmiä reaaliaikaisessa renderöinnissä (Akenine-Möller, Haines ja Hoffman 2018, s. 263), mikä on havaittavissa myös taulukosta 9. Varjokarttojen haasteita ovat pehmeiden varjojen muodostaminen kuten varjon pehmeneminen varjostajan etäisyyden kasvaessa (engl. *contact hardening*), varjokarttojen resoluution sovittaminen virtuaalikameran näkymään, varjoakne (engl. *shadow acne*) ja varjoakneen poistamisesta poikkeaman avulla aiheutuva varjon irtautuminen varjostavasta geometriasta (engl. *Peter Panning*). Sovelluskohteesta riippuen valaistus, mukaan lukien varjot, voidaan staattisille virtuaalimaailman kappaleille ja valonlähteille laskea etukäteen ja tallentaa eli leipoa (engl. *bake*) niin kutsuttuihin valokarttoihin (engl. *light map*).

Globaalin valaistuksen approksimointiin, kuten ympäristön okkluusion, sekä diffuusi ja spekulaari heijastusten kuvaamiseen on vuosien aikana kehitetty lukuisia rasterointipohjaisia menetelmiä, joita kattavasti esitellään ja vertaillaan artikkelissa (Lambru ym. 2021). Erityi-

9. Oleellinen syy tälle on, että rasteroinnissa virtuaalimaailman näkymä on rajattu suoraan virtuaalikameran näkymään, eikä virtuaalimaailmaa voida tarkastella yksittäisen pinta-ala-alkion näkökulmasta, jolloin kahden pinta-ala-alkion välistä näkyvyyttä ei kyetä todellisuudessa ratkaisemaan.

10. Perusajatus varjokartoissa on ensimmäisellä liukuhihnan läpikäynnillä muodostaa varjostavista virtuaalimaailman kappaleista syvyyspuskuri valonlähteen näkymästä, jonka jälkeen toisella liukuhihnan läpikäynnillä kameran näkymän fragmentit projisoidaan valonlähteen näkymään ja suoritetaan etäisyyden vertailu ensimmäisellä läpikäynnillä muodostettua syvyyspuskuria vasten. Mikäli fragmentin projisoitu syvyysarvo valonlähteen näkymässä on suurempi kuin valonlähteen syvyyspuskuriin tallennettu, on kyseinen fragmentti varjossa.

sesti spekulaari heijastusten ja ympäristön okkluusion kuvaamiseen suosittuja menetelmiä reaaliaikaisissa sovelluksissa ovat näkymäavaruuden menetelmät, mikä ilmenee myös taulukosta 9. Näkymäavaruuden menetelmissä virtuaalimaailmaa tarkastellaan virtuaalikameran näkökulmasta eli laskenta suoritetaan nimensä mukaisesti näkymäavaruudessa. Valon ilmenemismuotojen kuvaamiseen tarvittava data saadaan tyypillisesti G-puskurin sisällytetyistä kuvapuskureista ja esimerkiksi edellisen kehyksen kehyspuskurista. Näin ollen näkymäavaruuden menetelmien hyvänä puolena voidaan pitää sitä, että laskennan kompleksisuus on riippuvainen pikseleiden, eikä virtuaalimaailman sisältämien geometrioiden, lukumäärästä. Toisaalta niiden huono puoli johtuu juurikin samasta, eli menetelmissä näkyvyys rajoittuu kuvapuskureihin eli virtuaalikameran näkymään, jolloin virtuaalikameran näkymän ulkopuolelle ja näkymässä näkyvien primitiivien taakse jääviä primitiivejä ei kyetä menetelmissä hyödyntämään.

Tyypillinen rasterointipohjainen menetelmä lyhyen etäisyyden ympäristön okkluusion kuvaamiseen on näkymäavaruuden ympäristön okkluusio (engl. *screen space ambient occlusion*, *SSAO*), jonka Mittring (2007) esitteli ensimmäisen kerran. SSAO-menetelmässä pikselille määritellyn normaalin suuntaisesta puoliavaruudesta[11] kerätään satunnaisista suunnista näytteitä muuntamalla näytteen sijainti näkymäavaruuden koordinaatistoon ja vertaamalla näytteen syvyysarvoa syvyyspuskuriin[12] tallennettuun arvoon. Ympäristön okkluusio voidaan tällöin muodostaa näkymässä muiden primitiivien taakse ja eteen jäävien näytteiden suhteena. Ympäristön okkluusion arvot voidaan tallentaa erilliseen kuvapuskuriin, jolle voidaan suorittaa häiriönpoisto ennen arvojen käyttämistä, mikä on tarpeellista varsinkin alhaisilla näytemäärillä runsaan häiriön vuoksi. Siinä, missä ympäristön okkluusio itsessään on hyvinkin karkea approksimaatio globaalille valaistukselle, on SSAO-menetelmä itsessään karkeahko approksimaatio ympäristön okkluusiolle[13]. SSAO-menetelmä esitetään kuviossa 16

11. Alkuperäisessä kuvauksessa (Mittring 2007) näytteitä kerätään pallon rajaamasta kokonaisavaruudesta, jolloin tasomaisten pintojen keskellä keskimäärin puolet näytteistä jää tarkasteltavan tason taakse, mikä aiheuttaa tasomaisten pintojen keskiosien näyttäytymisen reunoja tummempina.

12. Oletetaan, että käytetään viivästettyä varjostusta ja G-puskuriin on sisällytetty syvyyspuskuri syvyysarvojen vertailua varten, kuin myös pikselien normaalit tallentava kuvapuskuri puoliavaruuden määrittämistä varten.

13. Jälleen rasteroinnilla ei pinta-ala-alkion todellista näkyvyyttä eri suunnista kyetä määrittelemään, vaan syvyyspuskuria vasten suoritettu vertailu antaa approksimaation pikselin näkyvyydestä näytteistettävästä pis-

ylhäällä. Kuviossa punaisella on merkitty neljä näytettä, joiden keskiarvo muodostaa ympäristön okkluusion arvon $x_a$ tarkasteltavalle pikselille $x$.

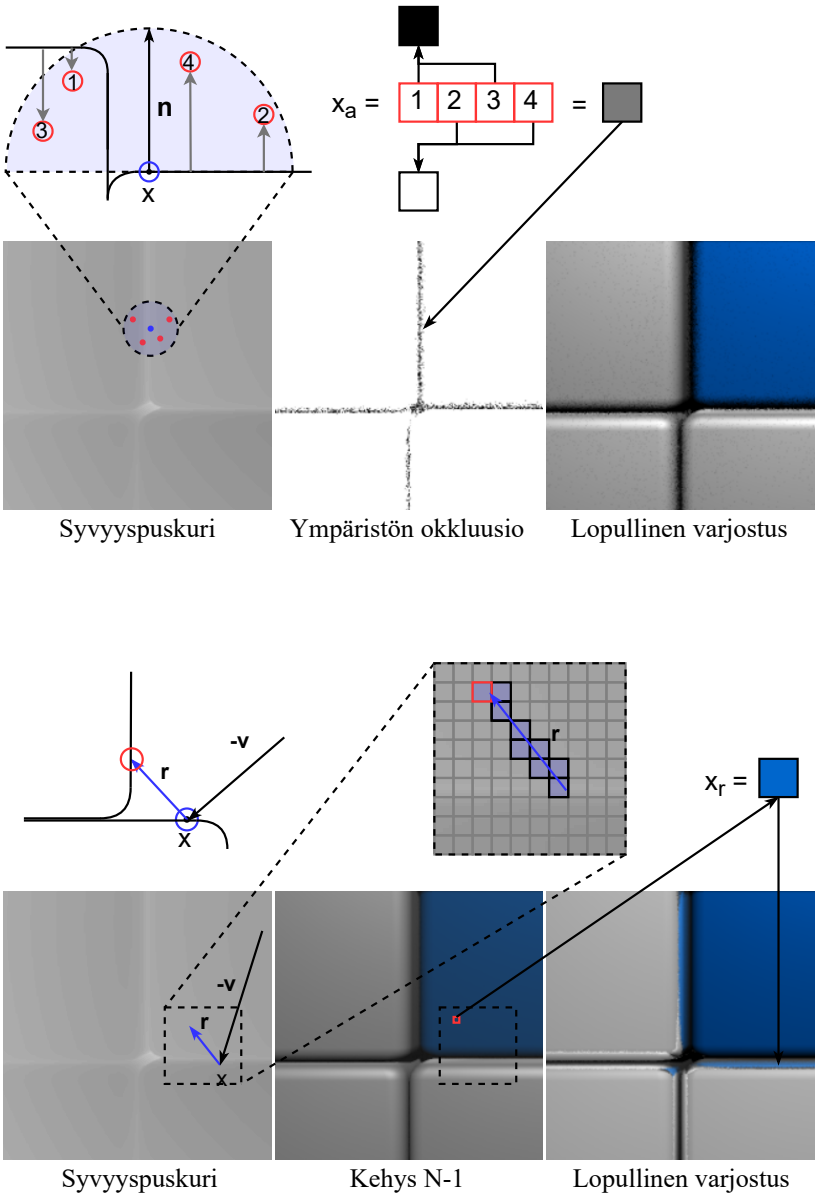
SSAO-menetelmästä on olemassa useampia toteutuksia, joista yksi on luvun 7 aineistossa esiintyvä AMD:n CACAO (Combined Adaptive Compute Ambient Occlusion) toteutus, mikä pohjautuu Intelin kehittämään ASSAO (engl. *adaptive screen space occlusion*) (Engel 2017, 183) toteutukseen ("FidelityFX CACAO - GitHub" 2022). ASSAO toteutus on kokonaisvaltainen ratkaisu ympäristön okkluusiolle sisältäen ympäristön okkluusion muodostamisen lisäksi ympäristön okkluusio arvojen häiriönpoiston. ASSAO toteutus on myös alkuperäistä SSAO toteutusta suorituskykyisempi (Engel 2017, 183) erotellun lomitetun (engl. *deinterleaved*) näytteistämisen (Bavoil 2014) ansiosta. Erotellussa lomitetussa näytteistämisessä kuvapuskurin pikselit jaetaan tasaisesti eri läpivienneille. ASSAO toteutuksessa ympäristön okkluusio muodostetaan neljällä läpiviennillä, joilla jokaisella näytteistetään vain yhdestä pikselistä 2x2 pikselin alueelta käyttäen läpivienti kohtaista näytteistämiskuviota, mikä vähentää syvyyspuskurin muistilukujen latenssia merkittävästi (Bavoil 2014).

Reaaliaikaisessa renderöinnissä käytettyjä yksinkertaisia rasterointipohjaisia menetelmiä spekulaari heijastusten kuvaamiseen ovat muun muassa tasoheijastukset ja ympäristökartat (engl. *environment map*). Tasoheijastuksissa virtuaalimaailma renderöidään kahdesta näkymästä: todellisen virtuaalikameran näkymästä, sekä heijastavan tason alapuolelle sijoittuvan ja tason heijastusvektorin suuntaan osoittavan virtuaalikameran näkymästä, mikä tekee siitä tyypillisesti laskennallisesti raskaan, eikä se nimensä mukaisesti toimi muille kuin tasomaisille geometrioille. Toisaalta menetelmä tuottaa todellisen kuvauksen heijastuksista. Ympäristökartta puolestaan on tietystä pisteestä muodostettu esilaskettu näkymä useampaan suuntaan[14] virtuaalimaailmassa tai geneerisemässä ympäristössä, josta heijastusvektorin avulla saadaan heijastukselle luettua väri. Toisin kuin tasoheijastukset, ympäristökartat soveltuvat käytettäväksi myös kaareville pinnoille, mutta ovat sidottuna tiettyyn pisteeseen virtuaalimaailmassa, eivätkä siten sovellu heijastusten kuvaamiseen dynaamisten geometrioiden pinnoilta tai dynaamisten geometrioiden heijastamiseen.

Varsinkin videopeleissä laajalti käytetty menetelmä spekulaari heijastusten mallintamiseen

teestä.

14. Tyypillisesti ympäristökarttana käytetään kuutiota (engl. *cube map*) tai palloa (engl. *sphere map*).

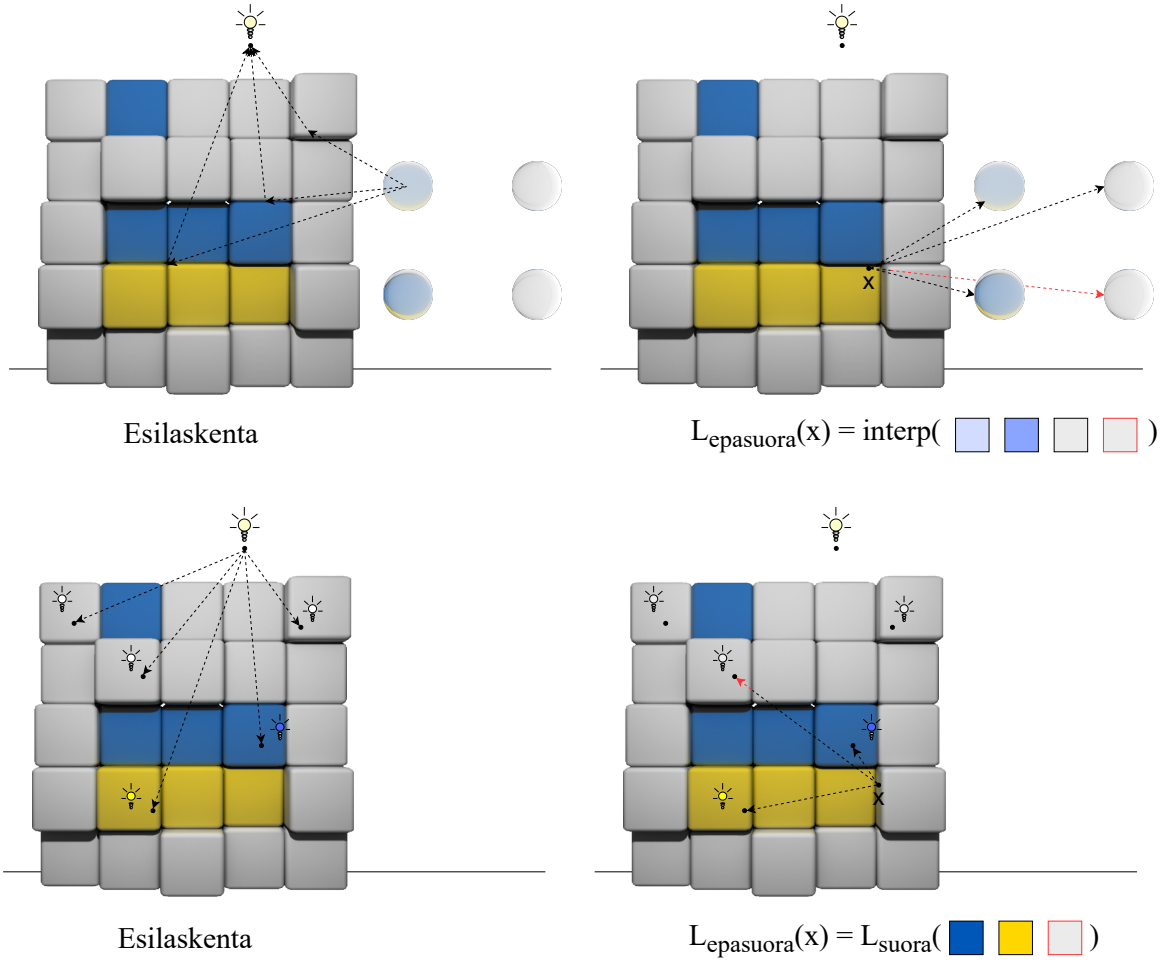



Kuvio 16. Näkymäavaruuden menetelmiä: ympäristön okkluusio pikselille *x* käyttäen neljää näytettä (yläpuolella) ja spekulaari heijastus kehyksellä *N*, pikselille *x* ja heijastusvektorille *r* (alapuolella).

on näkymäavaruuden heijastukset (engl. *screen space reflections*, *SSR*), mikä ilmenee myös taulukosta 9. Menetelmän ensimmäisenä esittelivät Sousa, Kasyan ja Schulz (2012). Näkymäavaruuden heijastuksissa heijastusvektori muodostetaan G-puskuriin sisällytettyjen pikseleiden normaalit ja sijainnit tallentavien kuvapuskureiden, sekä pikselissä näkyvän materiaalin kaksisuuntaisen heijastusjakaumafunktion avulla. Heijastusvektoria pitkin edetään (engl. *ray marching*) näyttökoordinaatistossa suorittaen vertailuja syvyyspuskuriin tallennettuja arvoja vasten. Jos heijastusvektorin ja syvyyspuskurin välillä todetaan leikkaus eli seurattava säde menee syvyyspuskurissa näkyvän primitiivin taakse, uudelleen projisoidaan leikkauspiste edellisen kehyksen kuvapuskuriin, josta heijastukselle saadaan väri. Siinä, missä näkymäavaruuden heijastukset on yleensä tasoheijastuksia kevyempi menetelmä laskennallisesti ja se soveltuu kaareville pinnoille, ei menetelmässä virtuaalikameran näkymän ulkopuolelle jääviä primitiivejä kyetä heijastamaan. Tyypillisesti tämä näkyy esimerkiksi videopeleissä lattiatason heijastusten katkeamisena lähellä virtuaalikameraa, kun heijastusvektori osoittaa jyrkässä kulmassa ylöspäin virtuaalikameran näkymän ulkopuolelle. Näkymäavaruuden heijastukset menetelmä esitetään kuviossa 16 alhaalla. Kuviossa punaisella merkitään leikkauspiste, jonka avulla edellisen kehyksen kehyspuskurista saadaan tarkasteltavan pikselin $x$ heijastama väri $x_r$.

Reaaliaikaisissa tietokonegrafiikan sovelluksissa ja varsinkin videopeleissä diffuusi heijastukset on usein tarpeen laskea etukäteen ja tallentaa ne valokarttoihin sellaisenaan tai esimerkiksi useimpien pelimoottoreiden tukemiin (Morgan McGuire 2019) niin kutsuttuihin irradianssiluotaimiin (engl. *irrandiance probe*, *light probe*)[15]. Irradianssiluotaimilla tarkoitetaan virtuaalimaailmaan *avoimeen tilaan* sijoitettuja pisteitä, joihin tallennetaan ympäristöstä saapuva radianssi eli irradianssi. Luotaimien luonti tapahtuu ohjelman suorituksen ulkopuolella säteenseurantamenetelmää hyödyntäen. Luotaimia voidaan sijoittaa hilarakenteeseen tai epäsäännöllisesti esimerkiksi sijainteihin, joissa irradianssi arvot muuttuvat merkittävästi. Ohjelman suorituksen aikana epäsuoran valaistuksen kuvaaminen tapahtuu interpoloimalla lähimpiä luotaimia. Näin ollen yksi merkittävä tekijä luotaimien sijoittelussa on mahdollistaa luotaimien nopea interpolointi. Toinen merkittävä tekijä luotaimien sijoittelussa on välttää luotaimien sijoittuminen virtuaalimaailman geometrioiden sisälle. Interpoloi-

15. Myös monia muita menetelmiä on, joista yksinkertaisimpana voidaan pitää ympäristöstä saapuvan valon approksimoimista vakiona valaistuslaskennassa, jolloin varjoon jääviä fragmentteja ei kuvata täysin mustina.




Kuvio 17. Rasterointipohjaisia menetelmiä globaalin valaistuksen kuvaamiseen: irradianssiluotaimet (yläpuolella) ja virtuaaliset pistevalot (alapuolella).

tavien luotaimien näytteistämisessä luotaimelta pinta-ala-alkion suuntaan saapuva radianssi määräytyy niin kutsutun irradianssifunktion[16] perusteella. Irradianssiluotaimien käyttö epäsuoran valaistuksen kuvaamiseen esitetään kuviossa 17 ylhäällä. Kuviossa vasemmalla esite-

16. Sen sijaan, että muistiin tallennettaisiin irradianssi kartta, johon sisällytetään kaikille kokoavaruuden kulmille irradianssi, voidaan irradianssifunktiona käyttää niin kutsuttuja palloharmonisia funktioita (engl. *spherical harmonics*), jolloin irradianssi karttaa voidaan approksimoida vain muutamaa vakiota käyttäen. Ensimmäisen kerran palloharmonisten funktioiden käyttö irradianssi karttojen approksimointiin esiteltiin artikkelissa (Ramamoorthi ja Hanrahan 2001). Palloharmonisten funktioiden avulla tapahtuvan irradianssi kartan approksimoinnin perusajatuksena on esittää kartta eri taajuuksien painotettuna summana, missä paino on taajuuskomponentille laskettu vakio. Käytännöllinen johdatus palloharmonisiin funktioihin tietokonegrafiikan yhteydessä esitellään artikkelissa (Sloan 2008).

tään yksittäisen luotaimen esilaskenta käyttäen säteenseurantateknikkaa ja oikealla esitetään neljän lähimmän luotaimen interpolointi pisteen $x$ epäsuoran valaistuksen kuvaamiseksi.

Ensimmäisen kerran irradianssiluotaimet esiteltiin osana artikkelissa (Greger ym. 1998) esiteltyä irradianssitilavuus (engl. *irradiance volume*) menetelmää, jonka jälkeen menetelmästä on muodostunut useampia muunnoksia. Moni alkuperäisen menetelmän muunnoksista pyrkii tarkkuuden ja suorituskyvyn ohella estämään irradianssiluotaimille ominaisen ongelman, valovuodon (engl. *light leak*). Valovuoto aiheutuu kun näytteistetään luotainta johon valaistavasta pisteestä ei ole todellista näkyvyyttä. Kuviossa 17 ylhäällä punaisella on korostettu luotaimen näytteistys, johon valaistavasta pisteestä $x$ ei ole näkyvyyttä. Kuviossa esitetyssä tilanteessa ei kuitenkaan aiheudu merkittävää valovuotoa, sillä ilman näkyvyyttä näytteistetyn luotaimen arvo ei juurikaan poikkea muista näytteistetyistä luotaimista. Todellisempi esimerkki valovuodoista voisi olla tilanne, jossa virtuaalimaailmassa on jokin ohut rakenne kuten seinä, jonka eri puolilla valaistus poikkeaa toisistaan suuresti, jolloin näytteistettäessä seinän toisella puolella olevaa luotainta valo vuotaa seinän lävitse. Yksi valovuotoja vähentävä, sekä diffuusi heijastusten lisäksi spekulaari heijastukset mahdollistava muunnelma irradianssitilavuusmenetelmästä on Remedyn videopeleissä, kuten luvun 7 aineiston Control-videopelissä, käytetty virtuaalimaailman vokselointia hyödyntävä menetelmä (Silvennoinen ja Timonen 2015). Valovuotojen estämiseksi menetelmässä säilytetään listaa vokseleista näkyvistä luotaimista, jolloin vältytään näytteistettämästä niitä luotaimia, jotka eivät tarkasteltavasta vokselista ole näkyvissä. Lisäksi menetelmässä pinta-ala-alkiolle näkyvien luotaimien määrittelyssä käytetään pinta-ala-alkion normaalia[17].

Tutkielman kannalta kiinnostava tapa kuvata diffuusi heijastuksia on niiden kuvaaminen niin kutsuttujen virtuaalisten pistevalojen (engl. *virtual point light*, *VPL*) avulla. Virtuaalisten pistevalojen avulla tapahtuva diffuusi heijastusten kuvaaminen johtaa juurensa artikkelissa (Keller 1997) esiteltyyn välitön radiositeetti (engl. *instant radiosity*) menetelmään. Virtuaalisella pistevalolla tarkoitetaan geometrian *pinnalle* sijoitettavaa ja pinnalta heijastuvaa valoa emittoivaa kuvitteellista valonlähdettä. Virtuaalisten pistevalojen käyttö valaistuslaskennassa

17. Pinta-ala-alkion normaaliin perustuva näkyvyyden approksimointi on yksi tyypillisistä heuristiikoista valovuotojen vähentämiseksi. Pinta-ala-alkion normaaliin perustuvan näkyvyyden approksimoinnin avulla voidaan esimerkiksi estää seinän pinnalle seinän toiselta puolelta aiheutuva valovuoto lukuun ottamatta kulmauksia.

esitetään kuviossa 17 alhaalla. Virtuaalisten pistevalojen avulla diffuusi heijastusten kuvaaminen tapahtuu samoin kuin suoran valaistuksen laskenta eli näkyviä valonlähteitä näytteistämällä. Kuviossa punaisella nuolella esitetään virtuaalisen valonlähteen näytteistäminen, johon pinta-ala-alkiosta ei ole näkyvyyttä. Virtuaalisten pistevalojen muodostaminen tapahtuu seuraavassa luvussa esiteltävää säteenseurantaa hyödyntäen aloittaen säteiden seuraaminen valonlähteeltä ja lisäämällä virtuaalinen pistevalo säteen ja geometrian leikkauspisteeseen.

## 3.2 Säteenseuranta

Verrattuna edellisessä luvussa esitettyihin rasterointipohjaisiin menetelmiin, voidaan säteenseurannan sanoa pohjautuvan suoremmin valon fyysiseen käyttäytymiseen. Siinä, missä sädeoptiikassa säde määritellään valon aaltorintaman normaalin mukaisesti, määritellään säteenseurannassa säde useimmiten[18] päinvastaiseen suuntaan eli havainnoitsijalta kohti valonlähdettä[19]. Tämä vähentää seurattavien säteiden lukumäärää merkittävästi sillä, jos säteitä seurattaisiin valonlähteeltä satunnaisiin suuntiin, niin todennäköisesti suuri osa säteistä ei kulkeutuisi virtuaalikameraan, eivätkä siten vaikuttaisi lopulliseen kuvaan.

Säteenseurannan perusajatuksena on muodostaa kuva mallintamalla valon säteiden kulkeutumista virtuaalimaailmassa. Säteenseurannan säde voidaan rinnakkaistaa luvussa 2.2 esiteltyyn radiometriseen säteeseen, joilla molemmilla on infitesimaalinen pinta-ala, mistä seuraa se, että säde sopii kuvattavaksi vektorina. Yleisesti säde voidaan kuvailla alkupisteen $O$, suuntavektorin $d$ ja etäisyyden $t$ avulla funktiona

$$P(t) = O + t\mathbf{d}, \tag{3.1}$$

missä $t \in [0, +\infty[$. Säteenseurannan yhteydessä säde on usein kuitenkin tarpeellista esittää suljetulla välillä $t \in [t_{min}, t_{max}]$, kuten säde esitetään esimerkiksi luvussa 5.3 esiteltävässä

18. Poikkeuksena on esimerkiksi kaksisuuntainen polunseuranta (engl. *bidirectional path tracing*) (Lafortune ja Willems 1998), jossa säteitä seurataan sekä havainnoitsijalta valonlähteelle, että valonlähteeltä kohti havainnoitsijaa, mikä hyödyttää tilanteissa, joissa valonlähde on vaikeasti osuttavissa säteillä. Kaksisuuntainen polunseuranta on havainnollistettu kuviossa 18 alhaalla.

19. Heijastusjakaumafunktion vastavuoroisuus ominaisuus mahdollistaa tämän.

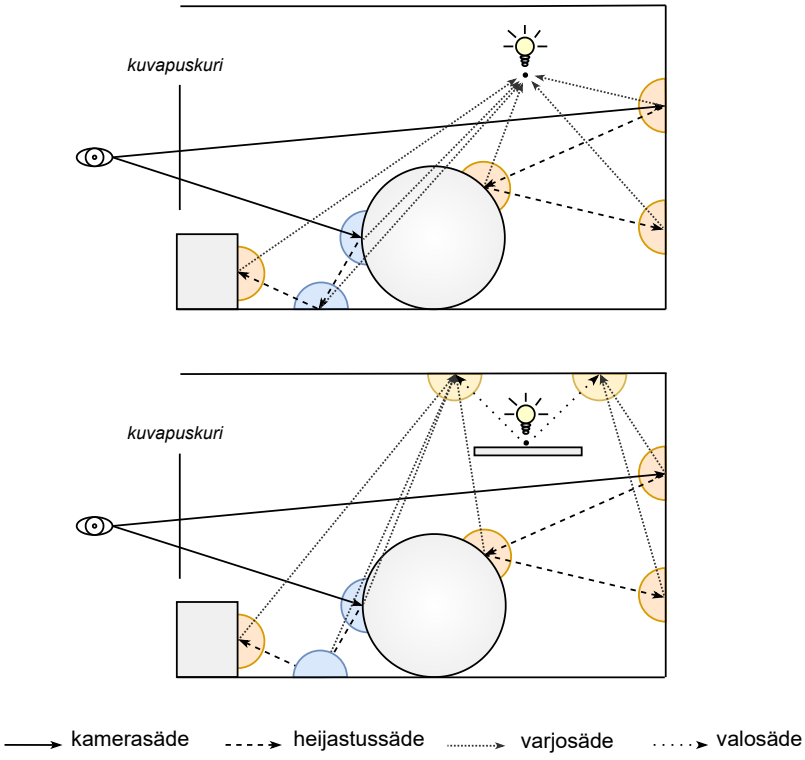
Vulkan API:ssa, kuin myös Microsoftin DirectX API:ssa (“Microsoft DirectX Raytracing Functional Spec” 2022). Tällöin säteen ja virtuaalimaailman geometrioiden välisiä leikkaavuuksia ei tarkastella säteellä lähempänä kuin $t_{min}$ säteen alkupistettä[20], eikä pidemmällä kuin $t_{max}$ säteen alkupisteestä. Toisin sanoen säteen seuraaminen voidan aloittaa pisteestä $P(t_{min})$ ja lopettaa viimeistään pisteessä $P(t_{max})$.

Valon säteiden kulkeutumisen kuvaamiseksi, säteenseurannan keskeisenä tehtävä on ratkaista säteen ja virtuaalimaailman geometrioiden välinen leikkaavuus, jonka tavoitteena usein on löytää *lähin* primitiivi, kuten kolmio, jonka kanssa säde leikkaa määrätyllä välillä[21]. Tällöin leikkauspisteestä voidaan jatkaa valonsäteen kulkeutumisen kuvaamista seuraamalla uusi säde esimerkiksi primitiivin normaalin ja säteen tulokulman määräämän heijastuskulman suuntaan. Jokaiselle seurattavalle säteelle suoritetaan leikkaavuustarkastelu säteen ja virtuaalimaailman geometrioiden välillä. Riippuen sovelluskohteesta, voi virtuaalimaailma koostua jopa useista miljoonista primitiiveistä, jolloin tarkasteltavien primitiivien lukumäärää on usein tarvetta vähentää käyttämällä hierarkkisia rakenteita primitiivien säilömiseen kuten rajaavientilavuuksien hierarkiaa (engl. *bounding volume hierarchy*, *BVH*), jonka lehtisolmuihin primitiivit säilötään. Tällöin, jos säde ei leikkaa hierarkiassa ylemmälle tasolle määrätyn rajaavan tilavuuden kanssa, voidaan hierarkiassa tätä alemmat tasot ja siten kaikki kyseisen haaran lehtisolmuissa sijaitsevat primitiivit jättää tarkastelematta. Rajaavientilavuuksien hierarkia on myös luvussa 5 esiteltävän laitteistokiihdytteisen säteenseurannan keskiössä reaaliaikaisuuden mahdollistamisessa.

Kuten myös rasteroinnilla, voidaan säteenseurannalla suorittaa näkyvyystarkastelu virtuaalikameran näkymässä. Siinä, missä rasteroinnissa päätellään missä pikseleissä tarkasteltava primitiivi näkyy, voidaan säteenseurannalla päätellä mikä primitiivi tarkasteltavassa pikselissä näkyy seuraamalla säteitä havainnoitsijalta kehyspuskurin pikseleiden lävitse. Näin ollen näiden kahden tekniikan oleellinen ero näkyvyystarkastelussa on järjestys, missä asioita tarkastellaan. Pikseleiden lävitse seurattavista säteistä käytetään termiä primääri- eli kamera-

20. Lähimmäisetäisyyden $t_{min}$ tyypillinen käyttökohde on estää säteen leikkaaminen vektorin alkupisteen $O$ kanssa liukulukujen pyöristysvirheistä johtuen.

21. Jos säteenseurantaa käytetään esimerkiksi kahden pinta-ala-alkion välisen näkyvyyden tarkasteluun määrätyllä välillä, riittää tällöin löytää *mikä tahansa* leikkaavuus säteen ja virtuaalimaailman primitiivin välillä näkyvyyden estymisen toteamiseksi.




Kuvio 18. Perinteinen polunseuranta (ylhäällä) ja kaksisuuntainen polunseuranta (alhaalla). Säteen ja geometrian leikkauspisteen normaalin suuntainen puoliavaruus on kuvattu puolipallolla.

säde. Koska näkyvyystarkastelu on säteenseurannalla yleensä rasterointia hitaampaa, ei primäärisäteitä tyypillisesti reaaliaikaisessa renderöinnissä seurata, vaan sekundäärisiä ja mahdollisesti sitä seuraavia säteitä. Tällöin käytettäessä rasterointia ja viivästettyä varjostusta, saadaan sekundäärisäteiden suuntaamiseen tarvittava data G-puskurista[22]. Suorittaessa näkyvyystarkastelu virtuaalikameran näkymässä säteenseurannalla, saadaan sekundäärisäteen suuntaamiseen tarvittava data puolestaan lähimmän leikkauspisteen ja lähimmän leikkaavan primitiivin ominaisuudet tuntemalla.

Se, missä säteenseurantatekniikka merkittävästi poikkeaa rasterointitekniikasta on se, että sekundäärisäteillä ja sitä seuraavilla säteillä kyetään tekemään näkyvyystarkastelu virtuaali-

22. Olettaen, että G-puskuriin on tallennettu vähintäänkin pikseleiden sijainnit sisältävä kuvapuskuri. Heijastusten kuvaamiseksi G-puskuriin tulisi olla sijaintien lisäksi tallennettu normaalit ja esimerkiksi materiaalien ID:t sisältävät kuvapuskurit, joiden avulla heijastussäteelle voidaan muodostaa suunta.

maailmassa mielivaltaisista pisteistä ja mielivaltaisiin suuntiin, eikä näkyvyystarkastelu näin ole sidottu virtuaalikameran näkymään[23]. Näin ollen sekundäärisäde ja sitä seuraavat säteet sopivat erilaisten valon ilmenemismuotojen, kuten varjojen ja heijastusten eli suoran ja epäsuoran valaistuksen kuvaamiseen. Varjosäteeksi kutsutaan pinta-ala-alkiolta kohti valonlähdettä seurattavaa sädettä. Mikäli varjosäde leikkaa läpikuultamattoman primitiivin pinta-ala-alkion ja valonlähteen välillä, on pinta-ala-alkio tarkasteltavalta valonlähteeltä varjossa. Toisin sanoen, varjon kuvaamiseksi tulee suorittaa näkyvyystarkastelu pinta-ala-alkion ja valonlähteen välillä. Heijastusten kuvaamiseksi puolestaan tulee tuntea havainnoitsijan suunta, leikkauspisteen normaali ja pinnan kaksisuuntainen heijastusjakaumafunktio, joiden avulla heijastussäteelle saadaan muodostettua suunta. Heijastussuunnasta saapuvan radianssin määrittelemiseksi tulee heijastussäteen ja primitiivin leikkauspisteen valaistus ratkaista. Kuviossa 18 ylhäällä esitetään säteenseurantaa hyödyntävä polunseurantamenetelmä seuraten kahta eri pikseleiden lävitse kulkeutuvaa polkua alkaen havainnoitsijalta. Kumpikin polku sisältää yhden primäärisäteen lisäksi kaksi heijastussädettä, sekä molemmista kimmokkeista (engl. *bounce*) eli leikkauspisteistä varjosäteen leikkauspisteen suoran valaistuksen ratkaisemiseksi. Kuviossa alhaalla esitetään kaksisuuntainen polunseurantamenetelmä, jossa säteitä seurataan sekä havainnoitsijalta kohti valonlähdettä että valonlähteeltä ympäristöön, mikä auttaa säteiden osumista vaikeasti tavoitettaviin valonlähteisiin.

Kuten johdannossa mainittiin, on säteenseuranta pitkään tunnettu tietokonegrafiikassa Arthur Appelin (1968) ollessa ensimmäinen, joka kuvasi säteenseurannan käytön tietokonegrafiikassa. Hän kuvasi säteenseurannan ei-rekursiivisena suoran valaistuksen muodostamiseksi kappaleelle. Ensimmäisenä nykyaikaisena säteenseurantaa hyödyntävänä menetelmänä pidetään Whittedin (1979) esittämää menetelmää, jossa säteenseurantaa hyödynnetään rekursiivisesti peiliheijastusten, kovien varjojen ja valon ideaalisen taittumisen kuvaamiseen. Seuraavana merkittävänä menetelmänä voidaan pitää artikkelissa (Cook, Porter ja Carpenter 1984) esiteltyä hajautettua säteenseurantaa (engl. *distributed ray tracing*), jossa yksittäisen säteen sijasta puoliavaruutta näytteistetään satunnaisista suunnista useammalla säteellä mah-

23. Rasteroinnilla sama vaatisi virtuaalimaailman tarkkailun toisen virtuaalikameran lävitse kuten tasoheijastuksista, mikä ei enemmissä määrin ole mahdollista reaaliajassa. Toisaalta, koska virtuaalimaailmaa voidaan tarkastella mielivaltaisiin suuntiin, ei kuvapyramidillä kyetä säteenseurannassa käytettäviä geometrioita triviaalisti karsimaan.

dollistaen erilaiset spekulaari heijastukset ja pehmeät varjot. Matemaattisen ja täsmällisen määritelmän hajautetulle säteenseurannalle kuvaili Kajiya (1986) renderöintiyhtälön muodossa. Samassa artikkelissa Kajiya (1986) esittelee polunseurantamenetelmän, joka tuottaa approksimaation renderöintiyhtälöön. Vuosien aikana menetelmä on vakiinnuttanut asemansa säteenseurantaa hyödyntävien menetelmien keskuudessa, mistä syystä tässä tutkielmassa säteenseurantaa tarkastellaankin suureksi osaksi seuraavana esiteltävän polunseurantamenetelmän kautta.

# 4 Polunseuranta

Kuten tutkielmassa aikaisemmin mainittiin, on polunseurantamenetelmä yksi tapa approksimoida renderöintiyhtälöä. Polunseurannassa approksimointi tapahtuu yhdistämällä Monte Carlo -integrointia ja edellisessä luvussa esiteltyä säteenseurantaa. Monte Carlo -integrointi, kuten Monte Carlo -menetelmät yleisemminkin, perustuvat satunnaisuuden hyödyntämiseen determinististen ongelmien ratkaisemisessa. Polunseurantamenetelmässä tämä tarkoittaa satunnaisuuden hyödyntämistä polun muodostavien säteiden suuntaamisessa. Menetelmässä yksi polku muodostaa yhden näytteen valon kulkeutumisesta virtuaalimaailmassa ja useampi näyte muodostaa kokonaisen kuvan[1]. Vaikkakin polunseurantamenetelmän perusajatus on hyvin yksinkertainen, vaativat menetelmän syvempi ymmärtäminen ja soveltaminen osakseen teoriaa, jota tässä luvussa esitellään. Aliluvussa 4.1 esitellään ne todennäköisyysteorian peruskäsitteet, joita tullaan tarvitsemaan puolestaan aliluvussa 4.2 esiteltävän Monte Carlo -integroinnin esittelemiseksi. Nämä aliluvut perustuvat pääosin Forsythin (2018, luvut 3 ja 4), sekä Veachin (1998) väitöskirjaan. Aliluvuissa 4.3 ja 4.4 puolestaan esitellään näytteistämisen ja häiriönpoiston menetelmiä Monte Carlo -estimaattorin varianssin pienentämiseksi.

## 4.1 Todennäköisyysteoria

Todennäköisyysteoria on todennäköisyyksiä tutkiva matematiikan osa-alue, josta muutamaa käsitettä polunseurantamenetelmän ymmärtämiseksi tarvitaan. Tässä aliluvussa lyhyesti esitellään todennäköisyysteorian peruskäsitteistä satunnaismuuttuja, todennäköisyysjakauma, odotusarvo ja varianssi. Todennäköisyysteorian yksi keskeisimpiä käsitteitä on satunnaismuuttuja, joka nimestään huolimatta ei ole muuttuja, vaan funktio, joka liittää satunnaisilmiön tulosvaihtoehdot reaalilukuihin. Tällöin, kun näyteavaruutta[2] (engl. *sample space*) eli perusjoukkoa merkitään $\Omega$:lla, voidaan reaaliarvoinen[3] satunnaismuuttuja matemaattisesti

1. Olettaen, että kerätään vähintään yksi näyte jokaista pikseliä kohden.

2. Näyteavaruus on kaikkien satunnaisilmiön tuloksien joukko, jolloin esimerkiksi nopanheiton näyteavaruus on $\Omega = \{1,2,3,4,5,6\}$. Näyteavaruuden alkioita kutsutaan alkeistapauksiksi (engl. *outcome*).

3. Satunnaismuuttujan maalijoukko voi olla myös jokin muu kuin reaalilukujen joukko, mutta tässä tutkielmassa keskitytään vain satunnaismuuttujiin, joiden maalijoukko on reaalilukujen joukko.

esittää funktiona[4] $X : \Omega \to \mathbb{R}$. Tässä luvussa $A$:lla merkitään satunnaismuuttujan arvojoukkoa, toisin sanoen $A \subset \mathbb{R}$, ja $x$:llä satunnaismuuttujan $X$ arvoa alkeistapaukselle $\omega$ eli $\omega$: $X(\omega) = x$, missä $x \in A$. Satunnaismuuttuja on diskreetti, jos sen arvojoukko on äärellinen tai numeroituvasti ääretön[5], kun jatkuvan satunnaismuuttujan arvojoukko puolestaan on jokin reaaliakselin osaväli.

Satunnaismuuttujaan liittyy käsite todennäköisyysjakauma, joka kertoo satunnaismuuttujan näyteavaruuden todennäköisyyksien jakautumisen näyteavaruuden alkeistapausten kesken. Diskreetin satunnaismuuttujan alkeistapausten todennäköisyyksiä kuvataan pistetodennäköisyysfunktion (engl. *probability mass function*, *pmf*) avulla. Pistetodennäköisyysfunktio on kuvaus $P : \mathbb{R} \to [0,1]$, jolle pätee[6]

$$\sum_x P(x) = 1 \tag{4.1}$$

ja

$$\forall x : P(x) \geq 0. \tag{4.2}$$

Jatkuvan satunnaismuuttujan näyteavaruuden alkeistapausten todennäköisyyksien jakaumaa kuvataan tiheysfunktion (engl. *probability density function*, *pdf*) avulla. Tiheysfunktio kertoo todennäköisyyden reaalilukujen välille[7]. Samoin kuin pistetodennäköisyysfunktiolle, tiheysfunktiolle pätee

$$\int_{-\infty}^{\infty} p(x)dx = 1 \tag{4.3}$$

ja

4. Oletetaan, että funktio $X$ on *mitallinen*.

5. Esimerkiksi luonnollisten lukujen joukko on numeroituvasti ääretön.

6. Merkitään $P(x)$ kuvaamaan todennäköisyyttä tapahtumalle, jossa diskreetti satunnaismuuttuja $X$ saa arvon $x$ eli $P(X = x)$. Vastaavasti jatkuvan satunnaismuuttujan tapahtumalle käytetään merkintää $p(x)$.

7. Yleensä ollaan kiinnostuneita infitesimaalisen välin todennäköisyydestä, joka tietokonegrafiikassa voidaan ajatella liukuluvun esitysmuodon tarkkuuden määräämäksi.

$$\forall x : p(x) \geq 0. \tag{4.4}$$

Polunseurantamenetelmässä todennäköisyysjakaumia käytetään osana Monte Carlo -integrointia näytteistettävien infitesimaalisten avaruuskulmien määräytymiseen. Näyteavaruus koostuu pinta-ala-alkion normaalin määräämän puoliavaruuden kaikista mahdollisista infitesimaalisista avaruuskulmista tai sen osajoukosta[8]. Tyypillinen todennäköisyysjakauma tuntemattoman funktion, kuten esimerkiksi jossain tapauksissa renderöintiyhtälön, approksimointiin on tasajakauma (engl. *uniform distribution*), joka voidaan reaalivälille $[a,b]$ esittää tiheysfunktiona muodossa

$$p(x) = \begin{cases} \frac{1}{b-a} & \text{kun} x \in [a,b] \\ 0 & \text{kun} x \notin [a,b], \end{cases} \tag{4.5}$$

missä $a < b$. Toinen tunnettu ja esimerkiksi luvussa 4.3 esiteltävän häiriönpoiston yhteydessä vastaan tuleva jakauma on normaalijakauma eli Gaussin jakauma, joka voidaan esittää muodossa

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{4.6}$$

missä $\sigma^2$ on varianssi ja $\mu$ on keskiarvo, joka voidaan korvata seuraavaksi esiteltävällä satunnaismuuttujan odotusarvolla $E(X)$. Satunnaismuuttujan odotusarvo on satunnaisilmiön tuottamien lukujen odotettavissa oleva arvo[9]. Diskreetin satunnaismuuttujan odotusarvo[10] on

$$E(X) = \sum_{x \in A} xP(x) \tag{4.7}$$

8. Toisin sanoen seurattavan säteen suunta määräytyy satunnaisilmiön mukaisesti, missä käytetty todennäköisyysjakauma määrittää eri suunnille todennäköisyydet.

9. Tämä on satunnaismuuttujan todennäköisyysjakauman mukaisesti painotettu satunnaismuuttujan arvojoukon aritmeettinen keskiarvo.

10. Helposti omaksuttava esimerkki diskreetin satunnaismuuttujan odotusarvosta on nopanheiton odotusarvo $E(X) = \frac{1+2+3+4+5+6}{6} = 3{,}5$.

ja jatkuvan satunnaismuuttujan odotusarvo on

$$E(X) = \int_{x \in A} xp(x)dx. \tag{4.8}$$

Toisaalta odotusarvo voidaan laskea myös jollekin toiselle funktiolle $f$, joka puolestaan kuvaa satunnaismuuttujan $X$ reaalilukujen joukkoon. Tällöin myös funktio $f(X)$ on satunnaismuuttuja, jonka odotusarvo on

$$E(f) = \int_{x \in A} f(x)p(x)dx. \tag{4.9}$$

Näin ollen, jos satunnaismuuttuja $X$ kuvaa avaruuskulman $x$, voidaan satunnaismuuttujalla $f(x)$ kuvata esimerkiksi avaruuskulmasta $x$ saapuva radianssi. Odotusarvon määritelmästä voidaan huomata, että odotusarvo on riippuvainen tiheysfunktiosta, mikä mahdollistaa esimerkiksi approksimoitavalle funktiolle sopivampien tiheysfunktioiden käytön näytteistämisessä kuten esitellään luvussa 4.3. Odotusarvon avulla satunnaismuuttujalle voidaan johtaa toinen tutkielman kannalta merkittävä ominaisuus, satunnaismuuttujan varianssi[11]

$$Var(X) = E((X - E(X))^2), \tag{4.10}$$

jolloin diskreetin satunnaismuuttujan varianssi voidaan esittää myös muodossa

$$Var(X) = \sum_{x \in A} P(x)(x - E(X))^2 \tag{4.11}$$

ja jatkuvan satunnaismuuttujan varianssi muodossa

$$Var(X) = \int_{x \in A} p(x)(x - E(X))^2 dx. \tag{4.12}$$

Satunnaismuuttujan keskihajonta puolestaan on $\sigma(X) = \sqrt{Var(X)}$.

11. Merkitään sekaannusten välttämiseksi satunnaismuuttujan varianssia $Var(X) = \sigma^2(X)$.

## 4.2 Monte Carlo -integrointi

Kuten aikaisemminkin todettiin, perustuvat Monte Carlo -menetelmät satunnaisuuden hyödyntämiseen deterministisen ongelmien ratkaisemisessa. Monte Carlo -integrointi menetelmässä integraali korvautuu odotusarvolla. Näin ollen, jos meillä jokin mitallinen funktio $f : \mathbb{R} \to \mathbb{R}$, jonka integraalia reaaliakselin osavälille $A$ ratkaistaan:

$$F = \int_{x \in A} f(x)dx, \tag{4.13}$$

sekä $N$ kappaletta näytteitä[12] $X_1$,...,$X_N$ valittuna todennäköisyysjakauman $p$ mukaisesti, voidaan Monte Carlo -integroinnin mukaisesti integraali korvata tällöin estimaattorilla

$$F_N = \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)}. \tag{4.14}$$

Edellä esitettyä estimaattoria kutsutaan Monte Carlo -estimaattoriksi ja estimaattorin tuottamaa yksittäistä arvoa estimaatiksi. Estimaattori on harhaton (engl. *unbiased*) jos $p(x) > 0$, kun $f(x) > 0$ ja $p(x) = 0$, kun $f(x) = 0$ kaikilla $x$. Harhattoman Monte Carlo -estimaattorin odotusarvo on

$$\begin{aligned} E(F_N) &= E\left(\frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)}\right) \\ &= \frac{1}{N} \sum_{i=1}^{N} E\left(\frac{f(X_i)}{p(X_i)}\right) \\ &= \frac{1}{N} \sum_{i=1}^{N} \int_{x \in A} \frac{f(x)}{p(x)} p(x)dx \\ &= \frac{1}{N} \sum_{i=1}^{N} \int_{x \in A} f(x)dx \\ &= \int_{x \in A} f(x)dx, \end{aligned} \tag{4.15}$$

12. Tietokonegrafiikan kirjallisuudessa tyypillisesti, kuten myös tässä tutkielmassa, käytetään termiä näyte tarkoittamaan samaa kuin havainto (engl. *observation*) tilastotieteessä (Veach 1998).

jolloin harhattoman estimaattorin odotusarvo on sama kuin funktion todellinen arvo. Suurten lukujen lain (engl. *law of large numbers*) mukaisesti estimaattorin tuottama approksimaatio lähenee sen *odotusarvoa*, kun näytteiden määrää $N$ kasvatetaan. Jos estimaattorin tuottama approksimaatio lähenee funktion *todellista arvoa* näytteiden määrän lähentyessä äärettömyyttä, estimaattorin sanotaan olevan tarkentuva (engl. *consistent*). Monte Carlo -estimaattorin varianssi ja keskihajonta (Veach 1998) ovat

$$Var(F_N) = Var\left(\frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{p(x_i)}\right) = \frac{Var\left(\frac{f(X)}{p(X)}\right)}{N} \tag{4.16}$$

ja

$$\sigma(F_N) = \frac{1}{\sqrt{N}}\sigma\left(\frac{f(x)}{p(x)}\right), \tag{4.17}$$

jolloin *estimaattorin varianssi* ja siten keskihajonta pienevät näytemäärää kasvattamalla.

Erityisen hyvin Monte Carlo -integrointi soveltuu moniulotteisten integraalien ratkaisemiseen käytännössä, joiden analyyttinen ratkaiseminen on vaikeaa tai mahdotonta. Tätä puoltaa menetelmän yksinkertaisuuden lisäksi estimaatin tarkentuminen lineaarisesti näytteiden kaksinkertaistuessa eli nopeudella $O(\sqrt{N})$ riippumatta ulottuvuuksien lukumäärästä (Veach 1998). Tutkielman kannalta merkittävä käyttökohde Monte Carlo -integroinnille on polunseurantamenetelmän avulla tapahtuva renderöintiyhtälön approksimointi, mikä voidaan mieltää ääretönulotteisen[13] integraalin approksimoimiseksi. Polunseurantamenetelmän yhteydessä näytteellä tarkoitetaan yhtä seurattua polkua[14], jolla on äärellinen syvyys[15] eli kimmokkeiden lukumäärä. Merkitsemällä polkujen eli näytteiden lukumäärää $N$:llä voidaan renderöintiyhtälön approksimaatio pinta-ala-alkiolle $p_x$ esittää Monte Carlo -integrointia käyttäen muodossa

$$L_o(p_x, \mathbf{v}) = \int_{2\pi} f_r(p_x, \mathbf{l}, \mathbf{v}) L_i(p_x, \mathbf{l}) \mathbf{n} \cdot \mathbf{l} d\mathbf{l} \approx \frac{2\pi}{N} \sum_{i=0}^{N} f_r(p_x, \mathbf{l_i}, \mathbf{v}) L_i(p_x, \mathbf{l_i}) \mathbf{n} \cdot \mathbf{l_i}, \tag{4.18}$$

13. Tämä on seurausta yhtälön rekursiivisuudesta ilman rekursion päätösehtoa.
14. Näin ollen näytemäärä ei vielä yksistään kerro seurattujen säteiden lukumäärää.
15. Reaaliaikaisessa renderöinnissä syvyys on yleensä hyvinkin alhainen, kuten yksi tai kaksi kimmoketta.

missä $l_i$ on satunnaisilmiön määräämä näytteistettävä avaruuskulma. Esimerkiksi kuviossa 18 satunnaisilmiön määräämät näytteistettävät avaruuskulmat ovat merkitty nuolin ja pinta-ala-alkioiden normaalien määräämät puoliavaruuksien rajoittamat avaruuskulmien arvojoukot puoliympyröin.

Polunseurantamenetelmän heikkoutena on tyypillisesti estimaattorin suuri varianssi, mikä ilmenee häiriönä kuvassa. Estimaattorin varianssia on mahdollista pienentää kasvattamalla näytteiden lukumäärää, mikä ei reaaliaikaisessa renderöinnissä useinkaan ole mahdollista. Niinpä reaaliaikaisessa, kuin myös ei-reaaliaikaisessakin, renderöinnissä käytetään muita tekniikoita estimaattorin varianssin pienentämiseksi, joita esitellään seuraavaksi.

## 4.3 Näytteistäminen

Sen lisäksi, että Monte Carlo -estimaattorin varianssia voidaan pienentää näytteiden lukumäärää kasvattamalla, voidaan varianssia pienentää käyttämällä sopivaa todennäköisyysjakaumaa näytteistämisessä. Tämä tapahtuu valitsemalla todennäköisyysjakauma, joka mahdollisimman hyvin myötäilee approksimoitavaa funktiota, jolloin todennäköisyys on korkea niille näytteille, joiden arvokin on korkea ja alhainen niille näytteille, joiden arvo on alhainen. Kuviossa 19 esitetään saman funktion näytteistäminen erilaisten todennäköisyysjakaumien mukaisesti. Kuvion ylimmäisellä rivillä kahdella eri näytemäärällä esitetään näytteistäminen tasajakauman mukaisesti, mikä on tyypillinen todennäköisyysjakauma tuntemattoman funktion näytteistämiseen[16].

Kun funktio puolestaan tunnetaan ainakin osittain, voidaan funktion näytteistämisessä käyttää tasajakaumasta poikkeavaa todennäköisyysjakaumaa eli painottaa eri näytteiden todennäköisyyksiä. Tästä käytetään nimitystä painotettu näytteistys (engl. *importance sampling*, *IS*)[17]. Painotettu näytteistäminen erilaisten todennäköisyysjakaumien mukaisesti esitetään kuviossa 19 keskimmäisellä ja alimmaisella riveillä. Kuviossa vasemmalla keskellä on painotetun näytteistyksen tavoite: verrannollinen todennäköisyysjakauma. Tällöin estimaattorin

16. Kuviosta voi huomata, että estimaattorin varianssi ja siten keskihajonta pienenevät näytteiden lukumäärän kasvaessa, jolloin suuremmalla näytemäärällä estimaatti on todennäköisemmin lähempänä odotusarvoa.

17. Jälleen, jotta estimaattori olisi harhaton, tulee todennäköisyysjakauman saada positiivisia arvoja, kun approksimoitava funktio saa ja nolla, kun approksimoitavan funktion arvo on nolla.

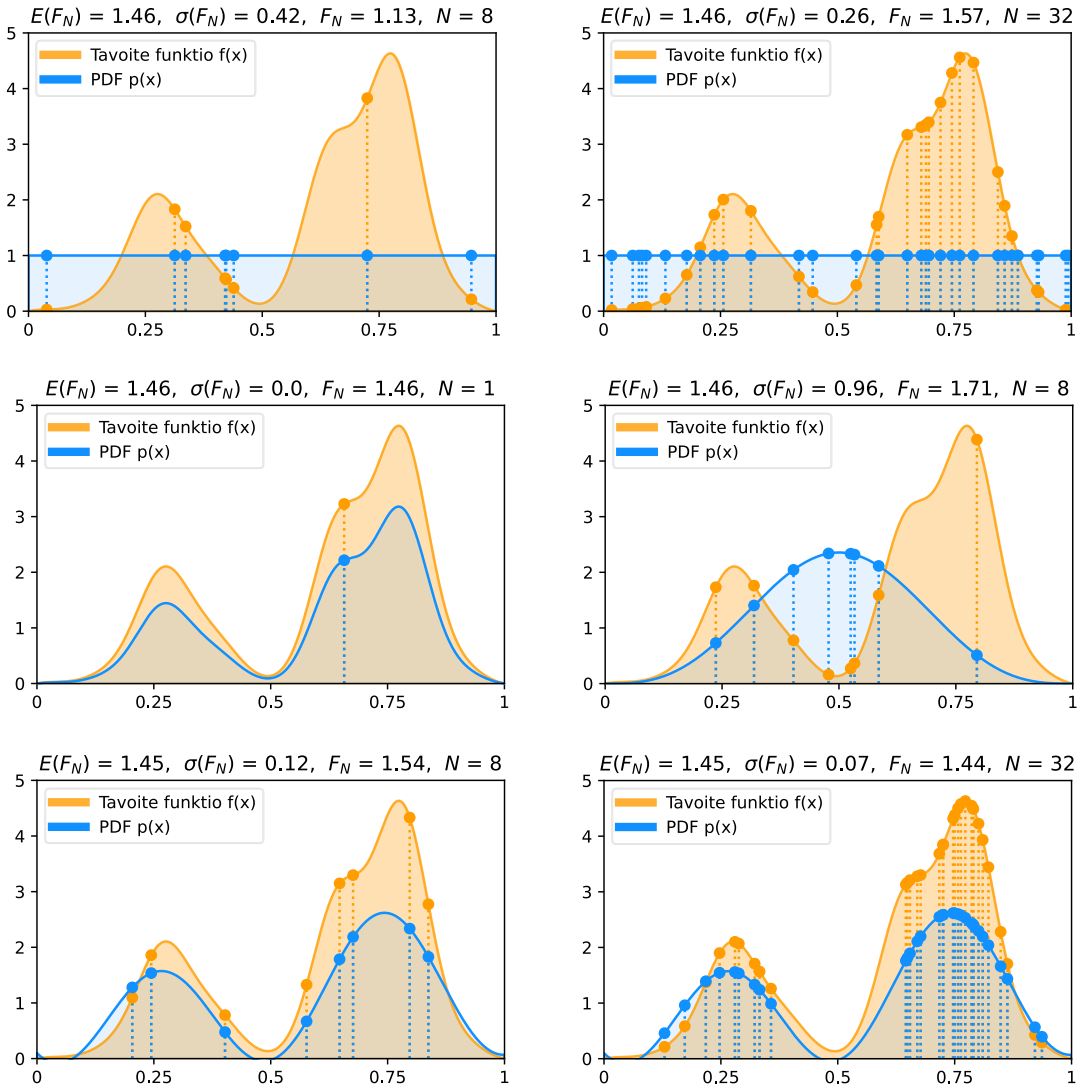
varianssi on nolla ja tarkka odotusarvo saadaan muodostettua yhdellä näytteellä.

Verrannollisen todennäköisyysjakauman muodostaminen ei useinkaan ole mahdollista tai kannattavaa, jolloin pyrkimyksenä on valita todennäköisyysjakauma, joka mahdollisimman hyvin myötäilee approksimoitavaa funktiota. Kuvion alimmaisella rivillä käytetään todennäköisyysjakaumaa, joka melko hyvin myötäilee approksimoitavaa funktiota. On kuitenkin huomioitava, että kyseinen todennäköisyysjakauma saa arvokseen nolla pisteissä, joissa approksimoitava funktio ei saa, jolloin estimaattori on harhallinen. Tämä ilmenee myös estimaattorin odotusarvossa, joka ei ole funktion todellinen arvo. Toisaalta kyseistä todennäköisyysjakaumaa käyttämällä funktiolle saadaan todennäköisesti parempi estimaatti kuin tasajakaumaa käytettäessä lukuun ottamatta hyvin suuria näytemääriä, jolloin estimaatti lähenee harhallista odotusarvoa. Tämä näkyy myös kuviossa, jossa kyseistä todennäköisyysjakaumaa käyttämällä saadaan parempi estimaatti funktion todellisesta arvosta sekä 8:lla että 32:lla näytteellä verrattuna tasajakaumaan. Kuviossa keskellä oikealla puolestaan esitetään heikosti approksimoitavaa funktiota myötäilevän todennäköisyysjakauman käyttö näytteistämisessä, mikä johtaa korkeaan estimaattorin varianssiin.

Polunseurantamenetelmässä painotettu näytteistys tarkoittaa esimerkiksi stokastista valonlähteiden tai kaksisuuntaisen heijastusjakauman mukaisesti painotettua näytteistämistä. Väitöskirjassaan Veach (1998) esittelee näytteistämisen painottamisen joko valonlähteiden tai pinnan kaksisuuntaisen heijastusjakauman mukaisesti, sekä näiden kahden niin kutsutun strategian soveltuvuus erilaisiin tilanteisiin. Koska kumpikaan strategia ei sovellu kaikkiin ja usein päinvastaisiin tilanteisiin, esitellään väitöskirjassa useampaa strategiaa yhdistävä menetelmä nimeltä monipainotettu näytteistys (engl. *multiple importance sampling*, *MIS*). Kun valitaan $M$ kappaletta strategioita $s$ näytteistämisen painottamiseen ja tuotetaan $N_s$ näytettä jokaisesta strategiasta, voidaan monipainotetun näytteistyksen estimaattori esittää muodossa

$$F_{N_{mis}}^{M} = \sum_{s=1}^{M} \frac{1}{N_s} \sum_{i=1}^{N_s} w_s(X_{s,i}) \frac{f(X_{s,i})}{p_s(X_{s,i})}, \tag{4.19}$$

missä $w_s$ on strategiakohtainen painofunktio. Tyypillinen painofunktio on Veachin (1998) väitöskirjassa esitelty niin kutsuttu tasapainoheuristiikka (engl. *balance heuristic*) $w_s = \frac{N_s p_s(x)}{\sum_{j=1}^{M} N_j p_j(x)}$ (Bitterli ym. 2020). Jotta monipainotetun näytteistyksen estimaattori on harhaton, tulee pai-




Kuvio 19. Erilaisia todennäköisyysjakaumia funktion $f(x)$ (oranssi) näytteistämiseen: tasajakauma (ylhäällä), verrannollinen eli ideaalinen PDF (keskellä vasemmalla), heikosti funktiota myötäilevä painotettu PDF (keskellä oikealla) ja hyvin funktiota myötäilevä painotettu PDF (alhaalla).

nofunktiolle $w_s$ päteä

$$\begin{gathered} \sum_{s=1}^{M} w_s(x) = 1, \text{ kun } f(x) \neq 0 \\ \text{ja} \\ w_s(x) = 0, \text{ kun } p_s(x) = 0. \end{gathered} \tag{4.20}$$

Yksi polunseurannassa tyypillisesti käytetty monipainotettu näytteistyksen menetelmä on NEE (engl. *next event estimation*) -menetelmä, mitä on hyödynnetty muun muassa kuviossa 13 esitetyn suoran ja epäsuoran valaistuksen kuvaamiseen. Myös kuviossa 18 esitetty polunseuranta noudattaa NEE-menetelmää, jossa jokaisesta kimmokkeesta sen lisäksi, että polkua jatketaan kaksisuuntaisen heijastusjakauman mukaisesti painottaen epäsuoran valaistuksen kuvaamiseksi, seurataan yksi säde kohti satunnaisesti valittua valonlähdettä suoran valaistuksen selvittämiseksi leikkauspisteelle. Näin ollen polku haarautuu jokaisen kimmokkeen kohdalla, mutta vain toista haaraa jatketaan rekursiivisesti halutulle syvyydelle.

Kuten aikaisemmin mainittiin, on painotetun näytteistyksen edellytyksenä, että approksimoitava funktio tunnetaan ainakin osittain. Kun approksimoitavaa funktiota ei tunneta etukäteen tai funktio on liian monimutkainen, jotta sille voitaisiin muodostaa sopiva todennäköisyysjakauma, voidaan soveltaa niin kutsuttuja uudelleen näytteistämismenetelmiä (engl. *resampling*). Tutkielman kannalta oleellinen uudelleen näytteistämismenetelmä on Talbot, Cline ja Egbert (2005) esittelemä RIS (engl. *resampled importance sampling*) -menetelmä, jossa yhdistyvät painotettu näytteistys ja SIR (engl. *sampling importance resampling*, *SIR*) (Rubin 1987)[18] -menetelmä.

RIS-menetelmä mahdollistaa näytteistämisen lähes ideaalisesta todennäköisyysjakaumasta $g$ ilman, että näytteistettävää funktiota $f$ tunnetaan etukäteen. Menetelmässä approksimoitavaa funktiota näytteistetään käyttäen jotain yksinkertaista todennäköisyysjakaumaa $p$ muodostaen $M$ kappaletta kandidaattinäytteitä $X = \{X_1, X_2, ..., X_M\}$, jonka jälkeen jokaiselle näytteelle lasketaan painokerroin $w(X_j) = \frac{g(X_j)}{p(X_j)}$ perustuen tavoitetodennäköisyysjakaumaan $g$. Normalisoitujen painokertoimien mukaisesti kandidaattinäytteistä valitaan $N$ kappaletta lopullisia näyteitä $Y = \{Y_1, Y_2, ..., Y_N\}$ estimaatin muodostamiseksi. Tällöin näyt-

18. Poiketen alkuperäisestä esittelystään (Rubin 1987) menetelmästä käytetään myös nimitystä IR (engl. *importance resampling*, *IR*), kuten esimerkiksi artikkelissa (Talbot, Cline ja Egbert 2005).

teet $Y = \{Y_1, Y_2, ..., Y_N\}$ tulevat valituksi tavoitetodennäköisyysjakauman $g$ mukaisesti. RIS-estimaattori voidaan esittää muodossa

$$F^M_{N_{ris}} = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{f(Y_i)}{g(Y_i)}\right)\frac{1}{M}\sum_{j=1}^{M} w(X_j) = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{f(Y_i)}{g(Y_i)}\cdot\frac{1}{M}\sum_{j=1}^{M}\frac{g(X_j)}{p(X_j)}\right). \tag{4.21}$$

Näin ollen valittujen näytteiden $Y$ jakautuminen tavoitetodennäköisyysjakauman mukaisesti tarkentuu kandidaattinäytteiden lukumäärän $M$ kasvaessa. RIS-estimaattori on harhaton, kun $M,N \geq 1$ ja todennäköisyysjakaumat $p$ ja $g$ saavat positiivisia arvoja, kun approksimoitava funktio $f$ saa ja nolla, kun approksimoitavan funktion arvo on nolla (Talbot 2005). Siinä, missä edellä esitetty RIS-estimaattori sallii yhden näytteistettävän todennäköisyysjakauman käytön, esittelee Talbot (2005) tutkielmassaan myös monipainotettuun näytteistykseen perustuvan RIS-estimaattorin. Kun valitaan $K$ kappaletta strategioita näytteiden painottamiseen ja merkitään strategioita vastaavia todennäköisyysjakaumia $p_1, ..., p_K$, voidaan yhtälössä 4.21 esitetyn RIS-estimaattorin painofunktio $w(X_j)$ korvata painofunktiolla[19]

$$W(X_j) = \frac{M_i p_i(X_j)}{\sum_{k=1}^{K} M_k p_k(X_j)}\frac{g(X_j)}{p_i(X_j)}, \tag{4.22}$$

missä $M_i$ on todennäköisyysjakauman $p_i$ mukaisesti valittavien kandidaattinäytteiden lukumäärä. Yksi intuitiivinen esimerkki RIS-menetelmästä on suoran valaistuksen laskenta virtuaalimaailmassa, jossa valonlähteiden lukumäärä on suuri, jolloin näkyvyystarkastelu jokaisen tarkasteltavan pinta-ala-alkion ja jokaisen virtuaalimaailman valonlähteen välillä olisi laskennallisesti liian raskas suorittaa[20]. Tällöin käyttäen jotain tunnettua todennäköisyysjakaumaa $p$, kuten tasajakaumaa, voidaan valita $M$ kappaletta valonlähteitä edustavia kandidaattinäytteitä $X = \{X_1, X_2, ..., X_M\}$, jonka jälkeen näytteille muodostetaan normalisoidut painoarvot ratkaisemalla suora valaistus ilman näkyvyys termiä $V(x,y)$. Näin ollen tavoitetodennäköisyysjakauma $g$ perustuu suoran valaistuksen arvoihin ilman, että näkyvyyttä huomioidaan. Estimaatin muodostamiseksi painotetuista kandidaattinäytteistä valitaan $N$ kappa-

19. Painofunktio $W(X_j)$ perustuu Veachin tasapainoheuristiikkaan.

20. Suoran valaistuksen yhtälön 2.24 näkyvyystermi $V(x,y)$ on tyypillisesti yhtälön laskennallisesti raskain termi, sillä se vaatii näkyvyystarkastelun, mikä säteenseurantatekniikalla tarkoittaa säteen seuraamista pinta-ala-alkion ja valonlähteen välillä.

letta näytteitä, joille suoritetaan suoran valaistuksen laskenta sisältäen näkyvyystermin.

## 4.4 Häiriönpoisto

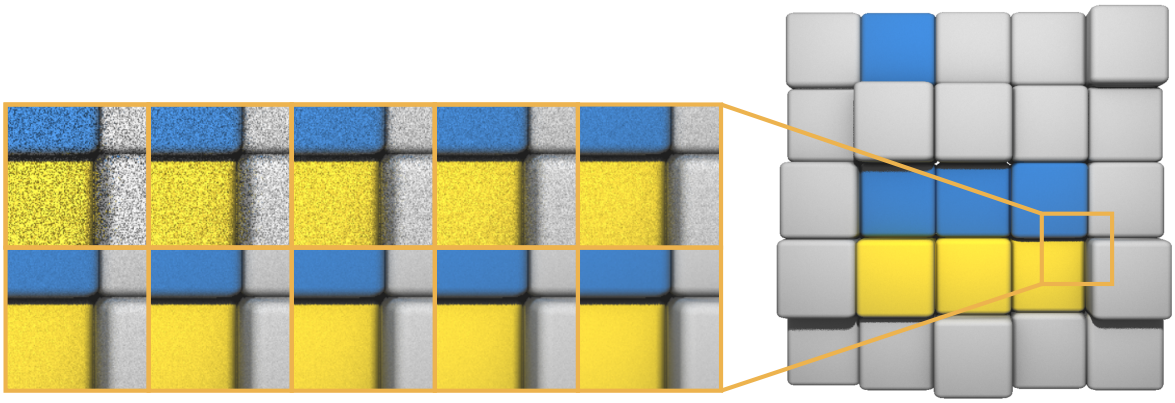
Huolimatta tehokkaasta näytteistämisestä, ovat renderöintiyhtälön ratkaisut yleensä liian monimutkaisia, jotta Monte Carlo -estimaattorin varianssia saataisiin näytteistämisellä häivytettyä kokonaan ilman suuria näytemääriä, mikä ilmenee häiriönä kuvassa. Toisaalta jo hyvin pelkistetyssäkin virtuaalimaailmassa renderöintiyhtälön approksimaatio polunseurantamenetelmällä voi tuottaa häiriötä alhaisella näytemäärällä, mikä on havainnollistettu kuviossa 20, missä virtuaalimaailma sisältää vain yhden kuvassa esiintyvän geometrian pinnalta lähes pistemäisenä näyttäytyvän pinta-alallisen valonlähteen. Sen lisäksi, että varianssia voidaan vähentää hyödyntämällä erilaisia näytteistämisen menetelmiä, voidaan varianssia vähentää myös näytteistämisen jälkeen häiriönpoistomenetelmiä soveltamalla. Tyypillinen näytemäärä reaaliaikaisessa renderöinnissä on yksi tai korkeintaan muutama *uusi* näyte kehyksellä pikseliä kohden, jolloin häiriönpoiston sijasta on usein sopivampaa puhua rekonstruktiosta (engl. *reconstruction*). Tällöin häiriönpoistamiseksi tarvitaan muutakin dataa, kuten esimerkiksi G-puskuriin tallennettuja häiriöttömiä kuvapuskureita. Tutkielmassa ei erikseen käytetä termiä rekonstruktio, vaan pidättäydytään häiriönpoisto termissä tarkoittaen näytteistämisen jälkeen tapahtuvaa[21] varianssin vähentämistä.

Siinä, missä häiriönpoisto vähentää varianssia, luo se tyypillisesti kuitenkin harhan[22]. Niinpä häiriönpoiston tehtävänä ei useinkaan ole puhtaasti oikeanlaisten, vaan visuaalisesti miellyttävämpien eli häiriöttömämpien kuvien tuottaminen. Häiriönpoisto voidaan suorittaa spatiaalisessa avaruudessa tai taajuusavaruudessa, joiden lisäksi liikkuvaa kuvaa renderöitäessä on usein mahdollista hyödyntää käytettävissä olevaa dataa ajallisesti. Spatiaalisella avaruudella

21. Näin ollen tutkielmassa ei esimerkiksi näytteistämisessä tapahtuvaa näytteiden uudelleenkäyttöä, kuten artikkelissa (Bitterli ym. 2020) esiteltyä ReSTIR-menetelmää, katsota varsinaiseksi häiriönpoistomenetelmäksi.

22. Häiriönpoistossa tyypillistä on yhdistellä eri pikseleiden näytteitä, mikä tarkoittaa eri Monte Carlo estimaattoreiden tuottamien estimaattien yhdistämistä joiden ratkaisuavaruudet ovat erilaiset. Toisaalta näytteiden yhdistämisessä usein pyritään yhdistelemään näytteitä, joiden ratkaisuavaruudet ovat likimäin samoja, kuten vierekkäisten pinta-ala-alkioiden, jolloin näytteiden yhdistämisestä aiheutuma harha voi olla merkityksettömän pieni.

tarkoitetaan pikseleiden arvojen ja sijaintien muodostamaa avaruutta. Taajuudella kuvankäsittelyssä voidaan käsittää pikselien arvojen muutos spatiaalisessa avaruudessa. Esimerkiksi, jos kahden viereisen pikselin arvot poikkeavat suuresti toisistaan, puhutaan korkeasta taajuudesta, mikä ennakoi esimerkiksi häiriötä tai kahden kuvassa esiintyvän esineen välistä rajaa. Häiriönpoiston tavoitteena usein on sumentaa eli suodattaa *häiriöön viittaavat*[23] korkeat taajuudet säilyttäen samalla matalat taajuudet. Kun suodatetaan korkeat taajuudet ja säilytetään matalat taajuudet, puhutaan alipäästösuodattimisesta (engl. *low-pass filtering*)[24].



Kuvio 20. Suoran ja epäsuoran valaistuksen kuvaaminen diffuusi pinnalle eri näytemäärillä. Tarkennetussa näkymässä vasemmalla näytemäärät pikseliä kohden vasemmalta oikealle ja ylhäältä alas ovat 1, 2, 4, 8, 16, 32, 64, 128, 256 ja 512. Näkymä oikealla on tuotettu käyttäen 1024 näytettä pikseliä kohden. Jokaista näytettä kohden seurataan neljä sädettä.

Perinteinen spatiaalisessa avaruudessa suoritettava häiriönpoistotekniikka kuvankäsittelyssä on lineaaristen filttereiden avulla tapahtuva sumentaminen[25], joka voidaan esittää diskreettinä konvoluutiona. Konvoluutio tarkoittaa kahden funktion, kuten kuvan $c(\mathbf{p})$ ja niin kutsutun ytimen (engl. *kernel*, *filter*, *mask*) $h(\mathbf{p})$, välistä operaatiota, joka tuottaa uuden funktion $c(\mathbf{p}) * h(\mathbf{p})$[26]. Koska kuva itsessään on diskreetti funktio pikseleiden sijainnin ja niitä vastaavien arvojen välillä, ollaan häiriönpoistossa kiinnostuneita diskreetistä konvoluutiosta, joka

23. Sumentaminen tulisi nimenomaan tapahtua siten, että muut korkeat taajuudet, kuten esimerkiksi virtuaalimaailman geometrioiden väliset rajat, eivät sumentuisi.

24. Päinvastoin kuvan terävöittäminen tapahtuu ylipäästösuodattimen (engl. *high-pass filter*) avulla.

25. Sumentamisen ohella erilaisten lineaaristen filttereiden avulla voidaan muun muassa terävöittää tai siirtää (engl. *shift*) kuvaa.

26. Merkitään pikselin koordinaatteja $\mathbf{p} = (x,y)$.

voidaan kuvan $c$ yksittäiselle pikselille $\mathbf{p}$ esittää ytimen $h$ ja ytimeen kuuluvien pikseleiden $\mathbf{q} \in S$ avulla esittää muodossa

$$(f * h)(\mathbf{p}) = \sum_{\mathbf{q} \in S} h(\mathbf{p} - \mathbf{q}) c(\mathbf{q}). \tag{4.23}$$

Diskreetissä konvoluutiossa pikseli korvataan viereisten pikseleiden painotettuna summana ja kokonainen kuva saadaan käsiteltyä suorittamalla konvoluutio jokaiselle kuvan pikselille eli toisin sanoen ytimellä pyyhkäistään kuvan ylitse[27]. Lineaarisen filtterin ytimen arvot ovat vakioita, eivätkä riipu pikseleiden arvoista, mikä tekee lineaarisista filttereistä laskennallisesti yksinkertaisia. Yksi hyvin tunnettu lineaarinen filtteri on Gaussin filtteri, joka voidaan kuvan $c$ yksittäiselle pikselille $\mathbf{p}$ esittää muodossa (Paris ym. 2009)

$$GC(c(\mathbf{p})) = \sum_{\mathbf{q} \in S} G_\sigma(||\mathbf{p} - \mathbf{q}||) c(\mathbf{q}), \tag{4.24}$$

missä $||\cdot||$ on euklidinen etäisyys ja $G_\sigma$ on kaksiulotteinen[28] Gaussin funktio

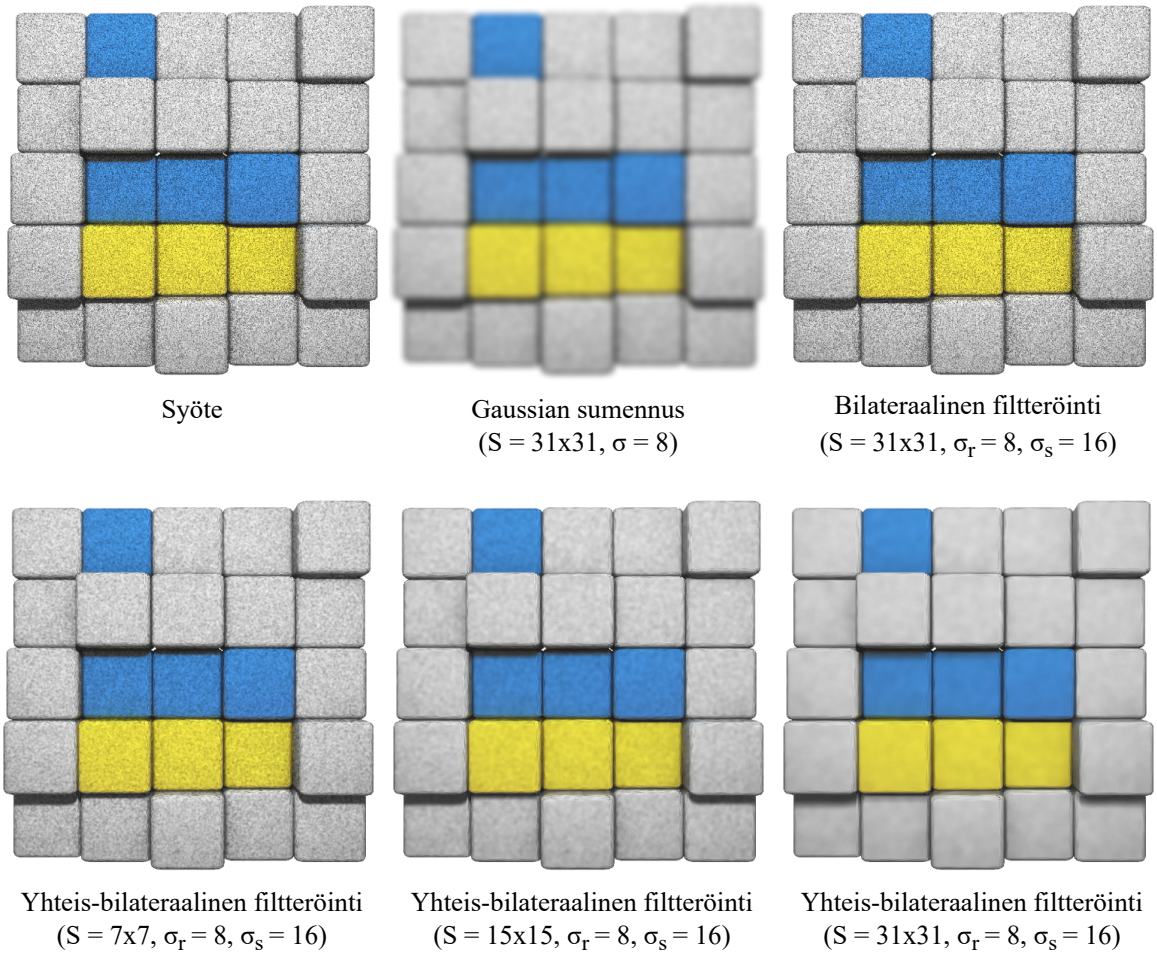
$$G_\sigma(x) = \frac{1}{2\pi\sigma^2} e^{(-\frac{x^2}{2\sigma^2})}. \tag{4.25}$$

Gaussin funktion sisältämän keskihajonnan $\sigma$ kontrolloiminen mahdollistaa sumennuksen voimakkuuden säätämisen yhdessä ytimen koon kanssa. Kuviossa 21 ylhäällä keskellä esitetään Gaussin sumennus. Kuten kuviosta voidaan huomata, johtaa Gaussin funktion ja kuvan välinen konvoluutio koko kuvan sumenemiseen, jolloin myös kuvassa esiintyvien geometrioiden väliset rajat sumenevat, mikä ei useinkaan ole haluttua. Tämä on seurausta vakioarvoisesta ytimestä.

Jotta kuvassa esiintyvien geometrioiden väliset rajat voidaan tunnistaa ja sitä kautta estää

27. Tällöin tulee huomioida ytimen ulottuminen kuvan ulkopuolelle reunapikseleitä käsiteltäessä, jolloin konvoluutio reunapikseleille voidaan suorittaa esimerkiksi laajentamalla kuvaa reunapikseleitä toistamalla.

28. Kenties tyypillisemmin, mutta yhtäpitävästi, kaksiulotteinen Gaussin ydin voidaan esitellä muodossa $G_\sigma(x,y) = \frac{1}{2\pi\sigma^2} e^{(-\frac{x^2+y^2}{2\sigma^2})}$, missä $x$ ja $y$ ovat etäisyydet origosta eli tässä yhteydessä pikselien koordinaatit suhteessa ytimen keskustaan.




Kuvio 21. Yhteis-bilateraalisessa käytetty normaali kuvapuskuria ohjaamaan reunantunnistusta.

niiden sumentaminen, tulee filtteröidessä yksittäisten pikseleiden arvot huomioida ja käyttää niitä filtteröinnin rajoittamiseen, jolloin puhutaan epälineaarisesta filtteröinnistä. Epälineaarista filtteröintiä ei voida yksistään vakioarvoisen ytimen ja kuvan välisenä konvoluutiona esittää, vaan ytimen arvot eli painokertoimet tulee laskea uudelleen pikseleittäin. Laajalti tunnettu epälineaarisen filtteröinnin menetelmä on bilateraalinen filtteröinti, joka ensimmäisen kerran esiteltiin artikkelissa (Tomasi ja Manduchi 1998), vaikkakin menetelmän voidaan katsoa saaneen alkunsa jo vuonna 1995 (Paris ym. 2009). Menetelmässä jokaisen pikselin arvo korvataan pikselin ja sitä lähimpänä olevien pikseleiden arvojen painotettuna summana, missä painotus riippuu sijainnin lisäksi pikselin arvosta, jolloin arvoltaan lähes samanarvoisia ja lähellä toisiaan sijaitsevia pikseleitä painotetaan voimakkaammin. Tällä vältytään su-

mentamasta eri virtuaalimaailman geometrioiden välisiä rajoja ja geometrian sisäisiä reunoja säilyttäen ne terävinä. Käyttäen Gaussin funktiota 4.25 painottamiseen, voidaan bilateraalinen filtteri kuvan $c$ yksittäiselle pikselille $\mathbf{p}$ esittää muodossa (Paris ym. 2009)

$$BF(c(\mathbf{p})) = \frac{1}{k(\mathbf{p})} \sum_{\mathbf{q} \in S} G_{\sigma_s}(||\mathbf{p} - \mathbf{q}||) G_{\sigma_r}(|c(\mathbf{p}) - c(\mathbf{q})|) c(\mathbf{q}), \tag{4.26}$$

missä $|\cdot|$ on itseisarvo ja normalisointitermi $\frac{1}{k(\mathbf{p})}$ takaa sen, että ytimen painojen summa on yksi. Täten, normalisointitermin nimittäjä on

$$k(\mathbf{p}) = \sum_{\mathbf{q} \in S} G_{\sigma_s}(||\mathbf{p} - \mathbf{q}||) G_{\sigma r}(|c(\mathbf{p}) - c(\mathbf{q})|). \tag{4.27}$$

Bilateraalisen filtterin termi $G_{\sigma_s}$ vähentää vaikutusta etäisyyden suhteen ja $G_{\sigma_r}$ vähentää vaikutusta pikselien arvojen mukaisesti. Kuviossa 21 bilateraalinen filtteröinti esitetään ylhäällä oikealla. Kun kuva sisältää paljon häiriötä, ei reunojen tunnistaminen useinkaan onnistu luotettavasti yksittäisestä kuvasta, jolloin bilateraalinen filtteröinti ei tehokkaasti poista häiriötä, mikä on havaittavissa myös kuviossa. Tällöin reunojen tunnistamisen helpottamiseksi voidaan hyödyntää niin kutsuttua yhteis-bilateraalista (engl. *cross-bilateral, joint-bilateral*) filtteröintiä, jossa tarkoituksena on käyttää toista kuvaa ohjaamaan filtteröintiä. Siinä, missä menetelmä alun perin esiteltiin kameralla salamalla ja ilman salamaa otettujen kuvien yhdistämiseen (Paris ym. 2009), voidaan reaaliaikaisen renderöinnin yhteydesssä yhteis-bilateraalisessa filtteröinnissä hyödyntää G-puskuriin tallennettuja häiriöttömiä kuvapuskureita, kuten esimerkiksi syvyysarvot ja normaalit tallentavia kuvapuskureita, filtteröinnin ohjaamiseen eli kuvassa esiintyvien rajojen sumentamisen välttämiseen. Merkitsemällä varsinaista kuvaa $c$:llä ja ohjaavaa kuvaa $g$:llä, voidaan yhteis-bilateraalinen filtteröinti pikselille $\mathbf{p}$ esittää muodossa

$$CBF(c(\mathbf{p}), g(\mathbf{p})) = \frac{1}{k(\mathbf{p})} \sum_{\mathbf{q} \in S} G_{\sigma_s}(||\mathbf{p} - \mathbf{q}||) G_{\sigma_r}(|g(\mathbf{p}) - g(\mathbf{q})|) c(\mathbf{q}), \tag{4.28}$$

jolloin normalisointitermin $\frac{1}{k(\mathbf{p})}$ nimittäjä on

$$k(\mathbf{p}) = \sum_{\mathbf{q} \in S} G_{\sigma_s}(||\mathbf{p} - \mathbf{q}||) G_{\sigma r}(|g(\mathbf{p}) - g(\mathbf{q})|). \tag{4.29}$$

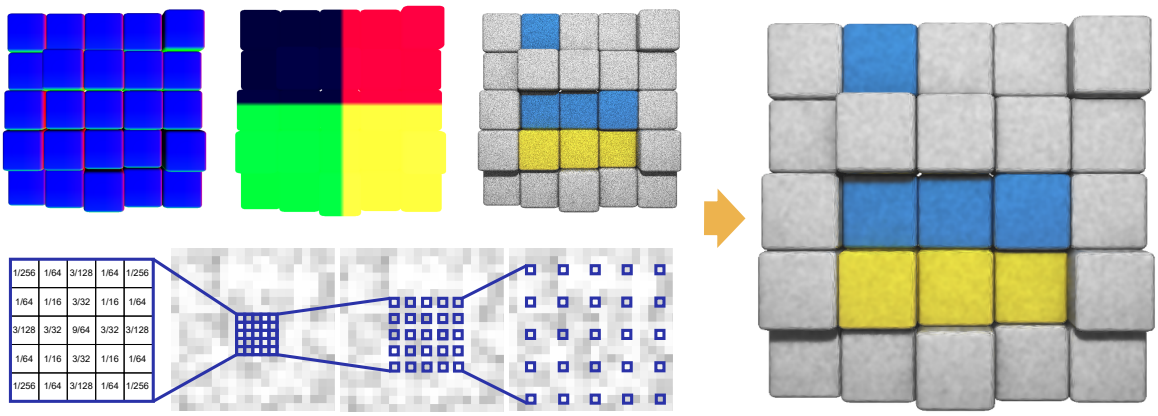
Kuviossa 21 alhaalla esitetään yhteis-bilateraalinen filtteröinti kolmella eri kokoisella ytimellä. Vaikkakin yhteis-bilateraalinen filtteröinti poistaa häiriön hyvin varsinkin isompikokoisilla ytimillä, ovat yhteis-bilateraalinen samoin kuin bilateraalinen filtteröinti tyypillisesti liian raskaita suoritettavaksi reaaliajassa ytimen koon ollessa suuri. Tämä onkin johtanut useampaan bilateraalista ja yhteis-bilateraalista filtteröintiä approksimoivien ja laskennallisesti kevyempien algoritmien kehittämiseen (Fattal, Agrawala ja Rusinkiewicz 2007). Näistä tutkielman kannalta mielenkiintoinen menetelmä on artikkelissa (Dammertz ym. 2010) esitelty niin kutsuttuun à-trous-aallokemuunnokseen (engl. *wavelet transform*) (Holschneider ym. 1989) perustuva virtuaalimaailman geometrioiden väliset rajat säilyttävä häiriönpoistomenetelmä.

Aallokemuunnos, samoin kuin Fourier-muunnos, mahdollistaa signaalin muuntamisen taajuusavaruuteen kuvaten alkuperäisen funktion eri taajuisina osafunktioina, mutta sen sijaan, että osafunktiot olisivat globaaleja kuten Fourier-muunnoksessa, ovat osafunktiot aallokemuunnoksessa lokaaleja[29]. Näin ollen ajallinen tieto, eli kuvankäsittelyn ja diskreetin aallokemuunnoksen yhteydessä spatiaalinen tieto, säilyy muunnoksessa, mikä puoltaa menetelmän käyttöä kuvankäsittelyssä, jossa menetelmää hyödynnetään muun muassa kuvanpakkaukseen[30] ja kuva-analyysin kautta erineviin tarkoituksiin, kuten häiriönpoistoon tai hahmontunnistukseen (engl. *pattern recognition*).

À-Trous-aallokemuunnos (Holschneider ym. 1989) on diskreetti aallokemuunnos, joka pohjautuu Burtin (1981) oivallukseen (engl. *generating kernel*), että laajoja Gaussin ytimiä voidaan approksimoida suorittamalla konvoluutio iteratiivisesti levittämällä ytimen merkitseviä elementtejä samalla säilyttäen ytimen merkitsevien elementtien lukumäärän vakiona. À-Trous-aallokemuunnoksessa kuva hajotetaan eri taajuuksia kuvaaviin (engl. *multiscale*

29. Siinä, missä Fourier-muunnoksessa osafunktiona käytetään äärettömiä eritaajuisia sini- ja kosiniaaltoja, käytetään aallokemuunnoksessa äärellisiä lyhyitä aaltoja eli niin kutsuttuja aallokkeita.

30. JPEG 2000 standardissa (ISO/IEC 15444) käytetään diskreettiä aallokemuunnosta, kun sitä edeltävässä JPEG 1 standardissa (ISO/IEC 10918) puolestaan käytetään diskreettiä Fourier-muunnosta muistuttavaa diskreettiä kosinimuunnosta (Lawson ja Zhu 2002).




Kuvio 22. À-Trous-aallokemuunnokseen pohjautuvan filtteröinnin kolme ensimmäistä iteraatiota. Syötteenä kehyspuskurin ohella reunojen välttämiseksi käytetään G-puskuriin tallennettuja pikseleiden sijainnit ja normaalit sisältäviä kuvapuskureita.

*decomposition*) niin kutsuttuihin aalloketasoihin (engl. *wavelet plane, wavelet coefficient plane*). Resoluutioltaan aalloketasot ovat samansuuruisia alkuperäisen kuvan kanssa, toisin kuin esimerkiksi moniresoluutio hajoitelmassa (engl. *multiresolution decomposition*) (Mallat 1989). Siinä, missä naiivissa diskreetissä aallokemuunnoksessa ytimen kokoa muutetaan eri aalloketasojen kuvaamiseksi, pysyy ytimen merkitsevien elementtien lukumäärä à-trous-aallokemuunnoksessa vakiona eri tasojen kuvaamiseksi. Tämä tapahtuu siten, että joka tasolla à-trous-aallokemuunnoksessa ytimen merkitsevien elementtien etäisyys kaksinkertaistetaan eli ytimeen lisätään merkityksettömiä eli nollaelementtejä merkitsevien elementtien väliin[31]. Kuviossa 22 alhaalla vasemmalla esitetään kolmannen asteen spline-käyrään ($B_3$ spline) perustuva 5x5 ydin ja sen merkitsevien elementtien levittäytyminen kolmella ensimmäisellä tasolla à-trous-aallokemuunnoksessa.

À-Trous-aallokemunnos voidaan suorittaa halutulle määrälle tasoja ja käyttämällä sumentamiseen tarkoitettua ydintä voimistuu sumentaminen tasojen lukumäärän myötä. Jokaisella tasolla $j$, missä $0 \leq j < N$, muodostetaan approksimaatio $c_j(\mathbf{p})$, jota käytetään syötteenä seuraavalle tasolle. Ensimmäisellä tasolla syötteenä toimii alkuperäinen kuva $c_0(\mathbf{p})$. Näin ollen, approksimaatio tasolla $j+1$ voidaan esittää muodossa

31. Tästä tulee algoritmin ranskankielinen nimitys à-trous, joka tarkoittaa rei'itettyä.

$$c_{j+1}(\mathbf{p}) = c_j(\mathbf{p}) * h_j, \tag{4.30}$$

missä $h_j$ on ydin, joka sisältää $2^j - 1$ nolla elementtiä merkitsevien elementtien välissä ytimen riveillä ja sarakkeilla. Tällöin aalloketaso voidaan muodostaa peräkkäisten approksimaatioiden erotuksena

$$d_{j+1} = c_j(\mathbf{p}) - c_{j+1}(\mathbf{p}), \tag{4.31}$$

jolloin kuvan $c$:n aallokemuunnos on $\{d_0, d_1, ..., d_{N-1}, c_N\}$ ja alkuperäisen kuvan rekonstruktio voidaan viimeisimmän approksimaation $c_N$ ja aalloketasojen avulla esittää muodossa

$$c = c_N + \sum_{j=0}^{N} d_j. \tag{4.32}$$

Sen sijaan, että kuva sumennettaisiin kokonaisuudessaan à-trous-aallokemuunnoksen avulla, esitellään artikkelissa (Dammertz ym. 2010) lisäys à-trous-aallokemuunnokseen virtuaalimaailman geometrioiden välisten rajojen ja geometrian sisäisten reunojen sumentamisen välttämiseksi. Rajojen ja reunojen välttäminen tapahtuu samoin kuten bilateraalisessa filtteröinnissä eli käyttämällä pikselien arvoista riippuvaista painofunktiota sumennuksessa, jolloin yhtälö 4.30 muutetaan muotoon

$$c_{j+1}(\mathbf{p}) = \frac{1}{k} \sum_{\mathbf{q} \in S} h_j(\mathbf{q}) \cdot W(\mathbf{p}, \mathbf{q}) \cdot c_j(\mathbf{p}), \tag{4.33}$$

missä $k$ on normalisointitermi

$$k = \sum_{\mathbf{q} \in S} h_j(\mathbf{q}) \cdot W(\mathbf{p}, \mathbf{q}) \tag{4.34}$$

ja painofunktio $W$ on väriarvojen $c$, normaaleiden $n$ ja sijaintien $x$ painofunktioiden[32] tulo

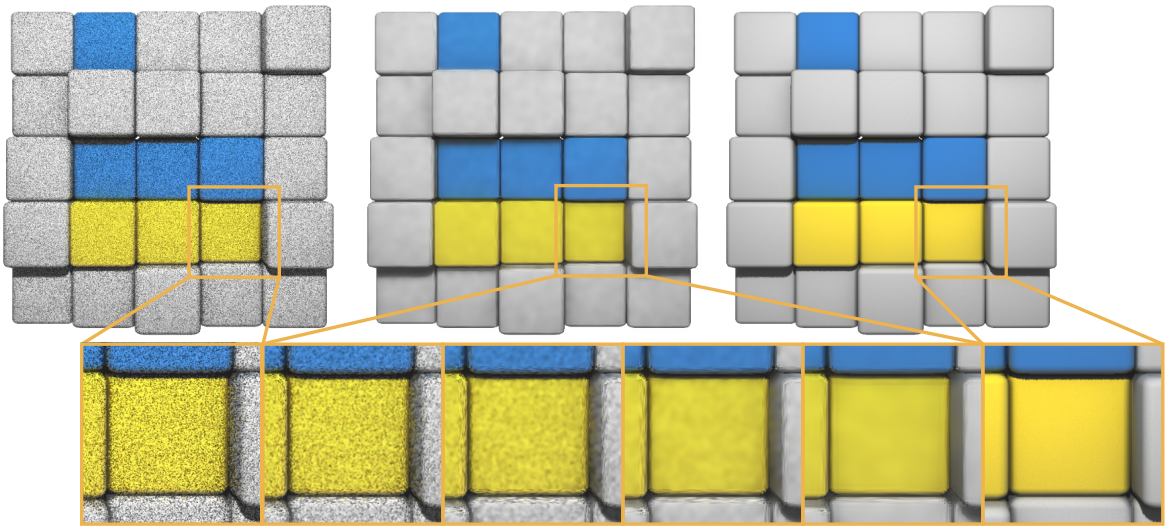
32. Koska väriarvot sisältävä kuvapuskuri sisältää odotetusti paljon häiriötä, parantaa häiriöttömän datan, kuten tässä tapauksessa pikselien normaalien ja sijaintien, käyttö rajojen ja reunojen tunnistamista.

$$W(\mathbf{p},\mathbf{q}) = w_c \cdot w_n \cdot w_x. \tag{4.35}$$

Näistä jokaisen ominaisuuden $o \in \{c,n,x\}$ painofunktio voidaan antaa muodossa

$$w_o(\mathbf{p},\mathbf{q}) = e^{\left(\frac{||f_X(\mathbf{p})-f_X(\mathbf{q})||}{\sigma_X^2}\right)}, \tag{4.36}$$

missä $f_X$ on ominaisuuden tallentava kuvapuskuri. Kuviossa 22 ylhäällä esitetään menetelmän syöte ja oikealla kolmelta ensimmäiseltä tasolta luotu rekonstruktio. Eri tasoilta luodut rekonstruktiot puolestaan esitetään kuviossa 23. Kuviosta voidaan havaita, että häiriön sumentaminen voimistuu tasojen myötä.



Kuvio 23. À-Trous-aallokemuunnokseen pohjautuva reunoja välttävä filtteröinti artikkelissa (Dammertz ym. 2010) esiteltyyn varjostimeen perustuen käyttäen yhtälön 4.36 ominaisuus painofunktioissa normalisoituja arvoja: $\sigma_x = 0{,}450$, $\sigma_c = 0{,}549$ ja $\sigma_n = 0{,}001$. Ylhäällä esitetään syöte (1 näyte/pikseli), tasolta neljä luotu rekonstruktio, sekä referenssi (1024 näytettä/pikseli). Alhaalla vasemmalta oikealle esitetään tarkennetut näkymät syötteestä, rekonstruktiosta neljältä ensimmäiseltä tasolta, sekä referenssistä.

Siinä, missä pysäytyskuvan häiriönpoistossa käytössä on vain yksittäinen kuva, mahdollistaa liikkuvan kuvan renderöinti myös datan hyödyntämisen ajallisesti, kuten tehdään esimerkiksi ajallisessa reunanpehmennyksessä. Säteenseurannan yhteydessä tunnettu ajallisesti

dataa hyödyntävä häiriönpoistomenetelmä on artikkelissa (Schied ym. 2017) esitelty à-trous-aallokemuunnokseen perustuva SVGF (*Spatiotemporal Variance-Guided Filtering*) menetelmä. SVGF-menetelmässä ennen kuvan rajat ja reunat säilyttävän à-trous-aallokemuunnoksen suorittamista kasvatetaan efektiivistä näyte-per-pikseli lukumäärää hyödyntämällä aikaisempien kehyksien näytteitä. Tämä tapahtuu samalla tavalla kuin ajallisessa reunanpehmennyksessä eli ylläpitämällä kehysten välisiä siirtymiä, joiden avulla edellisten kehysten näytteitä kyetään lukemaan.

Näytteiden uudelleenkäytön lisäksi ennen kuvan rajat ja reunat säilyttävän à-trous-aallokemuunnoksen suorittamista yhdistetyille eli uusille ja ajallisesti uudelleenkäytettäville näytteille suoritetaan SVGF-menetelmässä varianssin estimointi, jonka tarkoituksena on estää filtteröinti niissä osissa kuvaa, joissa on vain vähän tai ei ollenkaan häiriötä käyttämällä varianssin estimaattia osana rajat ja reunat säilyttävää painofunktiota. Menetelmässä varianssin estimointi samoin kuin aallokemuunnoksessa rajojen ja reunojen välttäminen osittain perustuu pikseleiden luminanssiarvoihin[33]. Ajallisesti uudelleenkäytettävien näytteiden puuttuessa, varianssin estimoinnissa näytteitä uudelleen käytetään spatiaalisesti[34]. Ennen rajat ja reunat säilyttävän à-trous-aallokemuunnoksen suorittamista, menetelmässä varianssin estimaateissa esiintyvien epävakauksien vähentämiseksi estimaatit filtteröidään käyttäen 3x3 kokoista Gaussin filtteriä.

SVGF-menetelmässä rajat ja reunat säilyttävän à-trous-aallokemuunnoksen painofunktiot perustuvat pikseleiden luminanssi-, syvyys- ja normaaliarvoihin, jolloin yhtälö 4.35 voidaan esittää tasolla $j$ muodossa

$$W_j(\mathbf{p},\mathbf{q}) = w_n \cdot w_z \cdot w_l, \tag{4.37}$$

missä eri ominaisuuksien painofunktiot ovat

33. RGB arvolle voidaan muodostaa approksimaatio luminanssista esimerkiksi kaavan $l(\mathbf{p}) = 0{,}2126 \cdot c_R(\mathbf{p}) + 0{,}7152 \cdot c_G(\mathbf{p}) + 0{,}0722 \cdot c_B(\mathbf{p})$ (ITU-R 2015, 4) mukaisesti.

34. Spatiaalinen uudelleen käyttö perustuu bilateraaliseen filtteröintiin käyttäen 7x7 ydintä sekä pikseleiden syvyysarvoja ja normaaleja.

$$
\begin{aligned}
w_n &= max(0, n(\mathbf{p}) \cdot n(\mathbf{q}))^{\sigma_n} \\
w_z &= e^{\left(\frac{|z(\mathbf{p})-z(\mathbf{q})|}{\sigma_z |\nabla z(\mathbf{p}) \cdot (\mathbf{p}-\mathbf{q})| + \varepsilon)}\right)} \\
w_l &= e^{\left(-\frac{|l_j(\mathbf{p})-l_j(\mathbf{q})|}{\sigma_l \sqrt{GC_{3x3}(Var(l_j(\mathbf{p})))} + \varepsilon}\right)},
\end{aligned}
\tag{4.38}
$$

missä puolestaan epsilonia $\varepsilon$ käytetään nollalla jakamisen välttämiseksi ja funktio $\nabla z(p)$ muuntaa syvyyspuskurin arvot lineaarisiksi arvoiksi näyttökoordinaatistossa.

Samoin kuin väriarvoista muodostetaan approksimaatio yhtälön 4.33 mukaisesti jokaisella tasolla seuraavan tason syötteeksi, tulee SVGF-menetelmässä lisäksi luminanssi arvojen varianssin estimaatti $Var(l_j(\mathbf{p}))$ päivittää jokaisella tasolla. Tason $j+1$ luminanssi arvojen varianssin estimaatti voidaan esittää muodossa

$$
Var(l_{j+1}(\mathbf{p})) = \frac{1}{k} \sum_{\mathbf{q} \in S} h_j(\mathbf{q})^2 \cdot W_j(\mathbf{p}, \mathbf{q})^2 \cdot Var(l_j(\mathbf{q})), \tag{4.39}
$$

missä normalisointitermi $k$ on

$$
k = (\sum_{\mathbf{q} \in S} h_j(\mathbf{q}) \cdot W_j(\mathbf{p}, \mathbf{q}))^2. \tag{4.40}
$$

Lukuun ottamatta muuttuneita painokertoimia, suoritetaan à-trous-aallokemuunnos ja muodostetaan rekonstruktio samalla tavalla kuin artikkelin (Dammertz ym. 2010) esittelemässä rajat ja reunat säilyttävässä à-trous-aallokemuunnoksessa[35]. À-Trous-aallokemuunnoksen jälkeen SVGF-menetelmässä suoritetaan ajallinen reunanpehmennys. Alkuperäisen SVGF-menetelmän kärsiessä haamukuvista[36], esitellään artikkelissa (Schied, Peters ja Dachsbacher 2018) adaptiivinen SVGF (A-SVGF), jossa ajallista uudelleenkäyttöä arvioidaan pikse-

35. Kuten useasti on tyypillistä, suoritetaan filtteröinti eli tässä tapauksessa à-trous-aallokemuunnos ilman tekstuureita tekstuureiden yksityiskohtien säilyttämiseksi, jonka lisäksi se helpottaa näytteiden ajallista uudelleenkäyttöä (Schied ym. 2017).

36. Menetelmässä käytetään kiinteää painokerrointa näytteiden uudelleenkäytölle, eikä täten vanhoja näytteitä hylätä, vaikka valaistus muuttuisi merkittävästikin.

leittäin, jolloin valaistuksen muuttuessa voidaan vanhentuneet näytteet hylätä. Menetelmän tarkempi kuvaaminen sivuutetaan tutkielmassa.

# 5 Laitteistokiihdytteinen säteenseuranta

Säteenseurannan ollessa pitkään tutkimuksen kohteena, oli se myös pitkään laskennallisesti liian raskas laajemmin käytettäväksi reaaliaikaisessa renderöinnissä. Läpimurto tapahtui kuitenkin vuonna 2018 NVIDIA:n esitellessä maailman ensimmäiset laitteistokiihdytteistä säteenseurantaa tukevat Turing-arkkitehtuurin näytönohjaimet (Corporation 2018), mikä on mahdollistanut säteenseurannan hyödyntämisen reaaliaikaisessa renderöinnissä esimerkiksi osana videopelejä. Seuraavissa aliluvuissa esitellään lyhyesti ensimmäinen laitteistokiihdytteistä säteenseurantaa tukeva näytönohjainarkkitehtuuri, NVIDIA:n Turing arkkitehtuuri, näytönohjainsäikeiden eriytyminen (engl. *divergence*), mikä on yksi näytönohjainohjelmoinnin merkittävimmistä haasteista, sekä laitteistokiihdytteinen säteenseuranta Vulkan API:ssa.

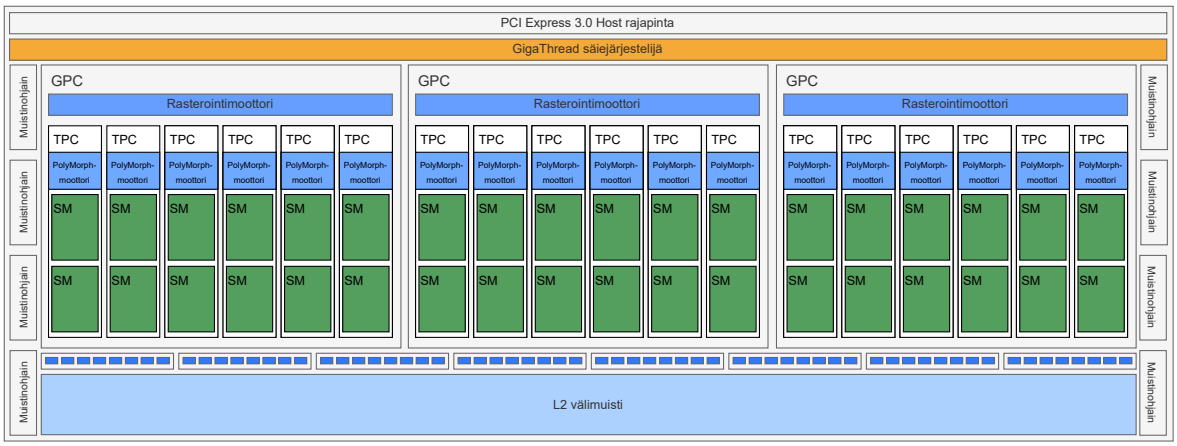
## 5.1 NVIDIA Turing-arkkitehtuuri

Kuten edellä mainittiin, tuli laitteistokiihdytteinen säteenseuranta ensimmäisen kerran mahdolliseksi NVIDIA:n julkaistua kuluttajaluokan Turing-arkkitehtuurin näytönohjaimet vuonna 2018. Verrattuna edeltävään Pascal-arkkitehtuuriin, Turing-arkkitehtuuri toi mukanaan useampia uudistuksia[1], joista tutkielman kannalta mielenkiintoisin on laitteistokiihdytteisen säteenseurannan mahdollistavat ensimmäisen[2] sukupolven RT-ytimet. RT-ytimet koostuvat kahdesta yksiköstä: rajaavientilavuuksien hierarkian (engl. *bounding volume hierarchy*) eli kiihdytysrakenteen läpikäyntiä kiihdyttävästä yksiköstä, sekä säde-kolmioleikkaavuustarkastelua kiihdyttävästä yksiköstä (Corporation 2018). Kiihdytysrakenteen tehtävänä on merkittävästi[3] vähentää säde-kolmioleikkaavuustarkasteluja muodostamalla kolmioista ja niitä rajaavista tilavuuksista puumainen hierarkia. Kiihdytysrakenteen muodostamisesta ja uudelleensovittamisesta vastaa ajuri (Corporation 2018).

1. Muita uudistuksia ovat toisen sukupolven Tensor-ytimet, ensimmäisen sukupolven Tensor-ytimien ilmestyessä ammattikäyttöön tarkoitettujen Volta-arkkitehtuurin näytönohjaimissa, sekä 32-bittisten liukuluku ja kokonaisluku operaatioiden rinnakkaissuorituksen mahdollistaminen (Corporation 2018).

2. Vuonna 2020 julkaistu NVIDIA:n Ampere-arkkitehtuuri esitteli kehittyneemmät toisen sukupolven RT-ytimet (Corporation 2021).

3. Esimerkiksi tasapainoisen binääripuun läpikäymisen kompleksisuusluokka on O(log(n)).

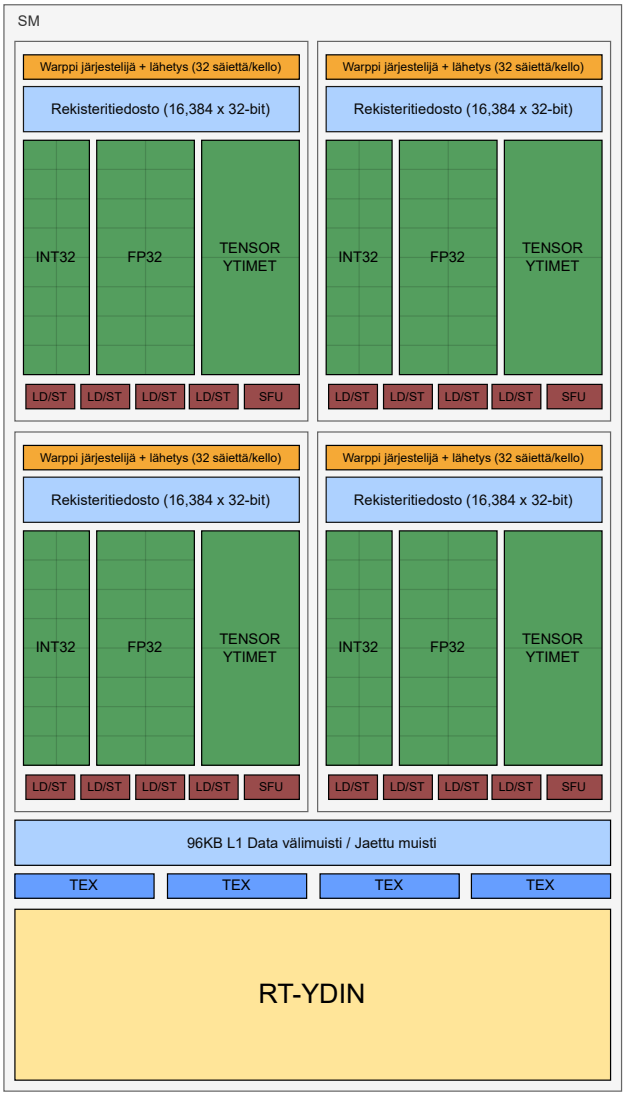



Kuvio 24. TU106 arkkitehtuuri.

Korkealla tasolla Turing-arkkitehtuurin, kuten myös sitä edeltävän Pascal-arkkitehtuurin ja Turing-arkkitehtuuria seuraavan Ampere-arkkitehtuurin näytönohjaimet koostuvat GPC (Graphics Processing Cluster) -yksiköistä, GigaThread-säiejärjestelijästä (engl. *GigaThread engine*), muistiyksiköistä ja muistiohjaimista (engl. *memory controller*), sekä I/O rajapinnoista. Korkean tason kuvaus Turing-arkkitehtuurin TU106-grafiikkapiiristä, johon tutkielman empiirisessä osuudessa käytetty TU106-200A-KA-A1-grafiikkapiiri pohjautuu, esitetään kuviossa 24. Edellä luetelluista korkean tason komponenteista näytönohjaimella tapahtuvasta laskennasta vastaavat GPC-yksiköt. GPC-yksiköt puolestaan koostuvat TPC (Texture Processing Clusters) yksiköistä ja rasterointimoottorista (engl. *raster engine*). Rasterointimoottori vastaa kuviossa 15 esitetyn loogisen grafiikkaliukuhihnan rasterointivaiheesta. TPC yksiköt puolestaan koostuvat PolyMorph-moottoreista (engl. *PolyMorph engine*), sekä näytönohjainlaskennan keskeisimmistä yksiköistä, SM-monisuorittimista (engl. *streaming multiprocessor*, *SM*). PolyMorph-moottori suorittaa muun muassa loogisen grafiikkaliukuhihnan tesselointivaiheen. SM-monisuorittimet vastaavat grafiikkaliukuhihnan varjostimien suorituksesta[4], sekä näytönohjaimella suoritettavasta yleisestä laskennasta (engl. *General-purpose computing on graphics processing units, GPGPU*).

Turing-arkkitehtuurissa SM-monisuorittimet voidaan edelleen jakaa neljään alilohkoon (engl.

4. Aiemmissa näytönohjainarkkitehtuureissa laitteistoon toteutettu kiinteä grafiikkaliukuhihna korvautui monikäyttöisillä SM-monisuorittimilla NVIDIA:n Tesla-arkkitehtuurin myötä (Lindholm ym. 2008), missä CUDA-ytimet (Compute Unified Device Architecture) suorittavat eri grafiikkaliukuhihnan vaiheiden grafiikkaoperaatioita, sekä yleistä laskentaa.




Kuvio 25. TU102/TU104/TU106 SM lohko.

| Ominaisuus | Arvo |
|---|---|
| CUDA-ytimet | 1920 |
| RT-ytimet | 30 |
| Tensor-ytimet | 240 |
| Kellotaajuus (OC) | 1365 (1710) MHz |
| Muisti | 6 GB |
| Gigasädettä/s | 5 |

Taulukko 1. GeForce RTX 2060 VENTUS XS 6G OC (TU106-200A-KA-A1) näytönohjaimen ominaisuudet. Seurattavien säteiden lukumäärä sekunnissa perustuu lähteeseen ("NVIDIA GeForce RTX 2060 Specification" 2022).

*sub partition*), jossa kussakin lohkossa on 16 kappaletta 32-bittisiin liukulukuihin erikoistunutta CUDA-ydintä ja 16 kappaletta 32-bittisiin kokonaislukuihin erikoistunutta ydintä, sekä kaksi useita laskentatarkkuuksia tukevaa ja matriisilaskentaan erikoistunutta Tensor-ydintä. Ytimien lisäksi alilohko sisältää rekisteritiedoston (engl. *register file*), sekä laskennan ajastamisesta ja suoritukseen lähettämisestä vastaavan warppi-järjestelijän (engl. *warp scheduler*) ja käskyjen lähetysyksikön (engl. *dispatch unit*). Kokonaisuudessaan Turing-arkkitehtuurin SM-monisuoritin sisältää 64 CUDA-ydintä[5], kahdeksan Tensor ydintä, sekä yhden RT-ytimen. Kuviossa 25 on esitelty Turing-arkkitehtuurin SM-monisuoritin ja taulukossa 1 puolestaan on esitelty luvun 6 empiirisessä osiossa käytetyn Turing-arkkitehtuurin näytönohjaimen ominaisuudet.

Turing-arkkitehtuurissa laitteistokiihdytteinen säteenseuranta alkaa SM-monisuorittimen lähettäessä sädekyselyn (engl. *ray query*) RT-ytimelle (Burgess 2020). RT-ytimellä säteenseuranta tapahtuu suorittaen säde-laatikkoleikkaavuustarkasteluja säteen matkatessa kiihdytysrakenteessa, jota seuraa säde-kolmioleikkaavuustarkasteluja, mikäli säde leikkaa hierarkian lehtisolmun rajaavan laatikon kanssa. Säteenseurannan jälkeen säteenseurannan tulos palautuu SM-monisuorittimelle. SM-monisuoritin vapautuu säteenseurannan ajaksi käytettäväksi

5. NVIDIA:n Ampere-arkkitehtuurissa kokonaislukuihin erikoistuneet ytimet kykenevät suorittamaan myös laskentaa liukuluvuilla, joka kasvattaa CUDA-ydinten määrän 64:stä 128:aan SM-monisuoritinta kohden Ampere-arkkitehtuurissa (Corporation 2021).

muuhun laskentaan (Burgess 2020).

## 5.2 Näytönohjainsäikeiden eriytyminen säteenseurannassa

Näytönohjaimen mahdollistaessa massiivisen rinnakkaislaskennan, tuo se mukaan myös omat haasteensa. Yksi merkittävä haaste näytönohjainohjelmoinnille on niin kutsuttu näytönohjainsäikeiden eriytyminen, joka vähentää näytönohjainlaskennan tehokkuutta. Pahimmassa tapauksessa säikeiden eriytyminen voi aiheuttaa kaikkien säikeiden peräkkäisen suorituksen rinnakkaisuuden sijasta. Tämän luvun tarkoituksena on lyhyesti esitellä säikeiden eriytyminen niin yleisesti kuin sen merkitys laitteistokiihdytteisessä säteenseurannassakin, sekä näytönohjaimen toiminta tarvittavilta osin. Luvussa käytetään NVIDIA/CUDA termistöä, jolloin warppi (engl. *warp*) tarkoittaa oleellisilta osin samaa kuin aaltorintama (engl. *wavefront*) ja säie (engl. *thread*) samaa kuin työ (engl. *work-item*) AMD/OpenCL termistössä.

Toisin kuin CPU:n, näytönohjainten säikeet eivät suoriudu yksin, vaan niin kutsuttuina warppeina[6] suorittaen rinnakkain saman käskyn kaikille warpin säikeille. Käsky voi olla osa grafiikkaliukuhihnan varjostinta tai näytönohjaimella suoritettavaa yleistä laskentaa. Näin ollen warpin voidaan kuvailla toimivan SIMT (Single Instruction, Multiple Threads)[7] mallin mukaisesti. Tästä seuraa se, että mikäli ainakin yhdestä saman warpin säikeestä suoritetaan esimerkiksi muistinouto, kuten tekstuurin luku, aiheutuu koko warpin sakkaaminen (engl. *stall*)[8], jolloin koko warppi asetetaan sivuun odottamaan muistinoudon valmistumista. Tällöin sakkautuneen warpin tilalle suoritukseen voidaan ottaa toinen warppi, mikäli käskyn suoritukseen kelvollisia (engl. *eligible*) warppeja on odottamassa vuoroaan. Kelvolliseksi warpiksi luetellaan aktiivisista warppeista eli SM-monisuorittimelle aikataulutetuista war-

6. Tyypillisesti NVIDIA:n näytönohjaimissa warppi käsittää 32 säiettä, kuin myös Turing-arkkitehtuurissa, jolloin Turing-arkkitehtuurin näytönohjaimissa yksittäisen warpin suorittamiseen tarvitaan kaksi sykliä (engl. *cycle*) (“NVIDIA Turing tuning guide” 2022).

7. Käytetään termiä SIMT termin SIMD sijasta korostaakseen sitä, että SIMT mallissa nimensä mukaisesti sama käsky suoritetaan eri säikeille, eikä yhden säikeen sisällä vektori muotoiselle datalle kuten tyypillisesti suoritetaan SIMD mallissa.

8. Muistiriippuvuuden ohella, muita syitä warpin sakkaamiselle ovat muun muassa riippuvuus suoritusjärjestyssä, käskyn nouto, käskyn suorittamiseen tarvittavan suoritusyksikön oleminen käytössä toisen warpin suorittamiseksi ja synkronisointi este (engl. *synchronization barrier*).

peista ne, jotka eivät ole sakkautuneena. Jos suorituksen kelvollisia warppeja ei ole saatavilla, kyseinen aikaikkuna käskyn suorittamiselle hukataan. Vaihtelemalla warppeja nopeasti, näytönohjain kykenee tehokkaasti piilottamaan latenssia, mikä johtaa suurempaan näytönohjaimen hyötykäyttöön, kun warppeja on saatavilla riittävästi[9].

Siinä, missä SIMT malli mahdollistaa nopean laskennan suorittamalla samanaikaisesti saman käskyn kaikille warpin säikeille[10], on mallin heikkoutena tilanne, jossa saman warpin säikeille tulisi suorittaa eri käsky. Tästä tilanteesta käytetään nimitystä säikeiden eriytyminen. Tyypillinen syy säikeiden eriytymiselle on suoritusvuon eriytyminen (engl. *control-flow divergence*), mikä johtuu haarauttavasta (engl. *branching*) käskystä suorituksessa, kuten if-else ehtolauseesta tai warpin säikeiden kesken epäyhtenäisestä silmukan päättävästä ehtolauseesta. Tällöin sen sijaan, että eri haarojen käskyt kyettäisiin suorittamaan rinnakkain, joudutaan eri haarojen käskyt suorittamaan peräkkäin. Peräkkäin suorittaminen voi tapahtua joko siten, että haarat suoritetaan yksittäin kokonaisuudessaan peräkkäin, kuten NVIDIA:n näytönohjaimissa ennen Volta-arkkitehtuuria, säilyttäen yhden käskyosoittimen (engl. *program counter*) warppia kohden tai limittäen eri haarojen käskyjä säilyttäen käskyosoittimen jokaista warpin säiettä kohden, kuten NVIDIA:n Volta-arkkitehtuurin ja sitä seuraavien NVIDIA:n Turing- ja Ampere-arkkitehtuurien näytönohjaimissa (Corporation 2017) (Corporation 2022b). Jälkimmäisestä tavasta NVIDIA käyttää termiä itsenäinen säikeiden aikataulutus (engl. *independent thread scheduling*). Molemmissa tavoissa haarautuminen johtaa siihen, että kokonaissuoritusaika on molempien haarojen suoritusaikojen summa, jonka lisäksi mahdollinen yleisrasite, mutta jälkimmäinen tapa antaa hienojakoisemman kontrollin säikeiden väliselle synkronisoinnille.

Artikkelissa (Pankratz ym. 2021) säikeiden eriytymiseen johtavat syyt laitteistokiihdytteisen säteenseurannan yhteydessä jaetaan suoritusvuon eriytymiseen, säteiden seuraamisen eriytymiseen ja varjostinkutsujen eriytymiseen. Laitteistokiihdytteisen säteenseurannan yhteydes-

9. Maksimaalisen hyötykäytön saavuttamiseksi jokaisella syklillä tulee olla saatavilla vähintään yksi kelvollinen warppi jokaista warppi-järjestelijää kohden. Saatavuuteen puolestaan vaikuttaa moni asia, kuten esimerkiksi latenssin kesto, aktiivisten warppien kokonaislukumäärä, sekä aktiivisten warppien kelpoisuuteen vaikuttavat tekijät.

10. Varsinaiseen käskyn suoritukseen lisäksi malli vähentää muun muassa käskyn noudosta, purusta, sekä suorituksen aikataulutuksesta aiheutuvaa yleisrasitetta (engl. *overhead*).

sä suoritusvuon eriytyminen voi aiheutua esimerkiksi polunseurantamenetelmää käytettäessä saman warpin sisältämien säikeiden seuraamien polkujen pituuksien epäyhtenäisyydestä, jolloin eri säikeet seuraavat eri määrän säteitä. Artikkelissa säteiden seurannan eriytymisellä tarkoitetaan tilannetta, jossa saman warpin säikeiden seurattavat säteet matkaavat kiihdytysrakenteessa eri ajan, jolloin warpin muut säikeet joutuvat odottamaan toisten säikeiden säteenseurannan valmistumista. Kuten seuraavassa luvussa esitellään, riippuu säteenseurannan liukuhihnalla suoritettava varjostin säteenseurannan lopputuloksesta, jolloin saman warpin säikeille voi tulla eri varjostin kutsuttavaksi. Artikkelissa tästä käytetään nimitystä varjostinkutsujen eriytyminen. Tutkielman kirjoittamisen aikaan julkaissuissa NVIDIA:n Ada Lovelace -arkkitehtuurin näytönohjaimissa on varjostinkutsujen eriytymisen vähentämiseksi toteutettu varjostinkutsujen uudelleenjärjesteleminen (engl. *shader execution reordering*, *SER*) (“GTC Sept 2022 Keynote with NVIDIA CEO Jensen Huang” 2022).

## 5.3 Laitteistokiihdytteinen säteenseuranta Vulkan API:ssa

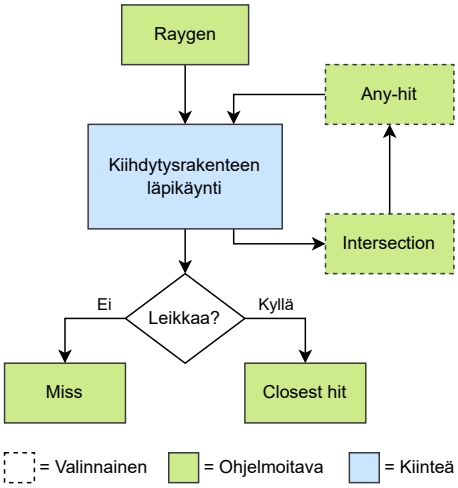
Vulkan API:lla laitteistokiihdytteinen säteenseuranta tuli mahdolliseksi Turing-arkkitehtuurin ja NVIDIA:n 411.63 ajurin julkaisujen myötä VK_NVX_ray_tracing[11] laajennoksen kautta käytettynä. Khronoksen julkaisemat näytönohjainvalmistajasta riippumattomat laajennokset[12] laitteistokiihdytteiselle säteenseurannalle tulivat vuoden 2020 joulukuussa Vulkan 1.2.162.0 SDK:n mukana (“Vulkan 1.2.162.0 SDK release notes” 2020). Tässä aliluvussa esitellään lyhyesti laitteistokiihdytteinen säteenseuranta Vulkan API:ssa, mutta samat pääpiirteet toistuvat myös Microsoftin DirectX grafiikka API:ssa (“Microsoft DirectX Raytracing Functional Spec” 2022). Täsmällisesti laitteistokiihdytteinen säteenseuranta Vulkan API:ssa on kuvattu spesifikaation (“Vulkan® 1.3.224 - A Specification” 2022) luvuissa 36-38 ja toteutuksen kannalta esimerkillisesti artikkelissa (Rusch, Bickford ja Subtil 2021, s. 213-255).

Laitteistokiihdytteinen säteenseuranta onnistuu Vulkan API:lla vain sitä tukevilla näytönohjaimilla[13]. Vulkan API:lla tuen ja laitteiston tukemat laitteistokiihdytteisen säteenseurannan

11. Vulkan API:ssa NV- ja NVX-liitteiset laajennokset ovat NVIDIA:n laatimia laajennoksia, joista NVX-liitteiset ovat kokeellisia, kun taas Khronos Groupin laatimat laajennokset tunnistaa KHR-liitteestä.

12. Ennen lopullista julkaisua VK_KHR_ray_tracing laajennos jaettiin erillisiin VK_KHR_acceleration_structure, VK_KHR_ray_tracing_pipeline ja VK_KHR_ray_query laajennoksiin.

13. Kirjoittamisen aikaan laitteistokiihdytteistä säteenseurantaa tukevat valitut NVIDIA:n Turing, Ampere ja




Kuvio 26. Laitteistokiihdytteisen säteenseurannan liukuhihna.

ominaisuudet voi tarkastaa kyselemällä fyysisen laitteen ominaisuuksia *vkGetPhysicalDeviceFeatures2* ja *vkGetPhysicalDeviceProperties2* funktioilla ketjuttamalla[14] VkPhysicalDeviceRayTracingPipelineFeaturesKHR, VkPhysicalDeviceRayTracingPipelinePropertiesKHR, VkPhysicalDeviceAccelerationStructureFeaturesKHR ja VkPhysicalDeviceAccelerationStructurePropertiesKHR tietueet. Taulukoissa 2 ja 3 esitetään edellä mainituilla kyselyillä täytetyt tietueet taulukossa 1 esitellylle RTX 2060 näytönohjaimelle ja GeForce Game Ready 512.15 - WHQL ajurille. Edellä mainittujen FeaturesKHR-päätteisten tietueiden avulla voi ottaa käyttöön halutut säteenseurannan ominaisuudet loogista laitetta (engl. *logical device*)[15] luotaessa.

Vulkan API tarjoaa kaksi tapaa suorittaa laitteistokiihdytteistä säteenseurantaa: säteenseurannan liukuhihnan ja sädekyselyt (engl. *ray query*). Sädekyselyt mahdollistavat säteenseurannan kutsumisen perinteisestä grafiikkaliukuhihnasta, kun säteenseurannan liukuhihna puolestaan esittelee uuden perinteisestä grafiikkaliukuhihnasta erillisen liukuhihnan. Säteenseurannan liukuhihna on kuvattu kuviossa 26. Säteenseurannan liukuhihna tuo mukaan vii-

Ada Lovelace arkkitehtuurien, AMD:n RDNA 2 ja RDNA 3 arkkitehtuurien, sekä Intel Arc™ tavaramerkin näytöohjaimet.

14. Osaan Vulkan API:n funktiosta parametrit viedään osoittimina tietueisiin (engl. *struct*), jotka sisältävät pNext-osoitin kentän, jota voidaan käyttää jatkamaan eli ketjuttamaan tietueita laajennosten tuomilla tietueilla.

15. Vulkan API:ssa laite-käsite jaetaan fyysiseen (engl. *physical device*) ja loogiseen, missä fyysinen laite nimensä mukaisesti edustaa fyysistä laitteistoa ja looginen laite on sovelluksen näkymä fyysiseen laitteeseen.

| VkPhysicalDeviceRayTracingPipelineFeaturesKHR | Arvo |
|---|---|
| rayTracingPipeline | tosi |
| rayTracingPipelineShaderGroupHandleCaptureReplay | epätosi |
| rayTracingPipelineShaderGroupHandleCaptureReplayMixed | epätosi |
| rayTracingPipelineTraceRaysIndirect | tosi |
| rayTraversalPrimitiveCulling | tosi |

| VkPhysicalDeviceRayTracingPipelinePropertiesKHR | Arvo |
|---|---|
| shaderGroupHandleSize | 32 |
| maxRayRecursionDepth | 31 |
| maxShaderGroupStride | 4096 |
| shaderGroupBaseAlignment | 64 |
| shaderGroupHandleCaptureReplaySize | 32 |
| maxRayDispatchInvocationCount | 1073741824 |
| shaderGroupHandleAlignment | 32 |
| maxRayHitAttributeSize | 32 |

Taulukko 2. Säteenseurannan liukuhihnaan liittyvät fyysisen laitteen ominaisuudet (NVIDIA RTX 2060, GeForce 512.15 WHQL ajuri).

si uutta varjostinta: säteenluonti- (engl. *raygen*), leikkauspiste- (engl. *intersection*), osuma- (engl. *any-hit*), ohi- (engl. *miss*) ja lähin osuma (engl. *closest hit*) -varjostimet. Näistä välttämättömiä ovat säteenluonti-, ohi- ja lähin osuma -varjostimet. Säteenluontivarjostin on säteenseurannan sisääntulopiste, jota kutsutaan, kun suoritetaan säteenseurannan aloittava *vkCmdTraceRaysKHR* tai *vkCmdTraceRaysIndirectKHR* komento. Säteenluontivarjostimen tehtävänä on luoda seurattavat säteet, jonka lisäksi säteenluontivarjostimesta tyypillisesti tehdään säteenseurannan lopputuloksen tallentaminen kuvapuskuriin. Säteenluontivarjostimen lisäksi uusia säteitä voidaan luoda ohi- ja lähin osuma -varjostimista. Ennalta määritellyille kolmioverkoille, oletus leikkauspistevarjostin toteuttaa säde-kolmioleikkaavuustarkastelun. Muille primitiiveille[16] tulee käyttäjän määritellä oma leikkauspistevarjostin. Leikkauspiste-

16. Muut primitiivit kattavat esimerkiksi erilaiset proseduraaliset geometriat, kuten täydellinen pallo, jolle

varjostimen löytäessä osuman säteen ja primitiivin välillä, tulee osumavarjostin kutsutuksi. Osumavarjostimessa osuma voidaan hyväksyä tai hylätä. Hyväksyttäessä säteen maksimaalinen seurattava etäisyys päivittyy osuman mukaisesti lyhyemmäksi. Tyypillinen käyttökohde osumavarjostimelle on läpinäkyvien materiaalien mallintaminen. Säde-primitiivi leikkaukset eivät aina esiinny etäisyys järjestyksessä, mistä syystä lähintä osumaa ylläpidetään sisäisesti seurannan ajan. Riippuen siitä, leikkasiko säde jonkun kiihdytysrakenteen primitiivin, tulee säteenseurannan päätyttyä kutsutuksi joko lähin osuma -varjostin[17] tai ohivarjostin. Varjostimien välillä dataa liikutellaan käyttäjän määrittelemässä tietueessa, sädekohtaisessa hyötykuormassa (engl. *ray payload*). Esimerkiksi sädekohtainen hyötykuorma voi sisältää väriarvon säteessä kulkeutuvalle valolle, johon kirjoitetaan lähin osuma -varjostimessa tai ohivarjostimessa. Säteen hyötykuorman suuruus vaikuttaa säteenseurannan suorituskykyyn (“NVIDIA RTX: Best Practices” 2019) (Sanzharov, Frolov ja Galaktionov 2020).

Toisin kuin kuviossa 15 kuvatussa perinteisessä grafiikkaliukuhihnassa, missä primitiivit matkaavat ennalta määrättyjen varjostimien ja vaiheiden lävitse, säteenseurannan liukuhihnassa varjostimia kutsutaan ennalta määräämättömästi riippuen säteenseurannan kulusta. Jotta oikea varjostin tulee kutsutuksi oikeassa tilanteessa, tulee käyttäjän muodostaa varjostinsidontataulukko (engl. *shader binding table*) ennen säteenseurannan liukuhihnan käyttöä. Varjostinsidontataulukko koostuu neljänlaisista tietuista: säteenluonti-, ohi-, kutsuttava- ja osumavarjostintietueista (engl. *hit*). Muistiosoitteet, jotka osoittavat tietueiden alkuihin, annetaan parametrina säteenseurannan kutsujen, *vkCmdTraceRaysKHR* tai *vkCmdTraceRaysIndirectKHR* komentojen, yhteydessä. Säteenluontivarjostimia on yksi säteenseurannan kutsua kohden eikä sitä erikseen indeksoida, vaan se tulee löytyä parametrina annetusta muistiosoitteesta. Ohivarjostimia voi olla useampia kutsua kohden, mutta yksi seurattavaa sädettä kohden. Käytettävä ohivarjostimen indeksi annetaan varjostimessa *traceRayEXT* kutsun parametrina. Kutsuttavat (engl. *callable*) varjostimet ovat liukuhihnan varjostimista poikkeavia varjostimia, joita voidaan kutsua säteenseurannan aikana. Kutsuttavan varjostimen suorittamiseksi, tulee varjostimen indeksi antaa *executeCallableEXT* kutsun parametrina. Osuma-

voidaan määritellä säde-palloleikkaavuustarkastelu säde-kolmioleikkaavuustarkastelun sijasta.

17. Lähin osuma -varjostimen kutsu on mahdollista myös ohittaa käyttäen gl_RayFlagsSkipClosestHitShaderEXT-lippubittiä varjostimessa *traceRayEXT* funktion yhteydessä.

18. Tutkielman kirjoittamisen aikaan NVIDIA ei tiettävästi ole julkaissut ajuria, joka tukisi kiihdytysrakenteen rakentamista CPU:lla.

| VkPhysicalDeviceAccelerationStructureFeaturesKHR | Arvo |
|---|---|
| accelerationStructure | tosi |
| accelerationStructureCaptureReplay | tosi |
| accelerationStructureIndirectBuild | epätosi |
| accelerationStructureHostCommands | epätosi [18] |
| descriptorBindingAccelerationStructureUpdateAfterBind | tosi |

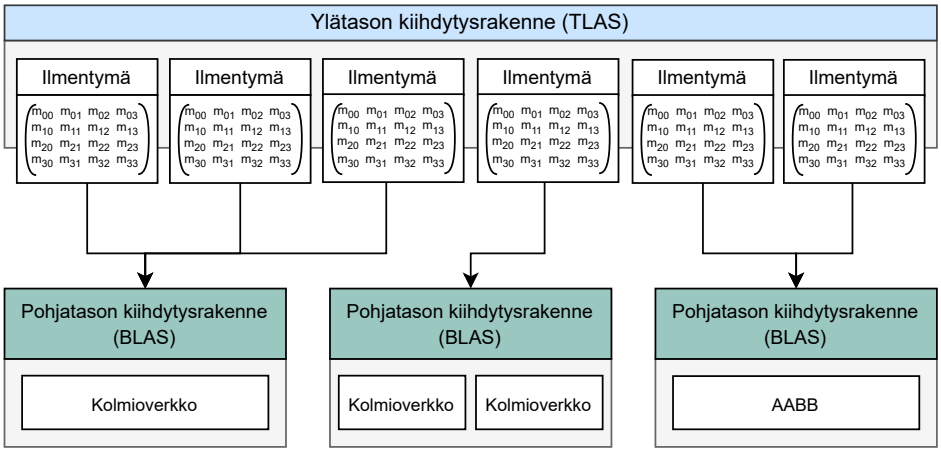
| VkPhysicalDeviceAccelerationStructurePropertiesKHR | Arvo |
|---|---|
| maxGeometryCount | 16777215 |
| maxInstanceCount | 16777215 |
| maxPrimitiveCount | 536870911 |
| maxPerStageDescriptorAccelerationStructures | 1048576 |
| maxPerStageDescriptorUpdateAfterBindAccelerationStructures | 1048576 |
| maxDescriptorSetAccelerationStructures | 1048576 |
| maxDescriptorSetUpdateAfterBindAccelerationStructures | 1048576 |
| minAccelerationStructureScratchOffsetAlignment | 128 |

Taulukko 3. Kiihdytysrakenteen fyysisen laitteen ominaisuudet (NVIDIA RTX 2060, GeForce 512.15 WHQL arjuri).

varjostintietue käsittää leikkauspiste-, osuma- ja lähin osuma -varjostimet. Varjostinsidontataulukon avulla käyttäjän tulee luoda assosiaatio kiihdytysrakenteen sisältämien geometrioiden ja osumavarjostintietueen välillä, jotta oikeat varjostimet tulevat kutsutuksi oikeille geometrioille[19].

Laitteistokiihdytteisen säteenseurannan keskeisimpänä käsitteenä voidaan pitää kiihdytysrakennetta, jota tarvitaan sekä sädekyselyissä, että säteenseurannan suorittamisessa liukuhihnan kautta. Vulkan API:ssa kuten myös DirectX API:ssa, kiihdytysrakenteen määritte-

19. Varjostinsidontataulukon indeksointia varten käyttäjä määrittelee jokaiselle ilmentymälle poikkeaman (engl. *offset*), jonka lisäksi varjostimen osoitteeseen vaikuttavat geometrian indeksi ilmentymässä, sekä taulukolle ja sen sisältämille tietueille määritellyt poikkeamat ja edistykset (engl. *stride*). Täsmällisesti indeksointi on kuvattu spesifikaation (“Vulkan® 1.3.224 - A Specification” 2022) luvussa 38.3.1.




Kuvio 27. Kiihdytysrakenne.

ly tapahtuu kahdessa tasossa: pohjatason kiihdytysrakenteina (engl. *bottom level acceleration structure*, *BLAS*) ja ylätason kiihdytysrakenteena (engl. *top level acceleration structure*, *TLAS*). Pohjatason kiihdytysrakenteet koostuvat yhdestä tai useammasta geometriasta, missä geometria käsittää joko kolmioverkon tai akselien suuntaisen rajaavan laatikon (engl. *axis-aligned bounding box*, *AABB*) käyttäjän määrittelemiä primitiivejä varten. Ylätason kiihdytysrakenne puolestaan koostuu pohjatason kiihdytysrakenteiden ilmentymistä. Kokonainen kiihdytysrakenne sisältää yhden ylätason kiihdytysrakenteen ja lukumäärältään enimmillään taulukossa 3 esitetyn *maxInstanceCount* vakion verran ilmentymiä. Kiihdytysrakenne korkealla tasolla esitetään kuviossa 27.

Kiihdytysrakenteen rakentamisen yhteydessä kiihdytysrakenteelle annetaan lippumuuttuja (engl. *flag*), jonka arvo vaikuttaa kiihdytysrakenteen rakentamiseen ja myöhempään käyttöön. VK_KHR_acceleration_structure laajennoksen tukemat lippubitit kiihdytysrakenteen rakentamiselle ovat lueteltu taulukossa 4. Erityisesti[20] ei-animoiduille eli staattisille pohjatason kiihdytysrakenteille mielenkiintoinen lippubitti on VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_COMPACTION_BIT_KHR, joka mahdollistaa kiihdytysrakenteen tiivistämisen pienentäen muistin käyttöä. Tiivistetyn kiihdytysrakenteen muistin käytön tarvetta ei tunneta ennen kiihdytysrakenteen rakentamista, vaan se pitää lukea vasta rakentamisen jälkeen[21]. Tiedettäessä tiivistetyn kiihdytysrakenteen todellinen muistin käytön tarve,

20. Usein pohjatason kiihdytysrakenteet vaativat yhteensä enemmän muistia kuin ylätason kiihdytysrakenne.
21. Tämä onnistuu ainoastaan kiihdytysrakenteilla, joille on VK_BUILD_ACCELERATION_STRUCTURE_

voidaan kiihdytysrakenne kopioida uuteen muistiosoitteeseen pienempään tilaan. Kiihdytysrakenteen muuttuessa kiihdytysrakenne voidaan joko rakentaa uudelleen tai päivittää[22]. Kiihdytysrakenteen päivittämisessä on ehtonsa, kuten esimerkiksi se, ettei primitiivien tai ilmentymien lukumäärää voi muuttaa, vaan ainoastaan niiden muunnoksia (engl. *transformation*). Täsmälliset ehdot kiihdytysrakenteen päivittämiselle on annettu spesifikaation ("Vulkan® 1.3.224 - A Specification" 2022) luvussa 36.1.4.

| Lippubitti |
|---|
| VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_COMPACTION_BIT_KHR |
| VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_UPDATE_BIT_KHR |
| VK_BUILD_ACCELERATION_STRUCTURE_PREFER_FAST_TRACE_BIT_KHR |
| VK_BUILD_ACCELERATION_STRUCTURE_PREFER_FAST_BUILD_BIT_KHR |
| VK_BUILD_ACCELERATION_STRUCTURE_LOW_MEMORY_BIT_KHR |

Taulukko 4. VK_KHR_acceleration_structure laajennoksen tukemat lippubitit kiihdytysrakenteen rakentamiselle.

ALLOW_COMPACTION_BIT_KHR-lippubitti asetettu. Todellinen muistin käytön tarpeen lukeminen tapahtuu käyttäen asynkronisia Vulkan API:n kyselyjä (engl. *queries*).

22. Päivittäminen vaatii VK_BUILD_ACCELERATION_STRUCTURE_ALLOW_UPDATE_BIT_KHR-lippubitin asettamisen kiihdytysrakenteen *rakentamisen* yhteydessä.

# 6 Laitteistokiihdytteisen säteenseurannan suorituskyky

Tutkielman yhtenä tutkimuskysymyksenä on tarkemmin tutkia laitteistokiihdytteisen säteenseurannan suorituskykyä ja siihen vaikuttavia tekijöitä, sekä pyrkiä tarkentamaan kuvaa siitä, missä määrin laitteistokiihdytteistä säteenseurantaa kyetään hyödyntämään reaaliaikaisessa renderöinnissä. Laitteistokiihdytteisen säteenseurannan suorituskyvyn näkökulmasta sen keskiössä ovat kiihdytysrakenne ja varjostinsidontataulukko[1], joista kiihdytysrakenne on osana säteenseurantaa sekä liukuhihnan, että sädekyselyjen kautta suoritettuna ja varjostinsidontataulukko yksinomaan liukuhihnan kautta suoritettuna. Sen lisäksi, että kiihdytysrakenne vaaditaan aina säteenseurannassa, tarjoaa kiihdytysrakenne sisällöllisen vaihtelun lisäksi paljon variaatiota rakenteiden määrittelyn kautta, mistä syystä tutkielman kokeellinen osio on rajattu koskemaan kiihdytysrakenteiden ominaisuuksien ja määrittelyjen vaikutusta säteenseurannan suorituskykyyn. Seuraavien alilukujen tehtävänä on antaa täsmällinen kuva korjärjestelmästä, koeasetelmista ja mittausmenetelmistä, sekä vastata tutkimuskysymykseen esitettyjen tulosten pohjalta.

## 6.1 Koeympäristö ja koeasetelmat

Tutkielman kokeellinen osuus suoritetaan Vulkan API:lla toteutetussa koeympäristössä ja taulukossa 5 esitetyllä laitteistolla. Koeympäristössä säteenseuranta suoritetaan säteenseurannan liukuhihnan kautta. Kokeissa tutkitaan kiihdytysrakenteen ominaisuuksien vaikutusta laitteistokiihdytteisen säteenseurannan suorituskykyyn säteiden seuraamisen, rakenteen rakentamiseen kuluneen ajan ja muistin käytön osalta. Koeympäristössä toteutetaan kolme koeasetelmaa: erikseen koeasetelmat, joissa tutkitaan pohja- ja ylätason kiihdytysrakenteiden ominaisuuksien ja määritysten vaikutusta muistin käyttöön, rakenteiden rakentamiseen kuluneeseen aikaan ja säteenseurannan suorituskykyyn, sekä koeasetelma, jossa tutkitaan ylätason kiihdytysrakenteen päivittämisen vaikutus säteenseurannan suorituskykyyn.

1. Varjostinsidontataulukon vaikutus säteenseurannan suorituskykyyn on oletettavasti riippuvainen varjostimien sisällöstä, sekä varjostinkutsujen eriytymisestä, johon puolestaan vaikuttavat varjostimien lukumäärä, sekä varjostimien ja samalla seurattavien säteiden suuntien jakautuminen virtuaalimaailmassa.

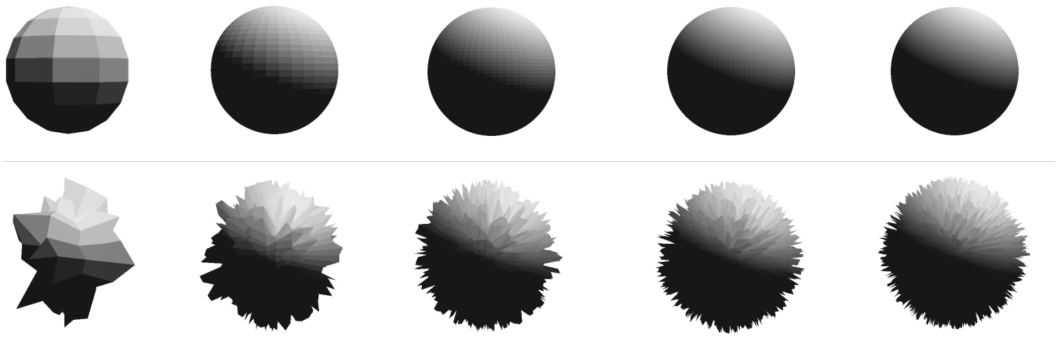
| Laite/ohjelmisto |
|---|
| GeForce RTX 2060 VENTUS XS 6G OC |
| GeForce Game Ready 512.15 - WHQL |
| AMD 3600 |
| 16 Gt 3600 MHz |
| Vulkan API 1.3.194 |
| Windows 10 |

Taulukko 5. Kokeissa käytetty laitteisto ja ympäristö. Näytönohjaimen tarkemmat ominaisuudet esitetään taulukossa 1.

Kokeissa käytetään kolmenlaisia kolmioverkkoja: eri tarkkuudella kuvattuja yksikköpalloja ja näiden pohjalta luotuja muokattuja yksikköpalloja, sekä kulmista pyöristettyä kuutiota. Yksikköpalloista, sekä muokatuista yksikköpalloista kokeissa käytetään 200:aa eri tarkkuudella kuvattua geometriaa, joiden kolmioiden lukumäärä vaihtelee 180:stä 23762:een. Muokatuissa yksikköpalloissa yksittäisten verteksien etäisyys pallon keskipisteestä vaihtelee tasajakauman mukaisesti välillä 0,5 - 1,5. Säteenseurannan suorituskykyä mitattaessa muokattuja yksikköpalloja kutistetaan 30 %, jolloin niiden peittävyys on likimäin sama kuin muokkaamattomissa yksikköpalloissa. Kuviossa 28 esitetään muutama käytetyistä yksikköpalloista ja muokatuista yksikköpalloista. Kulmista pyöristetty kuutio puolestaan koostuu 396:sta kolmiosta ja se esitetään esimerkiksi kuviossa 11. Kuution särmän pituus on 2 ja sille muodostetun minimaalisen rajaavan pallon säde on noin 1,6. Kaikkien kolmioverkkojen verteksit tallennetaan 32 tavuisina[2] ilman indeksointia näytönohjaimen erilliseen muistiin.

Kokeiden piirtoalue on kooltaan 2000 x 1000 pikseliä ja säteenseurannan suorituskyvyn mittauksissa seurataan neljä sädettä pikseliä kohden, jolloin seurattavien säteiden kokonaismäärä *vkCmdTraceRaysKHR* komentoa kohden on 8 miljoonaa. Seurattavien primäärisäteiden enimmäispituudeksi asetetaan 10000 ja varjosäteiden enimmäispituutena käytetään säteen lähtöpisteen ja lähellä virtuaalikameraa sijaitsevan pistemäisen valonlähteen välistä etäisyyttä. Sekä primääri-, että varjosäteet luodaan säteenluontivarjostimessa, jolloin enimmäisrekur-

2. Yksittäinen verteksi koostuu sijainnista (12 tavua), normaalista (12 tavua) ja UV-koordinaateista (8 tavua).



Kuvio 28. Kokeissa käytettyjä yksikköpalloja (ylhäällä) ja muokattuja yksikköpalloja (alhaalla). Kolmioiden lukumäärät geometrioissa ovat 180, 2380, 7080, 14280 ja 23544 vasemmalta oikealle.

siosyvyys eli maxPipelineRayRecursionDepth asetetaan yhteen[3]. Säteenluontivarjostimessa säteenseurannan aloittavan *traceRayEXT* kutsun yhteydessä käytetään gl_RayFlagsOpaqueEXT-lippubittiä[4] sekä primääri-, että varjosäteissa, jonka lisäksi varjosäteissä käytetään gl_RayFlagsTerminateOnFirstHitEXT ja gl_RayFlagsSkipClosestHitShaderEXT-lippubittejä[5].

Sädekohtainen hyötykuorma primäärisäteillä koostuu yksittäisestä RGB-väriarvosta ja varjosäteillä yksittäisestä totuusarvosta. Primäärisäteitä seurattaessa lähin osuma -varjostimessa suoritetaan minimaalinen valaistuslaskenta ratkaisemalla suoran valaistuksen yhtälö 2.24 yksittäisestä pistemäisestä valonlähteestä ilman näkyvyystermiä ja olettaen pinnan olevan Lambertin diffuusi. Tämä edellyttää säteen leikkaaman kolmion verteksien, kuten myös jokaiselle ilmentymälle määrätyn yksittäisen värin, noutamista muistista[6]. Varjosäteitä seurattaessa suoran valaistuksen yhtälöstä lisäksi ratkaistaan näkyvyystermi lähes pistemäisestä valonlähteestä. Varjosäteitä seurattaessa ilmentymille ei ole määrätty yksilöllisiä värejä, vaan muistis-

3. Rekursiosyvyys vaikuttaa siihen, kuinka suuri määrä pinomuistia varataan suoritettaessa *vkCmdTraceRaysKHR* komento (“NVIDIA RTX: Best Practices” 2019).

4. Näin ollen kaikki ilmentymät käsitellään läpikuultamattomina, jolloin osumavarjostimia ei tulla kutsumaan.

5. Tällöin varjosäteen seuranta lopetetaan minkä tahansa leikkaavuuden löydyttyä, eikä lähin osuma -varjostinta tulla kutsumaan. Ohivarjostinta käytetään tallentamaan tieto siitä, että pinta-ala-alkio ei ole varjossa.

6. Jokaiselle ilmentymälle tallennetaan verteksi- ja materiaalipuskurin osoitteet, joista ensimmäistä käytetään säteen ja primitiivin välisen leikkauspisteen normaalin ja sijainnin muodostamiseen ja jälkimmäistä ilmentymän värin määrittämiseen.

| Lippumuuttujan arvo[7] | Lyhenne | Koeasetelma |
|---|---|---|
| - | - | 1 |
| ALLOW_UPDATE_BIT_KHR | ALLOW_UPDATE | 1 |
| PREFER_FAST_TRACE_BIT_KHR | FAST_TRACE | 1 |
| PREFER_FAST_BUILD_BIT_KHR | FAST_BUILD | 1 |
| LOW_MEMORY_BIT_KHR | LOW_MEMORY | 1 |
| ALLOW_COMPACTION_BIT_KHR | + | 1 |
| ALLOW_COMPACTION_BIT_KHR + ALLOW_UPDATE_BIT_KHR | + ALLOW_UPDATE | 1, **2**, **3** |
| ALLOW_COMPACTION_BIT_KHR + PREFER_FAST_TRACE_BIT_KHR | + FAST_TRACE | 1 |
| ALLOW_COMPACTION_BIT_KHR + PREFER_FAST_BUILD_BIT_KHR | + FAST_BUILD | 1 |
| ALLOW_COMPACTION_BIT_KHR + LOW_MEMORY_BIT_KHR | + LOW_MEMORY | 1 |

Taulukko 6. Pohjatason kiihdytysrakenteilla käytettävät VkBuildAccelerationStructureFlagsKHR lippumuuttujan arvot, tuloksissa käytettävät lyhenteet, sekä koeasetelmat, joissa lippumuuttujan arvoa on käytetty. Lihavointi tarkoittaa, että lippumuuttuja on vakioitu koeasetelmassa.

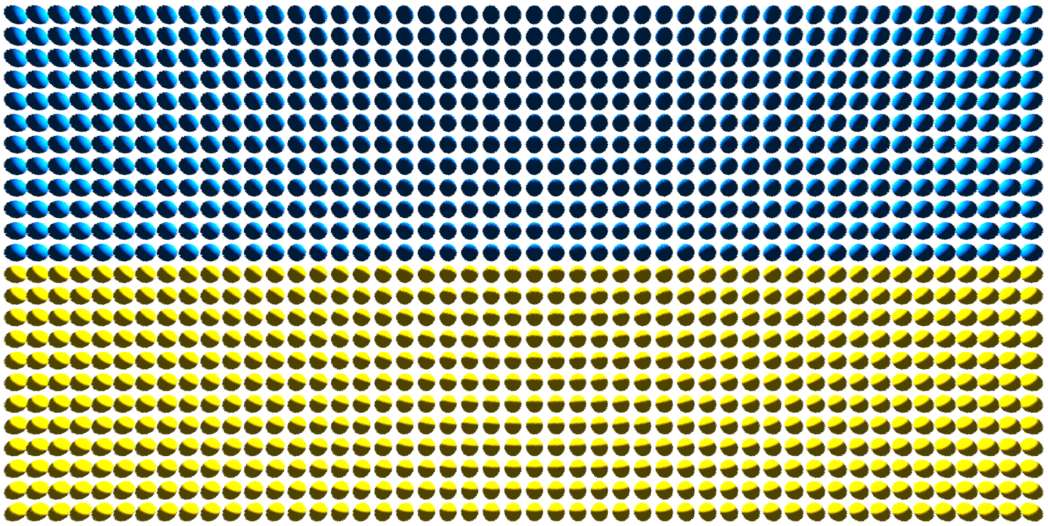
ta luetaan ainoastaan G-puskuriin tallennetut pikselien sijainnit ja normaalit. Sekä primääri-, että varjosäteitä seurattaessa epäsuoraa valaistusta approksimoidaan vakiolla.

Ensimmäisessä koeasetelmassa tarkoituksena on tutkia pohjatason kiihdytysrakenteen ominaisuuksien ja määrittelyn vaikutusta säteenseurannan suorituskykyyn, muistin käyttöön ja kiihdytysrakenteen rakentamiseen kuluvaan aikaan. Koeasetelmassa suoritettavissa kokeissa kolmioverkkoina käytetään yksikköpalloja ja muokattuja yksikköpalloja. Kokeet suoritetaan erikseen taulukossa 6 esitetyille VkBuildAccelerationStructureFlagsKHR-lippumuuttujan[8]

7. Lippumuuttujan arvot esitetään ilman etuliitettä VK_BUILD_ACCELERATION_STRUCTURE_-.

8. VkBuildAccelerationStructureFlagsKHR-lippumuuttuja on osa VkAccelerationStructureBuildGeometryInfoKHR tietuetta, johon annetaan osoitin *vkCmdBuildAccelerationStructuresKHR* kutsussa. Vulkan-spesifikaatio takaa ALLOW_COMPACTION_BIT_KHR ja ALLOW_UPDATE_BIT_KHR-lippubittien toi-

arvoille, jonka lisäksi kokeet suoritetaan erikseen sekä tiivistetyille[9], että ei-tiivistetyille eli alustaville pohjatason kiihdytysrakenteille. Pohjatason kiihdytysrakenteen muistin käyttöä ja rakentamiseen kuluvaa aikaa mittaavissa kokeissa luodaan koeympäristössä yksi yhden kolmioverkon sisältävä pohjatason kiihdytysrakenne. Säteenseurannan suorituskykyä mittaavissa kokeissa puolestaan luodaan yksi ylätason kiihdytysrakenne, joka koostuu virtuaalikameran näkymään säännöllisesti sijoitettavista edellä kuvatun pohjatason kiihdytysrakenteen 1152:sta ilmentymästä, kuten on havainnollistettu kuviossa 29. Näin ollen säteenseurannan suorituskykyä mittaavissa kokeissa kiihdytysrakenteen kolmioiden kokonaismäärä vaihtelee kolmioverkosta riippuen noin 207 tuhannesta noin 27 miljoonaan kolmioon.



Kuvio 29. Virtuaalikameran näkymä säteenseurannan suorituskykyä mittaavassa kokeessa ensimmäisessä koeasetelmassa.

Toisessa koeasetelmassa tarkoituksena on tutkia ylätason kiihdytysrakenteen ominaisuuksien ja määrittelyn vaikutusta säteenseurannan suorituskykyyn, muistin käyttöön ja kiihdytysrakenteen rakentamiseen kuluvaan aikaan. Koeasetelmassa luodaan yksi pohjatason kiihdytysrakenne, joka koostuu yksittäisestä kolmioverkosta. Kolmioverkkona käytetään kulmista pyöristettyä kuutiota. Pohjatason kiihdytysrakenne rakennetaan käyttäen ALLOW_UPDATE_BIT_KHR ja ALLOW_COMPACTION_BIT_KHR-lippubittejä, jonka jälkeen ra-

minnallisuuden, kun taas muiden lippubittien toiminnallisuus riippuu näytönohjaimen ajurista.

9. Tiivistäminen on mahdollista vain niille lippumuuttujan arvoille, joissa ALLOW_COMPACTION_BIT_KHR-lippubitti on asetettu.

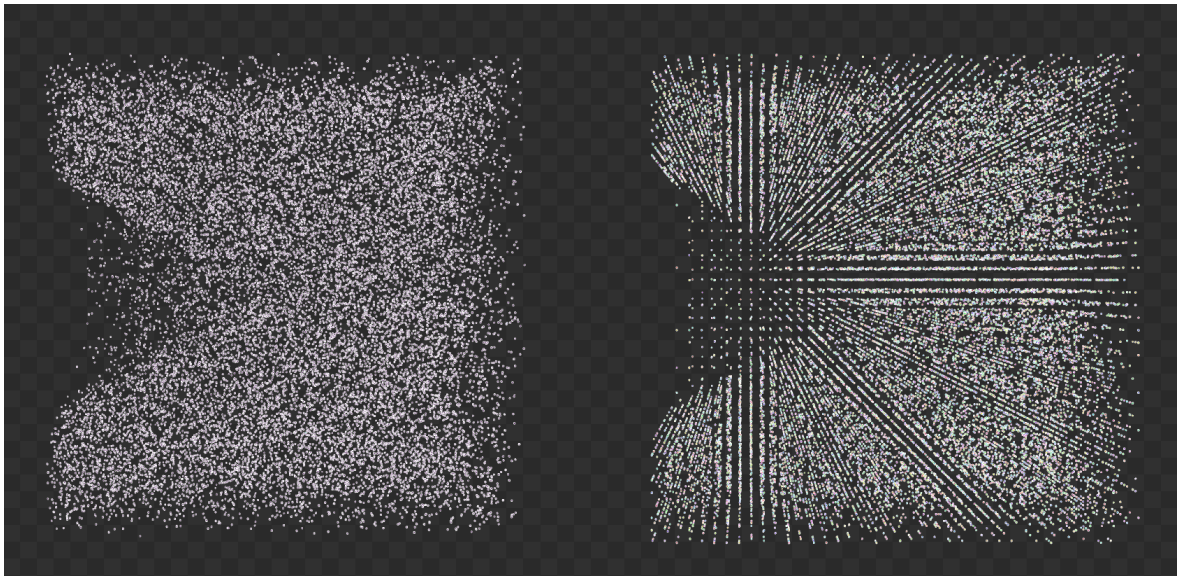
kenne tiivistetään. Koeasetelmassa pohjatason kiihdytysrakenteesta luodaan vaiheittain 1-20000 satunnaisesti kiertynyttä ilmentymää muodostaen ylätason kiihdytysrakenteen. Ilmentymät sijoitetaan suuruudeltaan 1000 x 1000 x 1000 kokoiseen virtuaalimaailmaan kahdella tavalla. Ensimmäisessä tavassa ilmentymät sijoitetaan virtuaalimaailmaan satunnaisiin sijainteihin, mutta estetään ilmentymien sijoittuminen päällekkäin[10]. Toisessa tavassa virtuaalimaailma jaetaan säännöllisin välein 64000 sijaintiin, joihin ilmentymiä sijoitetaan satunnaisessa järjestyksessä siten, ettei samaan sijaintiin sijoiteta kahta ilmentymää. Lisäksi molemmissa tavoissa estetään ilmentymien sijoittuminen virtuaalimaailman keskellä sijaitsevan virtuaalikameran näkymään suorittamalla leikkaavuustarkastelu virtuaalikameran katkaistun pyramidin ja kuution minimaalisen rajaavan pallon välillä. Virtuaalikameran katkaistun pyramidin kauempi taso (engl. *far plane*) ulotetaan virtuaalimaailman ulkopuolelle, jolloin virtuaalikameran näkymässä ei näy yhtään kuutiota. Säännöllinen ja epäsäännöllinen sijoittelu on havainnollistettu kuviossa 30. Virtuaalikameran katkaistu pyramidi näkyy ilmentymien harventumana kuviossa esitetyissä ylätason kiihdytysrakenteissa. Koeasetelmassa ohivarjostimesta palautetaan vakiomuotoinen arvo. Koeasetelmassa kokeet suoritetaan erikseen käyttämällä ylätason kiihdytysrakenteelle taulukossa 7 esitettyjä VkBuildAccelerationStructureFlagsKHR-lippumuuttujan arvoja.

| Lippumuuttujan arvo[11] | Lyhenne | Koeasetelma |
|---|---|---|
| ALLOW_COMPACTION_BIT_KHR | ALLOW_COMPACTION | 2 |
| ALLOW_UPDATE_BIT_KHR | ALLOW_UPDATE | 2, **1**, **3** |
| PREFER_FAST_TRACE_BIT_KHR | FAST_TRACE | 2 |
| PREFER_FAST_BUILD_BIT_KHR | FAST_BUILD | 2 |
| LOW_MEMORY_BIT_KHR | LOW_MEMORY | 2 |

Taulukko 7. Ylätason kiihdytysrakenteilla käytettävät VkBuildAccelerationStructureFlagsKHR-lippumuuttujan arvot, tuloksissa käytettävät lyhenteet, sekä koeasetelmat, joissa lippumuuttujan arvoa on käytetty. Lihavointi tarkoittaa, että lippumuuttuja on vakioitu koeasetelmassa.

10. Koeympäristössä ilmentymien päällekkäin sijoittuminen estetään perustuen kulmista pyöristetyn kuution minimaalisen rajaavan pallon halkaisijaan.

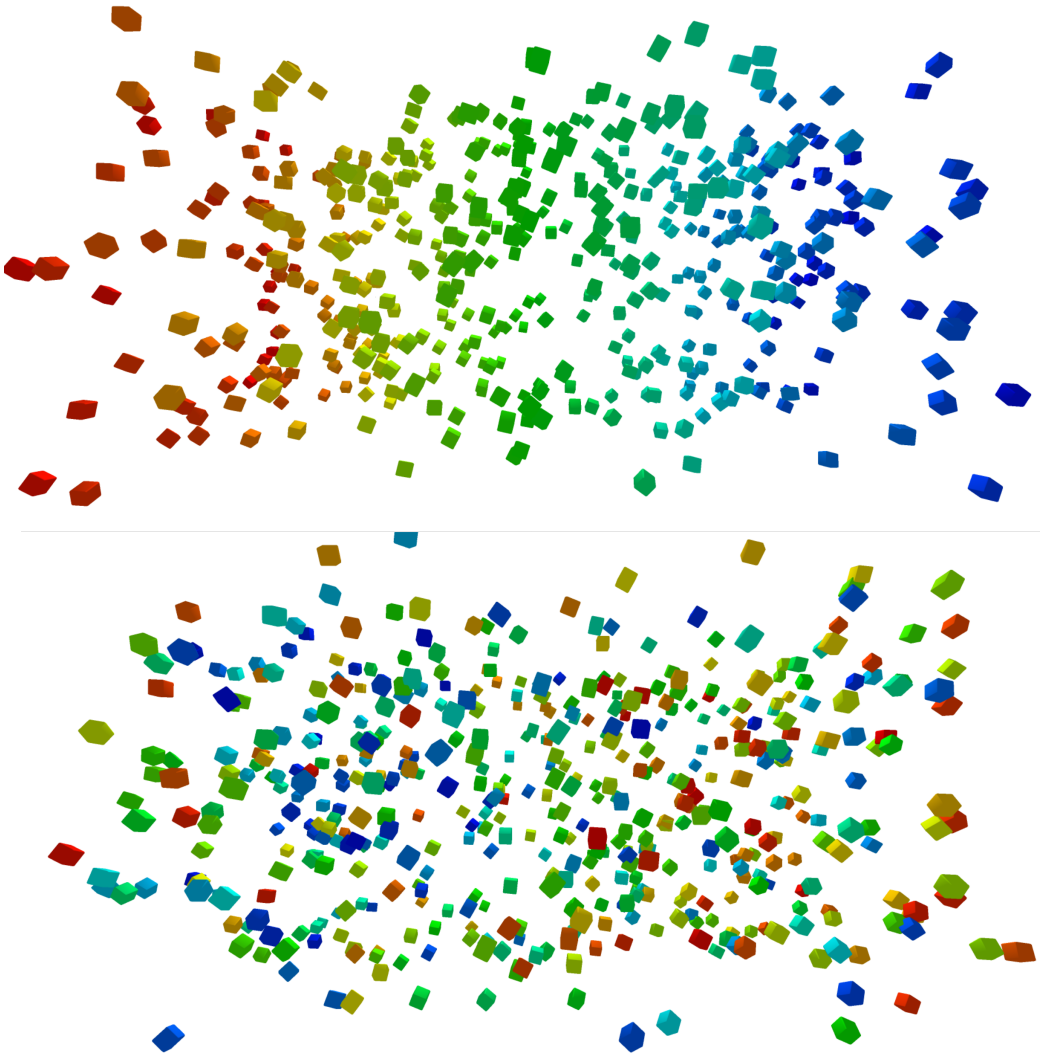
11. Lippumuuttujan arvot esitetään ilman etuliitettä VK_BUILD_ACCELERATION_STRUCTURE_-.



Kuvio 30. NVIDIA Nsight™ Graphics työkalulla tuotettu näkymä toisessa koeasetelmassa suoritettavan kokeen lopusta epäsäännöllisessa ilmentymien sijoittelulla (vasemmalla) ja säännöllisellä sijoittelulla (oikealla).

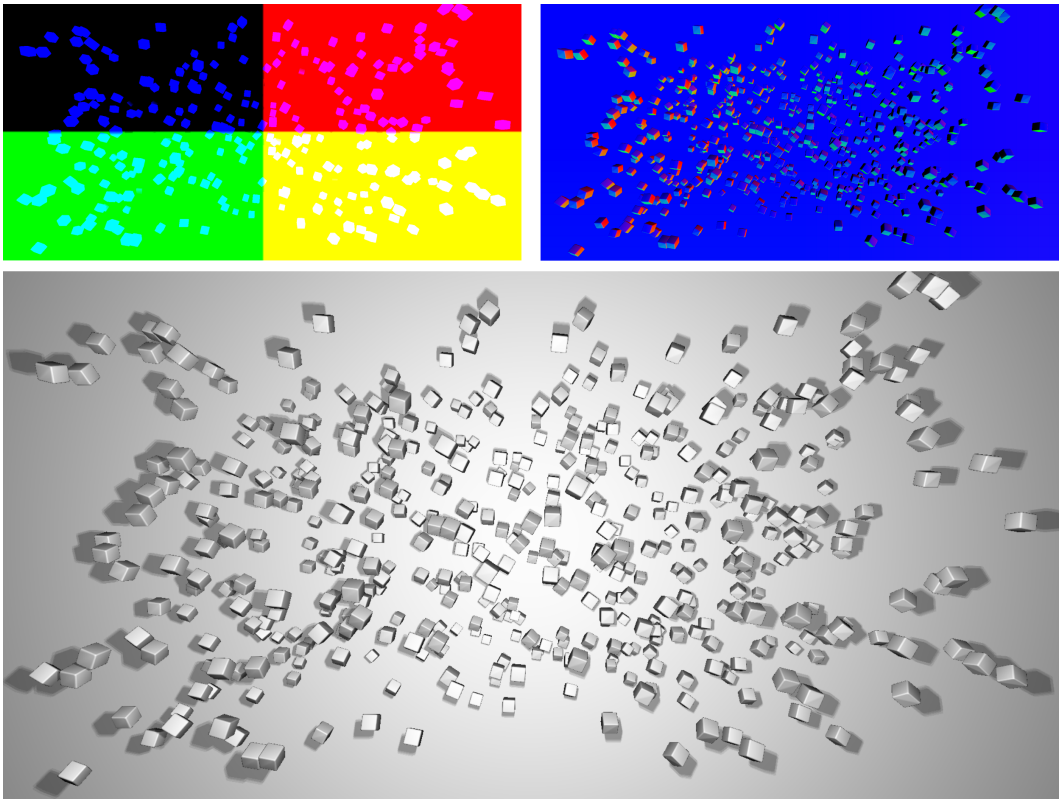
Kolmannessa koeasetelmassa tarkoituksena on tutkia ylätason kiihdytysrakenteen päivittämisen vaikutusta säteenseurannan suorituskykyyn. Koeasetelmassa käytetään samaa pohjatason kiihdytysrakennetta kuin edellä kuvatussa toisessa koeasetelmassa. Koeasetelmassa pohjatason kiihdytysrakenteesta luodaan 500 satunnaisesti kiertynyttä ilmentymää, jotka muodostavat ylätason kiihdytysrakenteen. Näin ollen kolmioiden kokonaismäärä kiihdytysrakenteessa on 198 tuhatta[12]. Ilmentymät sijoitetaan satunnaisesti suuruudeltaan 120 x 60 x 60 alueelle ja virtuaalikameran näkymä sovitetaan vastaamaan alueen rajoja siten, että kaikki ilmentyvät näkyvät virtuaalikamerassa. Kokeessa tietty osa ilmentymistä asetetaan liikkeeseen muiden ilmentymien pysyessä paikallaan. Liikkeeseen asetettujen ilmentymien suuntavektori valitaan satunnaisesti, mutta ilmentymän osuessa alueen rajoihin, vaihdetaan ilmentymän suunta heijastuslain mukaisesti. Näin ollen kaikki ilmentymät pysyvät virtuaalinäkymässä kokeiden ajan. Kokeet suoritetaan erikseen kahdella nopeudella ja eri osuudella liikkuvia ilmentymiä, jonka lisäksi säteenseurannan suorituskykyä mitataan erikseen primääri- ja varjosäteiden osalta. Kuviossa 31 esitetään virtuaalikameran näkymä primääri-

12. Koeasetelma toistetaan 1000:lle ja 2000:lle ilmentymälle, joissa kolmioiden kokonaismäärät ovat 396 ja 792 tuhatta.



Kuvio 31. Virtuaalikameran näkymä primäärisäteiden suorituskykyä mittaavassa kokeessa kokeen alussa (yläpuolella) ja kokeen lopussa (alapuolella) kolmannessa koeasetelmassa.

säteiden suorituskykyä mittaavasta kokeesta ennen ilmentymien liikkeelle asettamista, sekä kokeen lopusta. Varjosäteiden suorituskyvyn mittaamiseksi virtuaalimaailman taakse asetetaan tasomainen geometria, jotta seurattavien säteiden lukumäärä säilyy vakiona. Kuviossa 32 esitetään virtuaalikameran näkymä varjosäteiden suorituskykyä mittaavasta kokeesta, sekä vastaava G-puskurin sisältö.



Kuvio 32. Virtuaalikameran näkymä varjosäteiden suorituskykyä mittaavassa kokeessa (alapuolella) ja G-puskurin sisältävät pikseleiden sijainnit ja normaalit tallentavat kuvapuskurit (yläpuolella) kolmannessa koeasetelmassa.

## 6.2 Mittausmenetelmät

Kokeissa pyritään täsmällisesti mittaamaan laitteistotason suorituskykyä minimoimalla valaistuslaskentaan kuluva aika, sekä satunnaisuuden ja järjestelmän muun kuorman aiheuttama vaihtelu. Kokeissa minimoidaan varjostinsidontataulukon vaikutus tuloksiin käyttämällä samaa osumavarjostintietuetta kaikille ilmentymille, jolloin minimoidaan varjostinkutsujen eriytyminen. Myös varjostinsidontataulukon sisältämät varjostimet pidetään yksinkertaisina sisältäen vain vähän muistista lukuja. Koeympäristössä kaikki muistin varaaminen tapahtuu ennen mittausten aloittamista koeympäristön käynnistyessä. Koeasetelmien minimaalisuuden takia säteenseurannan suorituskykyä mittaavien kokeiden tarkoitus ei ole antaa täsmällistä kuvaa säteenseurannan suorituskyvystä useimmissa reaaliaikaisissa sovelluksissa, vaan odotettavissa olevaa ylärajaa ja siihen vaikuttavia tekijöitä.

Koska koeympäristö ei ole eristetty ja osassa kokeita hyödynnetään satunnaisuutta, kerätään tulosten luotettavuuden parantamiseksi mittauksia pidemmiltä aikajaksoilta ja useammilta toistoilta vaihdellen satunnaislukugeneraattorin siemenen (engl. *seed*) arvoa. Tulokset esitetään näin saatujen mittausten keskiarvoina. Ensimmäisessä koeasetelmassa pohjatason kiihdytysrakenteen rakentamiseen kulunutta aikaa mitataan suorittaen kymmenen toistoa jokaiselle geometrialle ja lippumuuttujan arvolle erikseen. Jokaisella toistolla suoritetaan *vkCmdBuildAccelerationStructuresKHR* kutsu ja jokaisella kutsulla rakennetaan 100 kertaa sama pohjatason kiihdytysrakenne samalla lippumuuttujan arvolla, jolloin tulokset esitetään tuhannen mittauksen keskiarvoina[13]. Samassa koeasetelmassa primäärisäteiden seuraamisen suorituskyvyn mittaukset tehdään 2000 peräkkäiseltä kehykseltä jokaiselle geometrialle ja lippumuuttujan arvolle erikseen. Toisessa koeasetelmassa jokaiselta kehykseltä kerätään yksi mittaus, jonka jälkeen ilmentymien lukumäärää kasvatetaan yhdellä. Satunnaisuuden ja järjestelmän muun kuorman aiheuttaman vaihtelun häivyttämiseksi koe toistetaan 200 kertaa jokaiselle lippumuuttujan arvolle erikseen. Kolmannessa koeasetelmassa suoritettavissa kokeissa jokaiselta kehykseltä otetaan yksi mittaus, jonka jälkeen dynaamisia ilmentymiä edistetään. Myös kolmannessa koeasetelmassa hyödynnetään satunnaisuutta ilmentymien lähtösijaintien ja nopeusvektorien määräytymisessä, mistä syystä koe toistetaan 100 kertaa erikseen eri määrille dynaamisia ilmentymiä. Lisäksi kaikissa säteenseurannan suorituskykyä mittaavissa koeasetelmissa mittauksia ei tehdä eikä järjestelmää edistetä ensimmäisellä 2000 kehyksellä.

Kokeissa ylätason kiihdytysrakenteen ja alustavan pohjatason kiihdytysrakenteen muistin käyttö haetaan *vkGetAccelerationStructureBuildSizesKHR* kutsulla. Tiivistetyn pohjatason kiihdytysrakenteen muistin käyttö puolestaan haetaan käyttäen *vkCmdWriteAccelerationStructuresPropertiesKHR* kyselyä. Yksittäisen komennon suorittamiseen kuluvan ajan mittaamiseen käytetään Vulkan API:n aikaleima kyselyjä (engl. *timestamp queries*), jolloin aikaleimat kirjoitetaan käyttäen *vkCmdWriteTimestamp* komentoa. *vkCmdWriteTimestamp* luo riippuvuuden jonoon (engl. *queue*)[14] lisättyjen komentojen suoritusjärjestykselle, jolloin ai-

13. Rakentamalla useampi pohjatason kiihdytysrakenne *vkCmdBuildAccelerationStructuresKHR* kutsua kohden, vähennetään kutsun tuoman viiveen vaikutusta tuloksiin.

14. Vulkan API:ssa samoin kuin DirectX API:ssa näytönohjaimella suoritettavat komennot tulee sisällyttää komentolistoihin, jotka asetetaan jonoon suoritettavaksi. Vulkan API:ssa komentolistasta käytetään termiä ko-

kaleima kysely suoritetaan vasta, kun jonon edeltävät komennot ovat suoritettu. *vkCmdWriteTimestamp* komennon suorittaminen itsessään kestää tyypillisesti alle 3000 ns. Ylätason ja pohjatason kiihdytysrakenteiden rakentamiseen kuluneen ajan mittaaminen tapahtuu mittaamalla komennon *vkCmdBuildAccelerationStructuresKHR* suorituksen kestoa. Säteiden seuraamiseen kuluneen ajan mittaaminen puolestaan tapahtuu mittaamalla komennon *vkCmdTraceRaysKHR* suorituksen kestoa. Näin ollen esimerkiksi luodun kuvan mahdolliseen käsittelyyn ja kuvan kopioimiseen kehyspuskuriin kuluvaa aikaa ei huomioida mittauksissa. Myöskään G-puskurin päivittämiseen kuluvaa aikaa varjosäteitä seurattaessa ei huomioida.

Kolmannessa koeasetelmassa, jossa tutkitaan ylätason kiihdytysrakenteen päivittämisen vaikutusta suorituskykyyn, ilmentymien siirtymä määritellään ilmentymän minimaalisen rajaavan pallon halkaisijan monikertoina. Näin ollen siirtymä x1 tarkoittaa, että ilmentymä on liikkunut minimaalisen rajaavan pallon halkaisijan verran nopeusvektorin suuntaisesti, jolloin ilmentymän kiertymästä ja suuntavektorista riippuen, siirtymä poikkeaa enemmän tai vähemmän kiihdytysrakenteen ilmentymän rajaavan AABB:n suhteelliseen siirtymään. Kokeissa mittauksia tehdään, kunnes keskimääräinen siirtymä ilmentymää kohden on x50, jolloin esimerkiksi, jos kaikkien ilmentymien lukumäärä 500, joista 250 ilmentymää on liikkeessä, suoritetaan mittauksia, kunnes liikkuvien ilmentymien keskimääräinen siirtymä on x100. Vastaavasti liikkuvien ilmentymien nopeus ilmoitetaan suhteessa minimaaliseen rajaavaan pallon halkaisijaan, jolloin 10 % nopeudella liikkuva ilmentymä liikkuu minimaalisen rajaavan pallon halkaisijan mittaisen matkan kymmenen kehyksen aikana.

## 6.3 Tulokset

Ensimmäisessä koeasetelmassa suoritettujen kokeiden tulokset on esitetty pohjatason kiihdytysrakenteen muistin käytön, rakentamiseen kuluneen ajan ja säteenseurannan suorituskyvyn osilta kuvioissa 33, 34 ja 35 vastaavassa järjestyksessä. ALLOW_COMPACTION_BIT_KHR-lippubitin vaikutus tuloksiin on lähes mitätön, mistä syystä muistin käytön ja rakentamiseen kuluneen ajan osalta viivadiagrammeina esitetään vain ne tulokset, joissa

mentopuskuri (engl. *command buffer*), johon yksittäiset komennot tallennetaan (engl. *record*). Tyypillisesti fyysiset laitteet tukevat useampia erityyppisiä jonoja ja jokaisen jonon kautta voidaan suorittaa vain tietyntyyppisiä komentoja.

ALLOW_COMPACTION_BIT_KHR-lippubitti on asetettuna, jonka lisäksi viivadiagrammit muistin käytön ja rakentamiseen kuluneen ajan osalta esitetään ainoastaan muokkaamattoman yksikköpallon osalta. Sen sijaan muistin käytön ja rakentamiseen kuluneen ajan osalta palkkidiagrammeissa ovat näkyvissä myös ne tulokset, joissa geometriana käytetään muokattua yksikköpalloa ja ne tulokset, joissa ALLOW_COMPACTION_BIT_KHR-lippubitti ei ole asetettuna.

Pohjatason kiihdytysrakenteiden muistin käyttöä mittaavien kokeiden tuloksista selviää, että pohjatason kiihdytysrakenteen muistin käyttö kasvaa lineaarisesti kolmioiden lukumäärän myötä, eikä yksikköpallon ja muokatun yksikköpallon muistin käytöllä ole eroa ennen tiivistystä. Jälkimmäinen havainto selviää yksittäisiä tuloksia tarkastelemalla, joissa muistin käyttö on identtistä niiden geometrioiden välillä, joissa on sama määrä kolmioita ja sama VkBuildAccelerationStructureFlagsKHR lippumuuttujan arvo. Tiivistetyn pohjatason kiihdytysrakenteen muistin käytössä sen sijaan havaitaan pientä eroa yksikköpallon ja muokatun yksikköpallon välillä, joista muokkaamattoman yksikköpallo useimmiten käyttää vähemmän muistia.

Kolmioiden lukumäärän lisäksi pohjatason kiihdytysrakenteen muistin käyttöön merkittävästi vaikuttaa VkBuildAccelerationStructureFlagsKHR-lippumuuttujan arvo, mikä korostuu kiihdytysrakenteen tiivistämisen jälkeen. Tiivistäminen vähentää merkittävästi muistin käyttöä ei-animoiduille eli ilman ALLOW_UPDATE-lippubittiä, kuin myös ilman PREFER_FAST_BUILD -lippubittiä[15] rakennetuille pohjatason kiihdytysrakenteille. Nämä havainnot ovat linjassa NVIDIA:n suositusten (“Tips: Acceleration Structure Compaction” 2021) kanssa. Alhaisin muistin käyttö tiivistämisellä saavutetaan käyttämällä LOW_MEMORY-lippubittiä, jolloin kokeessa käytettyjen pohjatason kiihdytysrakenteiden muistin käyttö vähenee yli 50 %:lla. Suhteutettuna pohjatason kiihdytysrakenteen sisältämän kolmioverkon muistin käyttöön, voi kiihdytysrakenteen muistin käyttö ja siten tiivistämisen tuoma hyöty olla merkittävä[16], eikä kokeissa tiivistämisen havaita alentavan säteenseurannan suo-

15. Asetettaessa PREFER_FAST_BUILD-lippubitti päälle NVIDIA:n laitteistossa käytetään erillistä tiivistysmenetelmää (“Tips: Acceleration Structure Compaction” 2021).

16. Esimerkiksi kokeissa käytetyn 20000:sta kolmiosta koostuvan yksikköpallon sisältämän pohjatason kiihdytysrakenteen muistin käyttö käytettäessä LOW_MEMORY-lippubittiä on 1,13 megatavua ennen tiivistämistä ja 0,45 megatavua tiivistämisen jälkeen. 20000:sta kolmiosta koostuvan yksikköpallon verteksien tallentami-

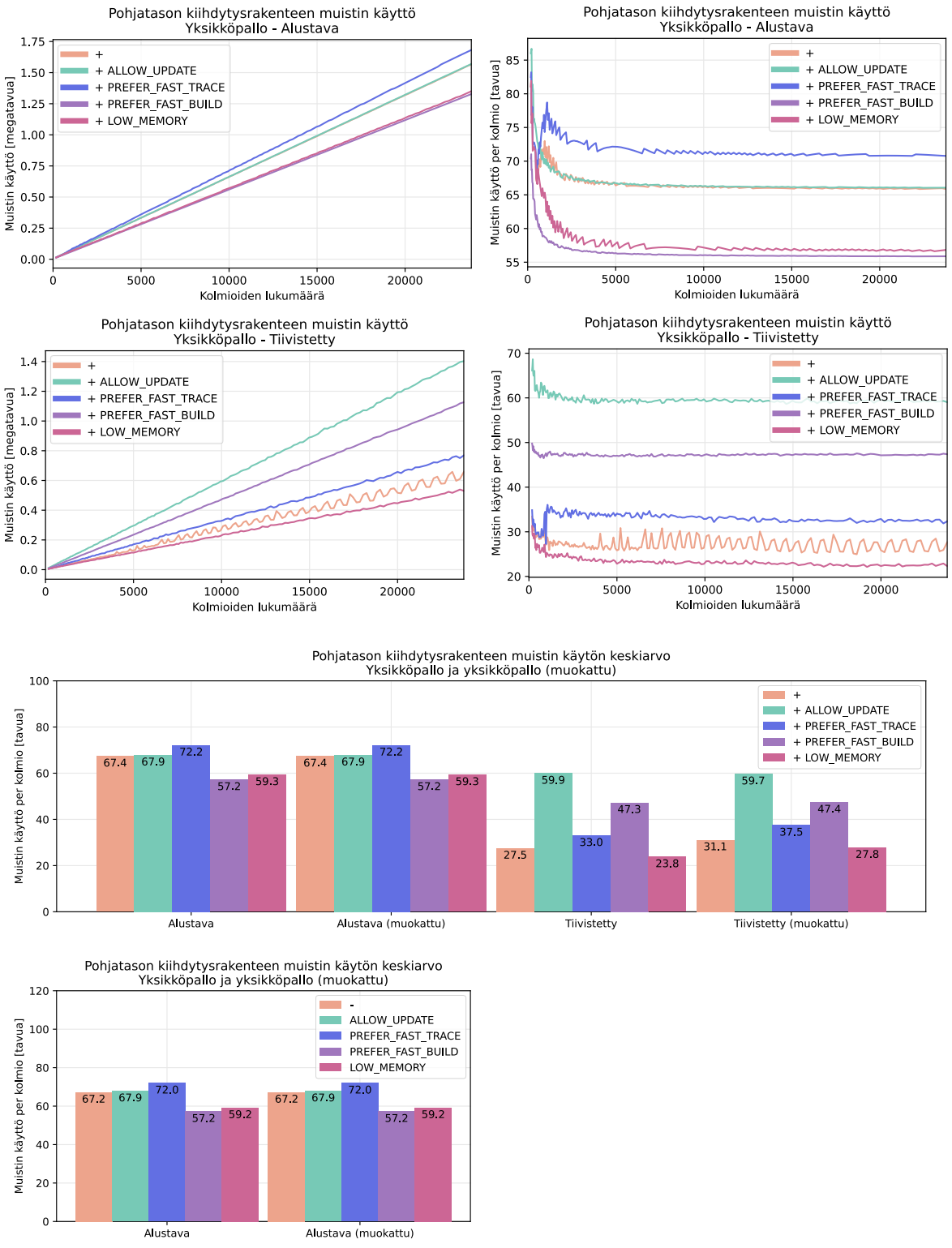
ristuskykyä, jolloin tiivistämistä voidaan pitää suositeltuna niille kiihdytysrakenteille, joita pidetään muistissa pitkiä aikoja (“Tips: Acceleration Structure Compaction” 2021).

Myös pohjatason kiihdytysrakenteen rakentamiseen kuluva aika kasvaa lineaarisesti kiihdytysrakenteen sisältämän kolmioverkon kolmioiden lukumäärän myötä. Samoin kuin pohjatason kiihdytysrakenteen muistin käyttöön, saadaan VkBuildAccelerationStructureFlagsKHR-lippumuuttujan arvolla vaikutettua myös kiihdytysrakenteen rakentamiseen kuluneeseen aikaan. PREFER_FAST_BUILD-lippubitin asettaminen vähentää pohjatason kiihdytysrakenteen rakentamiseen kuluvaa aikaa, mutta kasvattaa tiivistetyn rakenteen muistin käyttöä, sekä jokseenkin heikentää säteenseurannan suorituskykyä, jolloin PREFER_FAST_BUILD-lippubitin asettaminen ei välttämättä ole kannattavaa, mikäli kyseistä kiihdytysrakennetta käytetään useammilla kehyksillä. Toisaalta PREFER_FAST_BUILD-lippubitin asettaminen vähentää alustavan kiihdytysrakenteen muistin käyttöä, jolloin sen asettaminen soveltuu esimerkiksi tilanteisiin, joissa kiihdytysrakenne rakennetaan uudelleen joka kehyksellä, kuten voitaisiin tehdä animoiduille kolmioverkoille[17]. Puolestaan niillä pohjatason kiihdytysrakenteilla, joita käytetään useammilla kehyksillä tai esimerkiksi koko ohjelman suorituksen ajan, ei kiihdytysrakenteen rakentamiseen kuluvalla ajalla ole suurta merkitystä, jolloin PREFER_FAST_TRACE-lippubitin asettaminen voi olla perusteltua pidemmästä rakennusajasta huolimatta edellyttäen, että PREFER_FAST_TRACE-lippubitin asettaminen nopeuttaa säteenseurantaa.

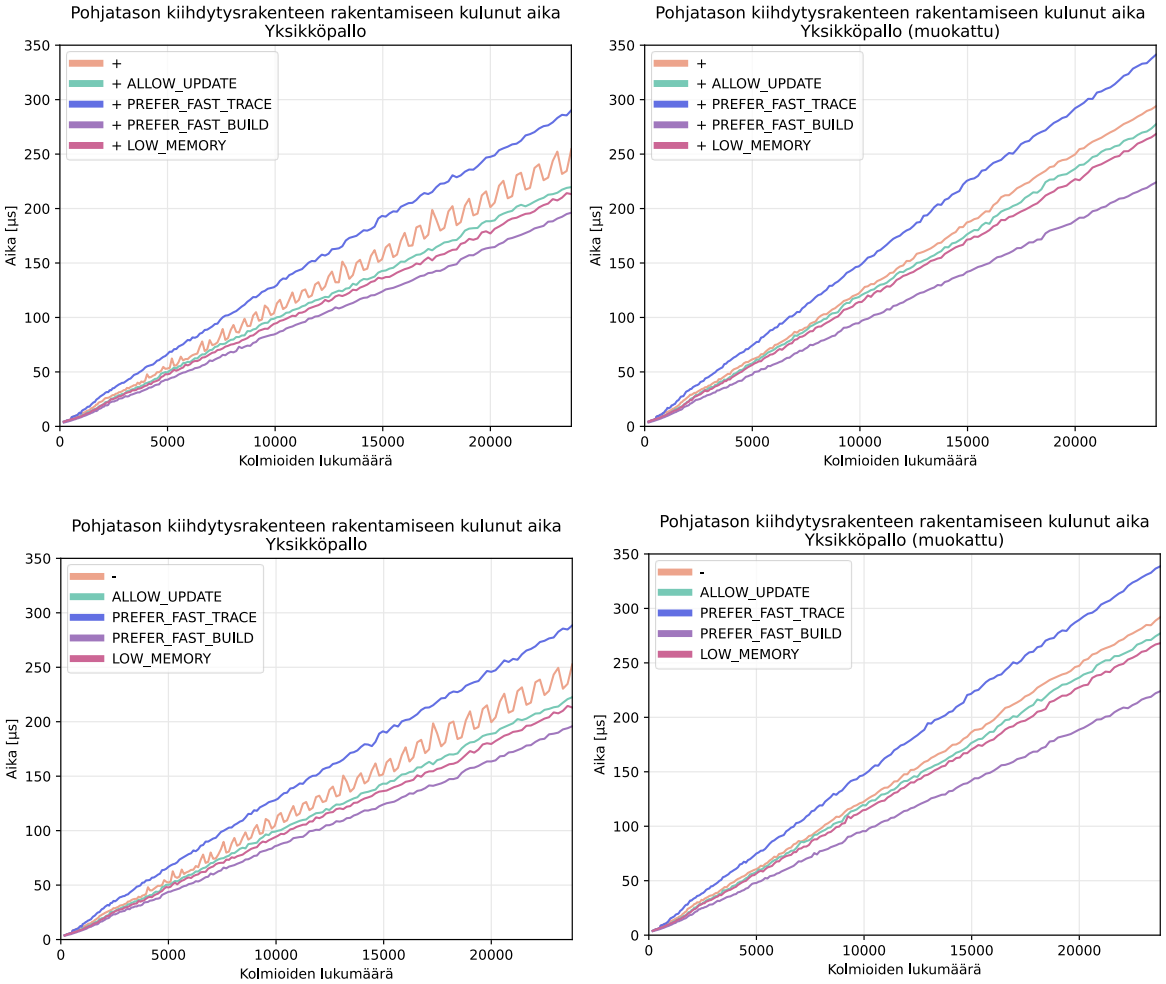
Siinä, missä pohjatason kiihdytysrakenteen muistin käytössä tai rakentamiseen kuluvassa

nen koeympäristössä käytetyllä verteksiformaatilla vaatii 1,92 megatavua ($20000 \cdot 3 \cdot 32$ tavua) muistia, jolloin kiihdytysrakenne kasvattaa muistin käyttöä 58 % ilman tiivistämistä ja 23 % tiivistämisen jälkeen suhteessa kolmioverkon muistin käyttöön. Käytettäessä indeksointia ja samaa verteksiformaattia kolmioverkon määrittelyssä, kolmioverkon sisältäessä 10100 uniikkia verteksiä olisi kolmioverkon muistin käyttö sen sijaan 0,44 megatavua ($10100 \cdot 32$ tavua $+ 60000 \cdot 2$ tavua), jolloin olettaen, että indeksointi ei ollenkaan tai ei ainakaan merkittävästi kasvata pohjatason kiihdytysrakenteen muistin käyttöä, on kiihdytysrakenteen muistin käyttö suhteessa kolmioverkkoon huomattavasti korkeampi. Toisaalta pohjatason kiihdytysrakenteen muistin käyttö voi olla pientä suhteessa kokonaisuuteen esimerkiksi sovelluksissa, joissa käytetään useita korkea resoluutioisia tekstuureja, sekä suhteessa myös kolmioverkkoon käytettäessä kompleksisempaa verteksiformaattia.

17. Animoitujen pohjatason kiihdytysrakenteiden päivittäminen voi alentaa säteenseurannan suorituskykyä (“Tips: Acceleration Structure Compaction” 2021) kuten kokeissa havaitaan käyvän ylätason kiihdytysrakenteille, jolloin uudelleenrakentaminen voi osoittautua paremmaksi vaihtoehdoksi kuin päivittäminen.



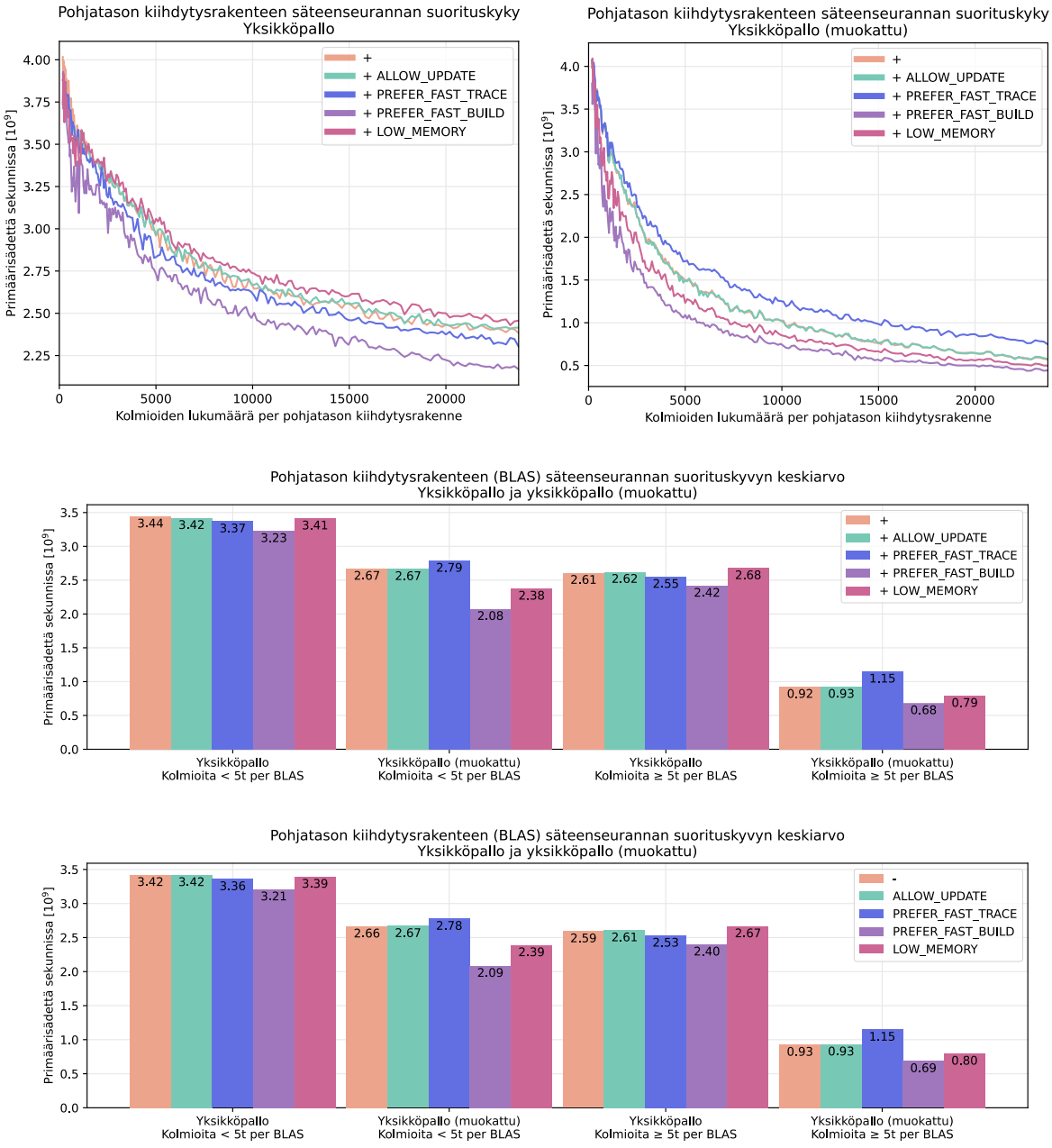

Kuvio 33. Pohjatason kiihdytysrakenteen muistin käyttö.




Kuvio 34. Pohjatason kiihdytysrakenteen rakentamiseen kulunut aika.

ajassa muokatun ja muokkaamattoman yksikköpallon välillä on havaittavissa vain pientä eroa, on ero säteenseurannan suorituskyvyssä merkittävä. Yksi todennäköinen selittävä tekijä on, että muokatun yksikköpallon kolmioverkko ei ole kupera, jolloin käytettäessä muokattua yksikköpalloa kolmioiden lukumäärän kasvaessa säde todennäköisemmin leikkaa useampien pohjatason kiihdytysrakenteen sisältämien AABB:den kanssa[18], kuin käytettäessä muokkaamatonta yksikköpalloa. Näin ollen säteenseurannassa todennäköisesti joudutaan suorittamaan useampia leikkaavuustarkasteluja säteen ja AABB:den, kuin myös säteen ja kolmioiden välillä lähimmän osuman löytämiseksi. Sen lisäksi, että kolmioverkon rakenne vaikuttaa

18. Muokkaamattoman yksikköpallon konveksisuus takaa sen, että säde leikkaa enimmillään kaksi kolmiota, kun taas muokatussa yksikköpallossa säde-kolmioleikkaavuuksia voi olla useampia säteen matkalla.




Kuvio 35. Pohjatason kiihdytysrakenteen vaikutus säteenseurannan suorituskykyyn.

säteenseurannan suorituskykyyn, laskee suorituskyky kolmioiden lukumäärän mukaisesti lähes logaritmisesti[19]. Saadut tulokset puoltavat yksinkertaistettujen eli matalamman tarkkuuden (engl. *level of detail*, *LOD*) kolmioverkkojen käyttämistä epätarkkuutta sietävässä säteenseurannassa kuten diffuusi heijastusten kuvaamisessa. Tuloksien perusteella lippubitin

19. Tätä voidaan pitää oletettuna tasapainoisen binääripuun kompleksisuusluokan ollessa O(log(n)).

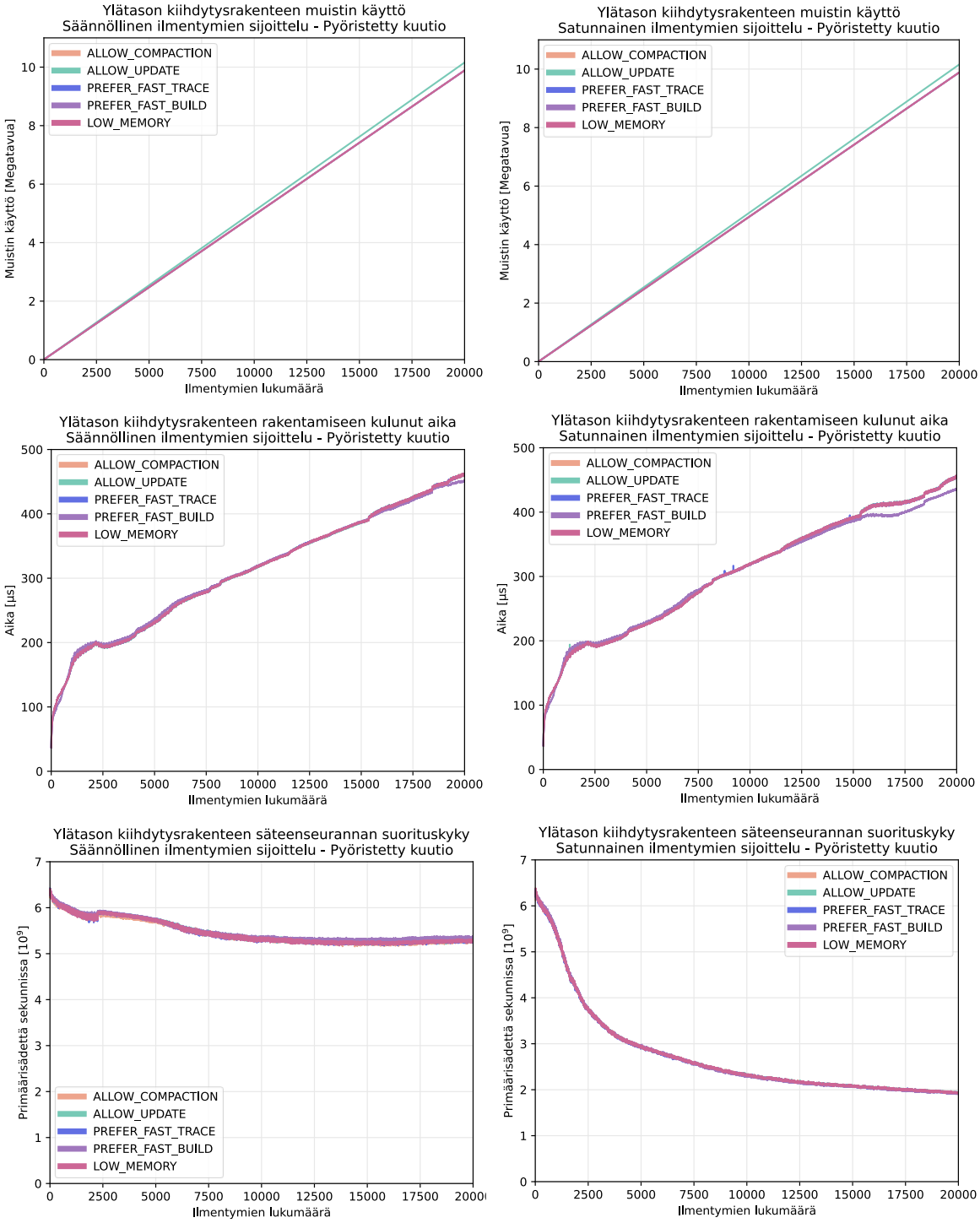
PREFER_FAST_TRACE asettamisen hyötyä ei voida yleisesti arvioida lippubitin asettamisen nopeuttaessa säteenseurantaa muokatulla yksikköpallolla, kun taas yksikköpallolla lippubitin asettaminen yllättäen hidastaa säteenseurantaa. Kenties tyypillisemmin käytettävät geometriat sijoittuvat näiden ääripäiden välille, joille lippubitin PREFER_FAST_TRACE asettamisen hyötyä ei voida kokeiden perusteella varmentaa. Toisaalta saadut tulokset suosivat yksinkertaistettujen geometrioiden käyttöä, joilla hyöty saattaa jäädä pienemmäksi.

Toisen koeasetelman kokeiden tulokset on esitetty ylätason muistin käytön, rakentamiseen kuluneen ajan ja säteenseurannan suorituskyvyn osilta kuviossa 36. Vaikka Vulkan-spesifikaatio ei erottele kiihdytysrakenteen rakentamisen yhteydessä annettavan VkBuildAccelerationStructureFlagsKHR-lippumuuttujan arvoja pohja- ja ylätason kiihdytysrakenteille, ei kokeiden tuloksissa ole havaittavissa merkittäviä eroja lippumuuttujan arvojen välillä ylätason kiihdytysrakenteella lukuun ottamatta muistin käytön osalta ALLOW_UPDATE-lippubittiä käyttämällä. Tuloksista nähdään, että ylätason kiihdytysrakenteen muistin käyttö kasvaa lineaarisesti ilmentymien lukumäärän kasvaessa. Vaikka ylätason kiihdytysrakenteen muistin käyttö ilmentymää kohden on suurempi kuin pohjatason kiihdytysrakenteessa kolmiota kohden, on ylätason kiihdytysrakenteen muistin käyttö todennäköisesti merkittävästi pienempää suhteessa kaikkien pohjatason kiihdytysrakenteiden kokonaismuistin käyttöön tyypillisemmässä virtuaalimaailmassa, jossa virtuaalimaailma sisältää useampia pohjatason kiihdytysrakenteita. Ylätason kiihdytysrakenteen rakentamiseen kulunut aika kasvaa lineaarisesti ilmentymien lukumäärän kasvaessa lukuun ottamatta alhaisella lukumäärällä ilmentymiä[20]. Ilmentymien sijoittautumisen säännöllisyys ei vaikuta muistin käyttöön, eikä merkittävissä määrin rakenteen rakentamiseen kuluneeseen aikaan. Sen sijaan säteenseurannan suorituskykyyn vaikutus on merkittävä. Toisaalta suorituskyvyn alentumisen välttäminen voi olla hankalaa ilman modulaarista virtuaalimaailman suunnittelua, mikä ei kaikissa sovelluskohteissa ole mielekästä[21][22]. Säteenseurannan suorituskyvyn alenemisen selittävää kiihdy-

20. Yksi todennäköinen syy epälineaarisuudelle on vain yhden kiihdytysrakenteen rakentaminen *vkCmdBuildAccelerationStructuresKHR* kutsua kohden, minkä havaittiin aiheuttavan vastaavanlaisen epälineaarisuuden pohjatason kiihdytysrakenteilla koeasetelmia suunniteltaessa.

21. Pois lukien vokselointiin perustuvat sovellukset, joissa tosin voidaan käyttää erilaista lähestymistapaa leikkaavuustarkastelussa.

22. Mielenkiintoinen lisäkoe olisi selvittää voitaisiinko yhdistelemällä lähimpiä kolmioverkkoja samoihin pohjatason kiihdytysrakenteiseen vähentää suorituskyvyn alentumista erityisesti sijoitettaessa ilmentymiä epä-




Kuvio 36. Ylätason kiihdytysrakenteen muistin käyttö, rakentamiseen kulunut aika ja vaikutus säteenseurannan suorituskykyyn.

säännöllisesti.

tysrakenteen rakenteellista muutosta ei voida tulosten perusteella tarkentaa[23], mutta tulokset puoltavat merkityksettömien tai merkitykseltään vähäisten ilmentymien karsimista pois ylätason kiihdytysrakenteesta erityisesti suurella lukumäärällä potentiaalisia ilmentymiä. Esimerkiksi kokeessa ylätason kiihdytysrakenteen sisältäessä 5000 epäsäännöllisesti sijoiteltua ilmentymää, laskee säteenseurannan suorituskyky 50 % huolimatta siitä, ettei kiihdytysrakenteen sisältämiin geometrioihin yksikään seurattava säde osu.

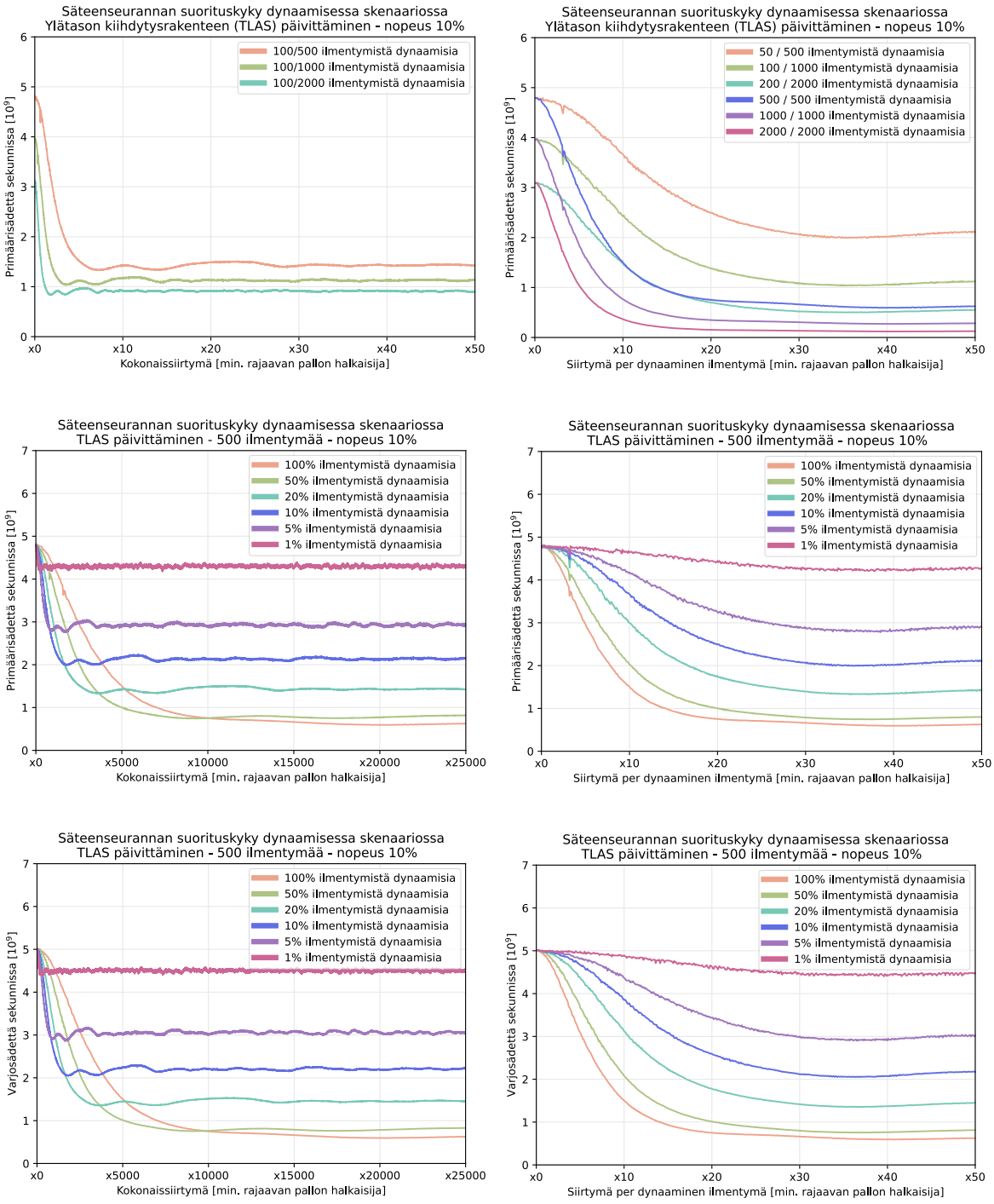
Kolmannessa koeasetelmassa suoritettujen kokeiden tulokset ovat puolestaan esitetty kuvioissa 37 ja 38. Tuloksissa siirtymällä tarkoitetaan ilmentymän kulkemaa matkaa, eikä siirtymää suhteessa ilmentymän lähtöpisteeseen. Ilmentymien liikkeen nopeus ei merkittävästi vaikuta tuloksiin[24], mistä syystä nopeudella 50 % suoritettujen kokeiden tuloksia ei esitetä erillisinä viivadiagrammeina. Sen sijaan nopeuksien 10 % ja 50 % välinen erotus esitetään pinta-alana kuviossa 38 oikealla ylhäällä. Tuloksista havaitaan, että ylätason kiihdytysrakenteen päivittämisen vaikutus säteenseurannan suorituskykyyn riippuu sekä liikkuvien ilmentymien lukumäärästä, että liikkuvien ilmentymien siirtymästä. Liikkuvien ilmentymien lukumäärä määrittää sen, mihin säteenseurannan suorituskyky lopulta asettuu siirtymän kasvaessa. Ylätason kiihdytysrakenteen päivittämisen vaikutus säteenseurannan suorituskykyyn voi olla hyvinkin merkittävää jo alhaisellakin määrällä liikkuvia ilmentymiä, mikäli siirtymät ovat suuria. Vaikka ylätason kiihdytysrakenteen päivittäminen on suhteessa huomattavasti uudelleenrakentamista nopeampaa, on ylätason kiihdytysrakenteen rakentaminen kuitenkin kohtalaisilla ilmentymien lukumäärillä mahdollista suorittaa reaaliajassa jokaisella kehyksellä, jolloin uudelleenrakentaminen voi osoittautua päivittämistä paremmaksi vaihtoehdoksi pienemmilläkin ilmentymien siirtymillä. Varjo- ja primäärisäteiden seuraamisen suorituskyvyn välillä ei tuloksissa havaita merkittävää eroa[25].

Tutkielman kokeet suoritettiin minimaalisessa koeympäristössä muun muassa ilman tekstuu-

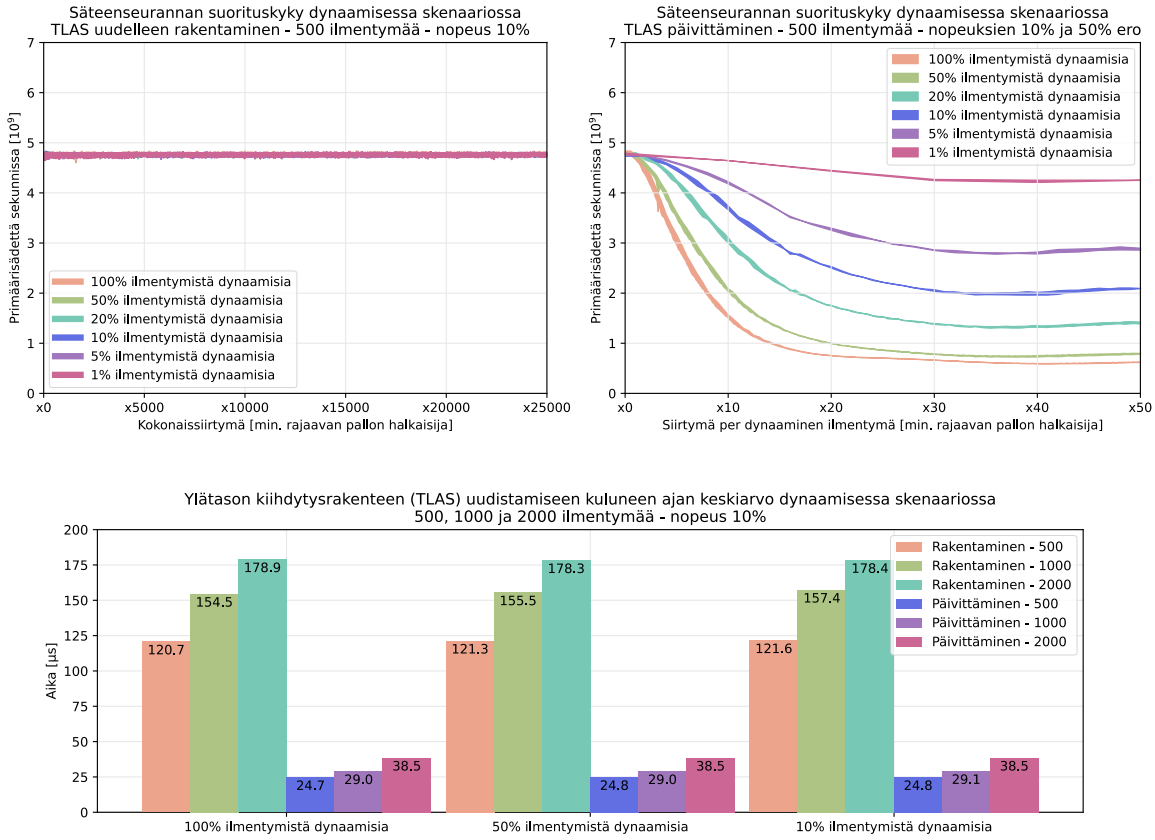
23. Koska säde ei leikkaa yhdenkään geometrian kanssa, tulee suorituskyvyn alenemiseksi säteen leikata useampien ylätason kiihdytysrakenteessa eri tasoilla sijaitsevien AABB:den kanssa.

24. Näin ollen ylätason kiihdytysrakenteiden päivityskutsujen määrä ei vaikuta tuloksiin, sillä 10 % nopeudella tehdään viisinkertainen määrä päivityksiä saman siirtymän aikana kuin nopeudella 50 %.

25. Kokeessa seurattavat vierekkäiset varjosäteet ovat lähes yhdensuuntaisia eli koherentteja, jolloin varjosäteiden seuraamisen suorituskyky säilyy vastaavanlaisena kuin primäärisäteiden, kun taas epäkoherenttiuden tiedetään vähentävän suorituskykyä etenkin koeasetelmaa kompleksisemmissa virtuaalimaailmoissa (Meister ym. 2020)




Kuvio 37. Ylätason kiihdytysrakenteen päivittämisen vaikutus säteenseurannan suorituskykyyn.




Kuvio 38. Ylätason kiihdytysrakenteen uudelleenrakentamisen vaikutus säteenseurannan suorituskykyyn eli referenssi (ylhäällä vasemmalla), ylätason kiihdytysrakenteen päivittämisen vaikutus säteenseurannan suorituskykyyn eri nopeudella liikkuvilla ilmentymillä (ylhäällä oikealla), sekä ylätason kiihdytysrakenteen päivittämiseen ja uudelleenrakentamiseen kulunut aika kolmannessa koeasetelmassa (alhaalla).

reita ja useampia pohjatason kiihdytysrakenteita, eikä säteenseurannan suorituskyvyn tuloksia voida siten sellaisenaan yleistää useimpiin reaaliaikaisen renderöinnin sovelluksiin, vaan kokeiden tarkoituksena on antaa yläraja säteenseurannan suorituskyvylle koelaitteistolla ja tutkia siihen vaikuttavia tekijöitä. Säteenseurannan suorituskyvyn mittauksissa päästiin ilmoitettuun viiteen gigasäteeseen sekunnissa (“NVIDIA GeForce RTX 2060 Specification” 2022) minimaalisessa koeasetelmassa. Renderöitäessä 1920 x 1080 kokoiseen kehyspuskuriin ja tavoitteena tuottaa 60 kehystä sekunnissa, kyettäisiin tällöin minimaalisessa virtu-

aalimaailmassa koelaitteistolla seuraamaan noin 40 sädettä pikseliä kohden[26] olettaen, että koko kehykseen käytettävä aika voidaan käyttää säteiden seuraamiseen. Hyödyntämällä renderöinnissä polunseurantamenetelmää seuraten neljä sädettä polkua kohden[27] saadaan kehyksellä pikseliä kohden muodostettua 10 näytettä olettaen, että sekundäärisäteitä kyetään seuraamaan samalla nopeudella kuin primäärisäteitä. Kuten kuviosta 20, jossa myöskin seurataan neljä sädettä näytettä kohden, voidaan havaita, esiintyy hyvinkin pelkistetyssä virtuaalimaailmassa merkittävää häiriötä kahdeksalla näytteellä, kuin myös 16:lla näytteellä, kunnes vasta sadoilla näytteillä häiriö häviää. Tämä luo tarpeen häiriönpoistolle, jonka lisäksi virtuaalimaailman sisältäessä dynaamisia ilmentymiä, tulee ylätason kiihdytysrakenne päivittää tai uudelleenrakentaa joka kehyksellä, mitkä tyypillisesti vievät aikaa säteenseurannalta, mikä tarkoittaa vähemmän näytteitä kehyksellä. Tästä seuraa se, että laitteistokiihdytyksestä huolimatta renderöintiyhtälön approksimointi reaaliajassa etenkin kompleksisemmissa virtuaalimaailmoissa on merkittävä haaste. Vaikka uudemman sukupolven säteenseurantaa tukevat näytönohjaimet todennäköisesti tarjoavatkin huomattavasti koelaitteistoa paremman suorituskyvyn[28], uskotaan haasteen säilyvän viimeisimmilläkin kuin tulevillakin näytönohjaimilla lähitulevaisuudessa. Seuraavan luvun tarkoitus onkin tutkia, missä määrin ja millä tavalla säteenseurantaa kyetään reaaliaikaisessa renderöinnissä hyödyntämään kompleksisemmissa virtuaalimaailmoissa.

26. $\frac{5 \cdot 10^9}{(1920 \cdot 1080 \cdot 60)} \approx 40{,}19$.

27. Neljällä säteellä voidaan esimerkiksi kuvata suora, sekä yhden kimmokkeen epäsuora valaistus seuraamalla primäärisäde, varjosäde, heijastussäde ja heijastussäteen leikkauspisteestä toinen varjosäde.

28. Tarkkaa tietoa viimeisimmän NVIDIA Ada Lovelace -arkkitehtuurin tai sitä edeltävän Ampere-arkkitehtuurin näytönohjaimien säteenseurannan suorituskyvystä ei ole saatavilla. Verrattuna 68 ensimmäisen sukupolven RT-ydintä sisältävään Turing-arkkitehtuurin NVIDIA GeForce RTX 2080 Ti näytönohjaimeen, jonka suorituskyky on 10 gigasädettä sekunnissa (Corporation 2018), on NVIDIA GeForce RTX 4090 näytönohjaimessa 128 kolmannen sukupolven RT-ydintä (Corporation 2022a), jolloin sen voidaan olettaa tarjoavan moninkertaisen suorituskyvyn koelaitteistoon verrattuna.

# 7 Säteenseurannan hyödyntäminen videopeleissä

Yhtenä tutkielman tutkimuskysymyksenä on tutkia, missä määrin laitteistokiihdytteistä säteenseurantaa kyetään hyödyntämään nykyisissä reaaliaikaisen renderöinnin sovelluksissa. Siinä, missä edellisessä luvussa tutkittiin säteenseurannan suorituskykyä ja siihen vaikuttavia tekijöitä minimaalisessa virtuaalimaailmassa, voidaan tutkimuskysymyksen kannalta mielenkiintoisina kohteina pitää suuremman mittakaavan sovelluksia, joissa virtuaalimaailma on huomattavasti koeasetelmia kompleksisempi. Tämä toisaalta asettaa haasteen sopivan aineiston löytämiselle sovellusten kaupallisen luonteen vuoksi. Yhtenä merkittävänä poikkeuksena voidaan kuitenkin pitää GDC:tä (Game Developers Conference)[1], joka tarjoaa tietoa tunnetuissa videopeleissä käytetyistä ratkaisuista muun muassa säteenseurantaan liittyen. Tästä syystä tutkimuskysymykseen vastataan yksinomaan laitteistokiihdytteistä säteenseurantaa hyödyntävien videopelien pohjalta. Tutkimuskysymystä varten tutkielmaan valittiin aineistoksi kahdeksan GDC konferenssiesityksen tallennetta, sekä kaksi artikkelia Ray Tracing Gems -kirjasarjan[2] toisesta osasta. Tutkimuksessa käytetty aineisto esitetään taulukossa 8. Luvussa säteenseurannan hyödyntäminen videopeleissä esitellään aineiston pohjalta, mikä ei välttämättä kata säteenseurannan hyödyntämistä kaikissa videopeleissä kokonaisuudessaan, jolloin taulukossa ei välttämättä ole lueteltuna kaikkia niitä valon ilmenemismuotoja, joihin videopeleissä säteenseurantaa on hyödynnetty[3].

Verrattuna edellisen luvun kokeissa käytettyihin minimaalisiin virtuaalimaailmoihin, tarjoavat videopelit täysin erilaisen näkökulman niiden virtuaalimaailmojen ollessa hyvinkin kompleksisia koostuen useista geometrioista ja dynaamisista valoista, sekä dynaamisista ja animoiduista geometrioista. Lisäksi videopelit tyypillisesti pyrkivät korkeisiin kehysnopeuksiin ja korkeaan resoluutioon, mikä tarkoittaa, että säteenseurannan tukena tulee hyödyntää häiriönpoistoa ja rasterointitekniikkaa, mikä taas tyypillisesti tarkoittaa G-puskurin luontia

1. GDC on vuodesta 1988 lähtien järjestetty jokavuotinen videopelikehittäjien konferenssi, jossa eri peliyhtiöiden edustajat jakavat tietoa videopelien toteutukseen liittyvistä haasteista ja ratkaisuista.

2. Kirjasarja on ilmaiseksi saatavilla verkossa lähteessä (“Ray Tracing Gems Series” 2021).

3. Osassa videopelejä säteenseuranta voidaan asettaa päälle valonilmenemismuotokohtaisesti, jolloin myöskään sitä, minkälaisella laitteistolla voidaan säteenseuranta asettaa päälle kokonaisuudessaan ilman reaaliaikaisuudesta luopumista ei voida aineiston pohjalta sanoa.

| Peli (pelimoottori) | Julkaisuvuosi | Lähde |
|---|---|---|
| Battlefield V (Frostbite) | 2018 | (DICE 2019) |
| Shadow of the Tomb Raider (Foundation Engine) | 2018 (2019) | (NVIDIA 2019) |
| Metro Exodus (4A Engine) | 2019 | (Games 2019) |
| Wolfenstein: Youngblood (id Tech 6) | 2019 | (MachineGames 2020) |
| Control (Northlight) | 2019 | (Sjöholm, Jukarainen ja Aalto 2021) |
| Fortnite (UE4) | 2020 | (Kelly ym. 2021) |
| Deathloop (Void engine) | 2021 | (Lyon 2022) |
| Far Cry 6 (Dunia engine 2) | 2021 | (Toronto 2022) |
| Hitman 3 (Glacier Engine) | 2021 (2022) | (Interactive 2022) |
| Forspoken (Luminous engine) | 2023 | (Productions 2022) |

Taulukko 8. Luvussa käytetty aineisto. Julkaisuvuoden perässä oleva vuosiluku tarkoittaa päivityksen mukana tulleiden säteenseuranta ominaisuuksien julkaisuvuotta, niillä videopeleillä joilla laitteistokiihdytteinen säteenseuranta ei ollut tuettu pelin julkaisuhetkellä. Far Cry 6 ja Wolfenstein: Youngblood -videopeleissä käytetyt pelimoottorit perustuvat lähteisiin (“Far Cry 6 Benchmarked” 2021) ja (“How Wolfenstein: Youngblood scales from top-end PC to Nintendo Switch” 2019).

jokaisella kehyksellä. Videopeleissä virtuaalimaailman dynaamisuus puolestaan tarkoittaa säteenseurannan kannalta sitä, että ylätason kiihdytysrakenne tulee joko päivittää tai uudelleenrakentaa jokaisella kehyksellä, jonka lisäksi mahdollisesti tulee luoda uusia tai päivittää olemassa olevia pohjatason kiihdytysrakenteita[4]. Näin ollen aineiston videopeleissä säteenseurantaan käytettävä aika asettuu enimmillään muutamaan millisekuntiin kehyksellä. Riippuen reaaliaikaisesta sovelluksesta, säteenseurantaan käytettävä aika saattaa poiketa tästä merkittävästikin, esimerkiksi tapauksessa, jossa virtuaalimaailma koostuu staattisista ja eianimoiduista geometrioista tai renderöinti perustuu pelkästään säteenseurantaan. Toisaalta pelimoottorit voivat toimia pohjana myös muille reaaliaikaisille sovelluksille kuin videopeleille, jolloin käytettävät menetelmät voivat sovelluskohteesta huolimatta olla samoja, mutta niitä käytetään esimerkiksi laajemmin valon eri ilmenemismuotojen kuvaamiseen tai samojen valon ilmenemismuotojen kuvaamiseen suuremmalla näytemäärällä keventäen häiriönpoistoa.

Aineistossa säteenseurannan laskennallinen vaativuus ja videopelien tiukka aikabudjetti nousevat merkittäväksi haasteeksi. Yksi aineiston videopeleissä kuin myös videopeleissä yleisemminkin käytetty ratkaisu säteenseurannan nopeuttamiseksi ja korkeiden kehysnopeuksien takaamiseksi on suorittaa renderöinti matalammalla resoluutiolla ja ylösskaalata (engl. *upscale*) kehyspuskuri ennen sen esittämistä näytöllä. Videopelien laajasti tukemia teknologioita ylösskaalaukseen ovat NVIDIA:n syväoppimiseen perustuva DLSS (Deep Learning Super Sampling), AMD:n ajalliseen uudelleen käyttöön perustuva FidelityFX™ Super Resolution ja Intelin tekoälyyn pohjautuva $X^eSS$ ($X^e$ Super Sampling)[5]. Toinen aineiston videopeleissä esille tullut ratkaisu reaaliaikaisuuden mahdollistamisessa on säteenseurannan rajaaminen tietyn tai tiettyjen valon ilmenemismuotojen kuvaamiseen yksinään tai yhdessä rasterointipohjaisten menetelmien kanssa eli niin kutsutun hybriditekniikan hyödyntäminen renderöinnissä. Hybriditekniikan hyödyntämistä aineiston videopeleissä tarkemmin esitel-

4. Lueteltujen lisäksi näytönohjaimella voidaan videopeleissä muun muassa suorittaa muuta laskentaa tai kuten tyypillistä on, voidaan renderöinti G-puskurin luonnin lisäksi suorittaa useammilla läpivienneillä esimerkiksi varjokarttojen päivittämiseksi tai erilaistan efektien kuvaamiseksi, jotka kaikki saattavat vaikuttaa säteenseurantaan käytettävään aikaan.

5. Näistä DLSS ylösskaalausta voidaan käyttää vain NVIDIA:n näytönohjaimilla, FidelityFX™ Super Resolution ylösskaalausta näytönohjainvalmistajasta riippumatta ja Intelin $X^eSS$ ylösskaalausta useimmilla näytönohjaimilla.

lään aliluvussa 7.1. Aliluvussa 7.2 puolestaan esitellään muut aineistossa esille tulleet aiheet, jotka vaativat erityistä huomiota reaaliaikaisuuden mahdollistamisessa.

## 7.1 Hybriditekniikan hyödyntäminen

Taulukossa 9 esitetään kooste hybriditekniikan hyödyntämisestä aineiston videopeleissä. Taulukon toisessa sarakkeessa esitetään ne valon ilmenemismuodot, joiden kuvaamiseen hyödynnetään laitteistokiihdytteistä säteenseurantaa. Taulukossa kuten tässä luvussa yleisestikin, spekulaari heijastukset ja ympäristön okkluusio erotellaan globaalista valaistuksesta, jolla tässä yhteydessä, kuin myös videopelien yhteydessä useastikin, tarkoitetaan diffuusi heijastuksia. Taulukon kolmannessa sarakkeessa on puolestaan esitetty ne rasterointipohjaiset menetelmät, joita käytetään säteenseurannan kanssa yhdessä saman valon ilmenemismuodon kuvaamiseen. Taulukon neljännessä sarakkeessa esitetään tuki ylösskaalausteknologioille perustuen aineiston lisäksi lähteisiin ("NVIDIA RTX: List Of All Games, Engines And Applications Featuring GeForce RTX-Powered Technology" 2022) ja ("AMD FidelityFX™ Supported Games" 2022).

Kuten taulukosta selviää, käytetään säteenseurantaa monipuolisesti eri valon ilmenemismuotojen kuvaamiseen yhdessä rasterointipohjaisten menetelmien kanssa. Kun säteenseurantaa käytetään yhdessä rasterointipohjaisten menetelmien kanssa saman valon ilmenemismuodon kuvaamiseen, tulee luoda heuristiikka sille, missä tilanteissa käytetään säteenseurantaa ja missä tilanteissa rasterointipohjaisia menetelmiä. Aineiston videopeleissä yleisiä heuristiikkoja säteenseurannan käyttämisen rajoittamiseksi ovat säteenseurannan käyttö ainoastaan lähellä virtuaalikameraa[6], säteenseurannan käyttäminen ainoastaan tietyn tai tietynlaisten valonlähteiden tuottaman valon ilmenemismuotojen kuvaamiseen, sekä säteenseurannan käyttö ainoastaan niissä pikseleissä, joissa rasterointipohjaisia menetelmiä ei kyetä luotettavasti hyödyntämään tai niiden tuottama laatu on alhainen.

Shadow of the Tomb Raider -videopelissä säteenseurannan käyttö on rajoitettu suuntavalojen aiheuttamien varjojen kuvaamiseen lähelle virtuaalikameraa, sekä pistemäisten valolähteiden aiheuttamien varjojen kuvaamiseen lähelle valonlähdettä. Muualla videopelissä

6. Tällöin säteenseurannan määräävänä peitteenä (engl. *mask*) voidaan käyttää syvyyspuskuria.

| Peli | Ilmenemismuoto | Rasterointi | Ylösskaalaus |
|---|---|---|---|
| Battlefield V | R | SSR | DLSS |
| Shadow of the Tomb Raider | S | SM | DLSS & $X^eSS$ |
| Metro Exodus | AO, GI | SSAO, SSR | DLSS |
| Wolfenstein: Youngblood | R | SSR, EM | DLSS |
| Control | GI, R, S | SM, VIR | DLSS |
| Fortnite | AO, GI, R, S | EM, SM | DLSS |
| Deathloop | AO, S | ? | DLSS & FSR |
| Far Cry 6 | R, S | SSR, EM | DLSS & FSR |
| Hitman 3 | R, S | SSR, PR, SM | DLSS & FSR & $X^eSS$ |
| Forspoken | AO, S | CACAO, SM | DLSS & FSR |

Taulukko 9. Valon ilmenemismuotojen lyhenteet: R = spekulaari heijastukset, GI = diffuusi heijastukset, AO = ympäristön okkluusio ja S = varjot. Rasterointipohjaisten menetelmien lyhenteet: SM = varjokartta, PR = tasoheijastukset, EM = ympäristökartta ja VIR = vokselointia hyödyntävä irradianssitilavuusmenetelmä. Aineistossa Deathloop-videopelistä ei ole saatavilla tietoa käytetyistä rasterointipohjaisista menetelmistä.

käytetään varjokarttoja varjojen kuvaamiseen. Vastaavasti etäisyyteen perustuen Forspoken-videopelissä säteenseurantaisen ympäristön okkluusion (RTAO) käyttö on rajoitettu lähelle virtuaalikameraa, kun taas kauempana virtuaalikamerasta sovelletaan AMD:n CACAO-menetelmää. Lisäksi pelin säteenseurantainen ympäristön okkluusio lasketaan varsinaista kuvapuskuria alhaisemmalla resoluutiolla. Pelissä säteenseurantaa hyödynnetään myös varjojen kuvaamiseen, missä sen käyttö on rajattu ainoastaan puolivarjojen alueelle täydentämään varjokarttoja.

Far Cry 6 -videopelissä säteenseurantaa käytetään heijastusten kuvaamiseen yhdessä näkymäavaruuden heijastusten ja ympäristökarttojen kanssa. Säteenseurannan käyttö on rajoitettu niihin pikseleihin, joissa kuvapuskurista ei kyetä heijastuksia tuottamaan. Monissa tapauksissa tämä tarkoittaa säteenseurannan hyödyntämistä lähellä virtuaalikameraa ja näkymäavaruuden heijastuksia kaukana virtuaalikamerasta, kun taas keskivaiheilla hyödynnetään säteenseurantaa ainoastaan silloin, kun kuvapuskurista ei kyetä heijastuksia tuottamaan. Li-

säksi ympäristökarttoja käytetään tilanteissa, joissa säteenseuranta tai näkymäavaruuden heijastukset eivät kumpikaan sovellu heijastusten kuvaamiseen.

Myös Battlefield V, Hitman 3 ja Metro Exodus -videopeleissä säteenseurantaa käytetään täydentämään näkymäavaruuden menetelmiä. Battlefield V -videopelissä säteenseurantaa käytetään adaptiivisesti näkymäavaruuden heijastusten ohella spekulaari heijastusten kuvaamiseksi. Pelissä kuvapuskuri jaetaan 16 x 16 pikseleiden muodostamiin tiileihin (engl. *tile*), joiden sisältämien virtuaalimaailman kappaleiden materiaaleille määriteltyjen karheuksien (engl. *roughness*) ja normaalien, sekä katselukulman perusteella määritellään tiilelle seurattavien säteiden lukumäärä, jolloin pikseliä kohden seurataan 1/8 - 1 heijastussädettä. Hitman 3 -videopelissä näkymäavaruuden heijastuksia täydennetään säteenseurannalla, kun kuvapuskurista ei kyetä heijastuksia tuottamaan. Lisäksi pelissä auringon ja kuun aiheuttamien varjojen kuvaamiseen käytetään säteenseurantaa varjokartoilla toteutettujen varjojen pehmentämiseen varjostajan etäisyyden kasvaessa, sekä paikkaamaan varjokarttojen resoluutiosta johtuvia ongelmia. Metro Exodus -videopelissä näkymäavaruuden menetelmiä ja säteenseurantaa käytetään yhden kimmokkeen globaalin valaistuksen ja ympäristön okkluusion[7] kuvaamiseen. Pelissä säteenseurannan käyttö rajoitetaan niihin pikseleihin, joissa kuvapuskurista ei kyetä heijastuksia tuottamaan. Lisäksi pelissä säteenseurannan käyttö globaalin valaistuksen kuvaamiseen rajoitetaan pelkästään auringon ja kuun tuottamaan valoon.

Wolfstein: Youngblood -videopelissä säteenseurantaa käytetään ensisijaisena menetelmänä heijastuksen tuottamiseen rajoittaen säteen enimmäispituutta materiaalin karheuden mukaisesti, jolloin karkeammilla materiaaleilla käytetään lyhyempiä säteitä kuin hyvin heijastavilla materiaaleilla. Lisäksi pelissä ympäristökartoilla täydennetään heijastuksia niiden säteiden osalta, jotka eivät enimmäispituuden rajoissa leikkaa virtuaalimaailman geometrian kanssa. Control-videopelissä säteenseurantaa käytetään heijastusten, lyhyen etäisyyden epäsuoran valaistuksen[8] (engl. *near field indirect diffuse illumination*) ja kontaktivarjojen[9] (engl. *con-*

7. Kun diffuusi heijastuksia kuvataan säteenseurannalla, voidaan heijastussäteiden pituuksien perusteella muodostaa approksimaatio ympäristön okkluusiosta.

8. Menetelmässä käytetään lyhyitä säteitä epäsuoran valaistuksen kuvaamiseen samoin kuin ympäristön okkluusiossa lyhyitä säteitä käytetään ympäristön valaistuksen approksimointiin.

9. Kontaktivarjoilla tarkoitetaan lyhyeltä etäisyydeltä tarkasteltuja varjoja, mutta poiketen ympäristön

*tact shadow*) kuvaamiseen. Heijastusten ja lyhyen etäisyyden epäsuoran valaistuksen tukena käytetään luvussa 3.1 lyhyesti esiteltyä esilaskettua vokselointia hyödyntävää irradianssitilavuusmenetelmään pohjautuvaa menetelmää. Muiden varjojen kuin kontaktivarjojen kuvaamiseen pelissä käytetään varjokarttoja. Aineiston peleistä monipuolisimmin säteenseurantaa tuetaan Fortnite-videopelissä, jossa säteenseurantaa hyödynnetään spekulaari heijastusten, yhden kimmokkeen globaalin valaistuksen, ympäristön okkluusion, sekä varjojen kuvaamiseen. Pelissä heijastusten kuvaamiseksi säteenseurantaa hyödynnetään ainoastaan hyvin heijastaville materiaaleille ja karheammille materiaaleille käytetään ympäristökarttoja. Lisäksi pelissä globaalin valaistuksen ja varjojen mallintaminen säteenseurannalla on rajattu ainoastaan auringonvalolle. Deathloop-videopelissä säteenseurantaa käytetään auringonvalon aiheuttamiin varjoihin, sekä ympäristön okkluusioon.

Riippuen siitä, mihin valon ilmenemismuotoon säteenseurantaa käytetään, vaihtelee aineiston videopeleissä säteenseurannalla tuotettujen näytteiden lukumäärä kehyksellä alle yhdestä näytteestä korkeintaan muutamaan näytteeseen pikseliä kohden, mikä johtaa huomattavaan häiriöön ja luo tarpeen häiriönpoistolle. Aineistossa häiriönpoistoa tarkemmin käsitellään vähemmissä määrin, mutta olettamuksena aineiston videopeleissä on, että häiriönpoistossa näytteitä yhdistellään sekä spatiaalisesti että ajallisesti eli käytettään niin kutsuttua tila-ajallista (engl. *spatio-temporal*) häiriönpoistoa. Esimerkiksi Wolfstein: Youngblood -videopelissä käytetään enimmillään 16:lta kehykseltä kerättyä 16:ta näytettä eli 256:ta näytettä pikseliä kohden häiriön poistamiseksi heijastuksista. Aineistossa mainittuja häiriönpoistomenetelmiä ovat luvussa 4.4 esitelty SVGF-menetelmä Fortnite-videopelissä, sekä AMD:n tila-ajallinen häiriönpoisto kokoelma FidelityFX™ Denoiser ja NVIDIA:n NRD (NVIDIA Real-Time Denoisers) häiriönpoisto kirjaston tila-ajalliset ReBLUR ja ReLAX[10] menetelmät. Sen lisäksi, että aineiston videopeleissä tyypillisesti näytteitä yhdistellään sekä spatiaalisesti että ajallisesti, on aineiston videopeleissä tyypillistä, että häiriönpoisto suoritetaan erikseen valon eri ilmenemismuodoille. Aineiston videopeleissä häiriönpoistoon käytettävä aika on samaa luokkaa varsinaiseen säteenseurantaan käytettävän ajan kanssa.

okkluusiosta, otetaan kontaktivarjoissa valon suunta huomioon. Näin ollen kontaktivarjoilla voidaan korostaa pieniä yksityiskohtia, kuten esimerkiksi ei-konveksin geometrian itseensä aiheuttamia varjoja.

10. NRD ReLAX on SVGF-menetelmän paranneltu versio (“NVIDIA Real-Time Denoisers (NRD)” 2021).

## 7.2 Säteenseurannan optimointi

Muita aineistossa laajasti esiintyviä aiheita säteenseurannan hyödyntämiseen videopeleissä ovat luvussa 5.2 esitetyn säikeiden eriytymisen hallinta, sekä kiihdytysrakenteen ylläpitäminen. Varjostinkutsujen eriytymisen hallitsemiseksi Control ja Far Cry 6 -videopeleissä käytetään kaikille materiaaleille yhtenäistä lähin osuma -varjostinta heijastusten kuvaamiseen. Myös Fortnite-videopelissä käytetään yksittäistä lähin osuma -varjostinta, mutta sen sijaan, että varjostimessa suoritettaisiin valaistuslaskenta, suoritetaan valaistuslaskenta erillisissä varjostimissa. Pelissä ennen erillisten varjostimien kutsumista pikselit lajitellaan materiaalien perusteella 64 x 64 tiileissä varjostinkutsujen eriytymisen vähentämiseksi. Wolfstein: Youngblood -videopelissä puolestaan lähin osuma -varjostimien määrä on rajattu viiteen. Lisäksi Control ja Fortnite -videopeleissä ohivarjostin hyvin pelkistetty, kun taas Far Cry 6 -videopelissä ohivarjostin on tyhjä. Käytettävien varjostimien lukumäärän vähentämisen ohella muita tapoja vähentää varjostinkutsujen eriytymistä, sekä säteiden seuraamisen eriytymistä, on rajoittaa seurattavien säteiden enimmäispituutta, sekä yhtenäistää viereisten säteiden suuntia, jolloin todennäköisyys sille, että säteet matkaavat kiihdytysrakenteessa samaa reittiä ja leikkaavat saman geometrian kanssa kasvaa. Hitman 3 ja Far Cry 6 -videopeleissä heijastuksia varten käytetään yksinkertaistettuja geometrioita, jolloin todennäköisyys sille, että viereisten pikselien normaalit ja siten heijastussäteet ovat samansuuntaiset, kasvaa. Lisäksi alhaisempi määrä primitiivejä vähentää säteenseurannassa suoritettavia leikkaustarkasteluja, jolloin säteenseuranta nopeutuu. Wolfstein: Youngblood -videopelissä puolestaan heijastussäteiden enimmäispituutta rajoitetaan jyrkästi materiaalin karheuden kasvaessa.

Varjostinkutsujen ja säteiden seuraamisen eriytymisen vähentämiseksi, säteenseurannan suorituskyvyn kasvattamiseksi, kuin myös ylätason kiihdytysrakenteen rakentamisen tai päivittämisen nopeuttamiseksi, tärkeä osa säteenseurantaa on tarpeettomien tai vaikutukseltaan mitättömien ilmentymien jättäminen pois ylätason kiihdytysrakenteesta. Siinä, missä rasterointitekniikkaa käytettäessä karsiminen on helppo suorittaa kameran kuvapyramidin ja geometrioille muodostettujen rajaavien tilavuuksien leikkaavuusmäärittelynä, ei säteenseurannassa karsintaa voida yhtä triviaalisti suorittaa, mikä aineistossa ilmenee erilaisina tapoina suorittaa karsinta. Far Cry 6 -videopelissä karsitaan ilmentymät ilmentymän ja virtuaalikameran välisen etäisyyden perusteella jättäen ylätason kiihdytysrakenteesta pois ne ilmen-

tymät, jotka ovat yli tietyn etäisyyden päästä virtuaalikamerasta. Myös Hitman 3, Control ja Metro Exodus -videopeleissä ilmentymiä karsitaan etäisyyden perusteella laajentaen kameran kuvapyramidia useampaan suuntaan (engl. *expanded frustum culling*). Metro Exodus pelissä lisäksi ilmentymiä karsitaan koon perusteella. Battlefield V, Wolfstein: Youngblood ja Fortnite -videopeleissä puolestaan karsiminen tapahtuu virtuaalikameran ja ilmentymän välisen projektioavaruuskulman mukaisesti, jolloin karsimiseen vaikutttaa ilmentymän koon lisäksi ilmentymän etäisyys virtuaalikamerasta.

Koska videopelin virtuaalimaailmat ovat tyypillisesti laajoja ja dynaamisia, ei kaikkien geometrioiden pohjatason kiihdytysrakenteita välttämättä ole mahdollista säilyttää muistissa koko ohjelman suorituksen ajan tai geometria muuttuu, jolloin pohjatason kiihdytysrakenne tulee joko päivittää tai rakentaa uudelleen. Aineiston videopeleissä kiihdytysrakenteiden muodostamisessa ja päivittämisessä hyödynnetään asynkronista laskentaa (engl. *asynchronous compute*)[11], jolloin maksimaalisen hyödyn saavuttamiseksi, samanaikaisesti näytönohjaimelle suoritukseen asetettavat työt tulee valita siten, että ne eivät käytä samoja resursseja näytönohjaimella[12]. Aineiston videopeleistä Battlefield V, Control, Far Cry 6 ja Hitman 3 -videopeleissä kiihdytysrakenteiden muodostaminen ja päivittäminen suoritetaan päällekkäin G-puskurin muodostamisen kanssa. G-puskurin muodostamisen lisäksi Battlefield V, Control ja Far Cry 6 -videopeleissä kiihdytysrakenteen muodostaminen suoritetaan päällekkäin varjokarttojen muodostamisen kanssa. Metro Exodus pelissä puolestaan kiihdytysrakenteiden muodostaminen suoritetaan päällekkäin näkymäavaruuden heijastusten kanssa.

11. Vulkan API ja DirectX API mahdollistavat komentolistoihin asettamisen samanaikaisesti useasta säikeestä eri näytönohjaimen jonoihin, jolloin asetettaessa komentolistat samanaikaisesti esimerkiksi grafiikka- ja laskentajonoihin, kykenevät nykyaikaiset näytönohjaimet limittämään, sekä samanaikaistamaan eri yksiköissä suoritettavien töiden suoritusta näytönohjaimella, mikä parantaa hyötysuhdetta.

12. Esimerkiksi NVIDIA:n näytönohjaimilla kiihdytysrakenteen muodostaminen käyttää liukulukuihin ja kokonaislukuihin erikoistuneita ytimiä matalalla suoritusteholla, jolloin se soveltuu rinnakkaistettavaksi hyvin G-puskurin ja varjokarttojen muodostamisten kanssa, jotka myös käyttävät samoja ytimiä tyypillisesti matalalla suoritusteholla tai DLSS:n kanssa joka käyttää Tensor ytimiä (“Advanced API Performance: Async Compute and Overlap” 2022).

# 8 Kirjallisuuskatsaus

Siinä, missä edellisessä luvussa tarkasteltiin laitteistokiihdytteisen säteenseurannan hyödyntämistä nykyisissä reaaliaikaisissa sovelluksissa, on tämän luvun tarkoituksena esitellä säteenseurannan nykytutkimusta kirjallisuuskatsauksen avulla. Kirjallisuuskatsauksen tarkoituksena on esitellä laajasti tutkimusta säteenseurannan ympäriltä reaaliaikaisessa renderöinnissä, mistä syystä tämän tutkielman puitteissa kirjallisuuskatsaus rajataan ACM (Association for Computing Machinery) Digital Library -tietokantaan. ACM on vuonna 1947 perustettu niin kutsuttu oppineiden yhteisö tietotekniikan tutkimuksen ja opetuksen edistämiseen (ACM 2023). Yhteisön ylläpitämä digitaalinen kirjasto ACM Digital Library kattaa yli 3 miljoona tietotekniikan julkaisua vuodesta 1951 alkaen. Tietokonegrafiikan ja erityisesti tämän tutkielman kannalta merkittäviä ACM Digital Library -tietokannasta löytyviä lähteitä ovat The Proceedings of the ACM in Computer Graphics and Interactive Techniques (PACMCGIT) ja ACM Transactions on Graphics (TOG) journaalit[1]. PACMCGIT on vuodesta 2018 lähtien julkaistu journaali kattaen ACM SIGGRAPH:in (Special Interest Group on GRAPHics and Interactive Techniques) organisoimien High Performance Computing (HPG), Interactive 3D Graphics and Games (I3D) ja Symposium on Computer Animation (SCA) konferenssien julkaisuista tehtyjä journaaliartikkeleita. TOG puolestaan on vuodesta 1982 lähtien julkaistu vertaisarvioitu journaali sisältäen artikkeleita monipuolisesti tietokonegrafiikan eri osa-alueilta. Marraskuun ja joulukuun vaihteessa julkaistava TOG journaalin erikoisjulkaisu kattaa SIGGRAPH Asia konferenssin julkaisuja.

Kirjallisuuskatsauksen aineiston keruuta varten ACM Digital Library -tietokantaan kohdistettiin 4.9.2022 kysely:

```
Title:("real-time")
AND Fulltext:("path tracing" OR "ray tracing" OR
              "path-traced" OR "ray-traced")
OR Abstract:("real-time" AND ("path tracing" OR "ray tracing" OR
                              "path-traced" OR "ray-traced")),
```

1. Tutkielman kannalta ACM Digital Library -tietokannan ulkopuolelle jäävistä lähteistä merkittvä lähde on Wiley Online Library -tietokannasta löytyvä Wileyn Computer Graphics Forum journaali.

| Lähde | Julkaisu | Kategoria |
|---|---|---|
| Moreau, Pharr ja Clarberg 2019 | HPG | N: Valohierarkia |
| Andersson ym. 2019 | HPG | N: Korkea kehysnopeus |
| Peters ja Dachsbacher 2019 | PACMCGIT | N: Pinta-alallinen valonlähde |
| Lin ja Yuksel 2019 | PACMCGIT | N: Valohierarkia |
| Koskela ym. 2019 | TOG | H: Lineaariregressio |
| Xu, Ren ja Wu 2019 | SIGGRAPH Asia | H: Filtteröinti |
| Lin ja Yuksel 2020 | PACMCGIT | N: Valohierarkia |
| Tatzgern ym. 2020 | I3D | N: Valohierarkia |
| Bitterli ym. 2020 | TOG | N: Uudelleenkäyttö |
| Lee ja Liktor 2020 | SIGGRAPH Asia | M: Laitteisto/ajuri |
| Peters 2021 | TOG | N: Pinta-alallinen valonlähde |
| Müller ym. 2021 | TOG | N: Uudelleenkäyttö |
| Liu ym. 2021 | MICRO | M: Laitteisto/ajuri |
| Xu ym. 2022 | PACMCGIT | N: Kontaktivarjot |
| Oberberger, Chajdas ja Westermann 2022 | PACMCGIT | H: Liike-epätarkkuus |
| Thomas ym. 2022 | PACMCGIT | H: Neuroverkko, ylösskaalaus |
| Boissé 2021 | SIGGRAPH Asia | N: Uudelleenkäyttö |

Taulukko 10. Kirjallisuuskatsauksen aineisto julkaisu ajankohdan mukaisessa järjestyksessä. Kategoriat: näytteistäminen (N), häiriönpoisto (H) ja muut (M). MICRO = IEEE/ACM International Symposium on Microarchitecture.

jonka lisäksi hakutulokset rajattiin julkaisuvuoden perusteella vuodesta 2019 alkaen ja sisällyttämällä tuloksiin ainoastaan tutkimusartikkelit (engl. *research article*) jättäen pois esimerkiksi SIGGRAPH Talk ja SIGGRAPH Poster julkaisut. Kyselyn tavoitteena on tiukasti rajata kyselyyn täsmäävät artikkelit käsittelemään reaaliaikaista säteenseurantaa tai polunseurantaa[2]. Kysely antaa 41 artikkelia, joista 19 artikkelia käsittelee laitteistokiihdytteisen säteenseurannan hyödyntämistä reaaliaikaisessa renderöinnissä tai säteenseurannan yhteydessä suoritettavaa reaaliaikaista häiriönpoistoa[3]. Reaaliaikaisuuden hyväksymiskriteerinä käytetään 30 kehystä sekunnissa, jolloin koko kehyksen tuottamiseen kulunut aika tulee menetelmää hyödyntäen jäädä alle 34 millisekuntiin. Niiden artikkeleiden kohdalla, joissa ei ilmoiteta koko kehykseen kulunutta aikaa, vaan ainoastaan menetelmään kulunut aika, arvioidaan menetelmän soveltuvuus reaaliaikaiseen renderöintiin huomioimalla menetelmään kulunut aika, kuin myös itse menetelmän luokittelu. Reaaliaikaisuuden hyväksymiskriteerin perusteella artikkelit (Lin, Wyman ja Yuksel 2021) ja (Lin ym. 2022) jätettiin pois kirjallisuuskatsauksesta[4], jolloin kirjallisuuskatsaukseen sisällytettiin kaikkiaan 17 artikkelia.

Kirjallisuuskatsaukseen sisälletyt artikkelit esitetään taulukossa 10. Aineiston 17 artikkelia muodostuu kymmenestä journaaliartikkelista ja seitsemästä konferenssiartikkelista. Aineiston artikkelit kategorisoitiin niissä esiteltävien menetelmien perusteella kolmeen pääkategoriaan: näytteistämiseen (N), häiriönpoistoon (H) ja muuhun (M). Artikkeleista 11 kategorisoitiin näytteistämiseen, neljä häiriönpoistoon ja kaksi artikkelia muu kategoriaan. Näytteistäminen kategoriaan luokitellut artikkelit esitellään luvussa 8.1, kun taas häiriönpoisto kate-

2. Kysely saattaa jättää pois esimerkiksi häiriönpoistoa käsitteleviä artikkeleita, jotka eivät varsinaisesti säteenseurantaan liity, mutta joita voitaisiin myös säteenseurannan yhteydessä hyödyntää.

3. Pois jätettyjä artikkeleita ovat muun muassa säteenseurannan hyödyntämistä renderöinnin ulkopuolella käsittelevät artikkelit, kuin myös CPU:lla suoritettavan säteenseurannan viitekehyksiä ja hajautettuun laskentaan, kuten pilvilaskentaan, soveltuvia renderöintiarkkitehtuureja esittelevät artikkelit. Lisäksi aineistosta pois jätettiin kaikki ne reaaliaikaiseen renderöintiin liittyvät artikkelit, joiden esittelemissä menetelmissä ei suoraan hyödynnetä säteenseurantaa osana menetelmää tai käytetä säteenseurannalla tuotettua dataa menetelmän syötteenä.

4. Molemmissa artikkeleissa (Lin, Wyman ja Yuksel 2021) ja (Lin ym. 2022) kehyksen tuottamiseen kulunut aika artikkeleissa esitellyissä tapauksissa vaihtelee tyypillisemmin 30 ja 80 millisekunnin välillä, jolloin menetelmät eivät kaikissa tapauksissa täytä asetettua reaaliaikaisuuden hyväksymiskriteeriä. Molemmat artikkelit käsittelevät luvussa 8.1.2 esiteltävää ReSTIR-menetelmää, jonka yhteydessä myös molemmat artikkelit mainitaan, mutta artikkelien esittelemien menetelmien tarkempi esittely tutkielmassa sivuutetaan.

goriaan luokitellut artikkelit luvussa 8.2 ja muu kategoriaan luokitellut artikkelit luvussa 8.3. Molemmat muu kategoriaan luokitellut artikkelit käsittelevät laitteiston, ajurin ja renderöintiohjelmointirajapinnan muutoksia.

## 8.1 Näytteistäminen

Kuten luvussa 4.3 esiteltiin, voidaan erilaisilla näytteistämisen menetelmillä vähentää Monte Carlo -estimaattorin varianssia kasvattamatta näytteiden lukumäärää. Reaaliaikaisessa renderöinnissä, missä näytteiden lukumäärä on tyypillisesti hyvinkin alhainen, korostuu näytteiden tehokas hyödyntäminen. Tämä huolimatta siitä, että näytteistämisen monimutkaistaminen todennäköisesti vähentää näytteiden lukumäärää entisestään. Siinä, missä epäsuoran valaistuksen ratkaiseminen on usein laskennallisesti vaativaa, voi myös suoran valaistuksen ratkaiseminen osoittautua vaativaksi virtuaalimaailmassa, joka koostuu sadoista tai tuhansista valonlähteistä. Kuten suoran valaistuksen yhtälöstä 2.24 huomataan, vaatii suoran valaistuksen yhtälön ratkaiseminen kaikkien valonlähteiden pinta-ala-alkioiden läpikäymistä, jolloin valonlähteiden peittäessä ratkaistavan pinta-ala-alkion puoliavaruuden, tulee pahimmassa tapauksessa ratkaistavaksi puoliavaruus kokonaisuudessaan. Varsinkaan reaaliaikaisessa renderöinnissä ei kaikkien valonlähteiden vaikutusta suoraan valaistukseen ole tällöin mahdollista ratkaista, jolloin puhutaan monivalo-ongelmasta (engl. *many-lights problem*). Tällöin tehtävänä on löytää pinta-ala-alkiolle merkityksellisemmät valonlähteet, joita näytteistää. Ideaalisesti näytteistettävien valonlähteiden määräytyminen tapahtuu stokastisen prosessin kautta harhattomasti, mutta toisaalta vaikutukseltaan oletetusti pieni harha voidaan menetelmästä ja sovelluskohteesta riippuen sallia esimerkiksi suorituskyvyn parantamiseksi. Kuten luvussa 7 huomattiin, lähestytään luvun aineiston useammassa videopeleissä monivalo-ongelmaa rajaamalla suuri osa säteenseurannalla kuvatuista valon ilmenemismuodoista koskemaan yksittäistä valonlähdettä, kuten aurinkoa tai kuuta, sen sijaan, että monivalo-ongelma todellisuudessa ratkaistaisiin. Sen lisäksi, että monivalo-ongelma on läsnä suoran valaistuksen ratkaisemissa, on se läsnä myös epäsuoran valaistuksen ratkaisemisessa, kun epäsuoran valaistuksen kuvaaminen tapahtuu virtuaalisten pistevalojen avulla[5].

5. Kuten luvussa 3.1 esitellään, ratkaistaan epäsuora valaistus virtuaalisten pistevalojen avulla samoin kuin suora valaistus ei-virtuaalisilta pistevaloilta.

Aineiston näytteistämistä käsittelevissä artikkeleissa monivalo-ongelmaa käsitellään niin suoran valaistuksen kuvaamisen, kuin myös virtuaalisten pistevalojen avulla tapahtuvan epäsuoran valaistuksen kuvaamisen yhteydessä. Aineiston artikkeleissa esiintyviä ratkaisuja monivalo-ongelmaan ovat aliluvussa 8.1.1 esiteltävät hierarkkiset rakenteet valonlähteiden tallentamiseen eli niin kutsutut valohierarkiat, joiden lisäksi aliluvussa 8.1.2 esiteltävää ReSTIR-menetelmää (Bitterli ym. 2020) voidaan käyttää monivalo-ongelman ratkaisemiseksi. ReSTIR-menetelmän ohella aliluvussa 8.1.2 esitellään neuroverkkoa hyödyntävä menetelmä useamman kimmokkeen epäsuoran valaistuksen kuvaamiseen. Muita aineistossa käsiteltäviä näytteistäminen kategoriaan luokiteltuja aiheita ovat aliluvussa 8.1.3 esiteltävät menetelmät pinta-alallisten valonlähteiden näytteistämiseen, kontaktivarjojen kuvaamiseen, sekä korkean kehysnopeuden renderöintiin.

### 8.1.1 Valohierarkiat

Kuten edellä mainittiin, ovat valohierarkiat yksi monivalo-ongelman ratkaisuista. Aineiston artikkeleista neljässä esitetään valohierarkia monivalo-ongelman ratkaisemiseksi. Valohierarkialla käsitetään hierarkkinen, usein rajaavien tilavuuksien hierarkia, johon virtuaalimaailman valonlähteet tallennetaan. Reaaliaikaisessa renderöinnissä valohierarkian tulee mahdollistaa hierarkian nopea muodostaminen tai päivittäminen, sekä nopea pinta-ala-alkiolle suotuisien näytteistettävien valonlähteiden löytyminen. Menetelmästä riippuen näytteistettävä valonlähde voi olla yksittäinen virtuaalimaailman valonlähde tai useampien virtuaalimaailman valonlähteiden yhdistelmä, jolloin yksittäinen näyte muodostaa approksimaation useamman valonlähteen vaikutuksesta. Tyypillisesti puumaisissa hierarkioissa näytteistettävien valonlähteiden löytäminen perustuu stokastiseen prosessiin painottaen hierarkian solmuja, jolloin valohierarkia voidaan nähdä eräänlaisena hierarkkisena painotettuna näytteistämisenä. Useasti käytetty painotus hierarkian solmuille on perustaa painotus approksimaatioon suoran valaistuksen yhtälöstä ilman näkyvyystermiä, jolloin todennäköisyys määräytyy valonlähteen tai valonlähteiden valovoimakkuuksien, sekä valonlähteen tai valonlähteiden ja pinta-ala-alkion välisen etäisyyden ja suuntien perusteella.

Vaikka monivalo-ongelma on haaste myös suoran valaistuksen ratkaisemisessa, aineiston valohierarkioita käsittelevissä artikkeleissa erityisesti huomiota saa virtuaalisten pistevalo-

jen avulla tapahtuva epäsuoran valaistuksen approksimointi. Aineiston artikkeleissa virtuaaliset pistevalot luodaan joka kehyksellä, mikä tarkoittaa myös valohierarkian uusimista joka kehyksellä. Lisäksi virtuaalisia pistevaloja luodaan tyypillisesti kymmeniä- tai satojatuhansia, mikä asettaa merkittävän haasteen valohierarkian muodostamiselle kuin myös suotuisien näytteistettävien valonlähteiden löytymiselle. Aineiston artikkeleissa (Lin ja Yuksel 2019) ja (Tatzgern ym. 2020) esitetyt valohierarkiat soveltuvat yksinomaan pistevalojen ja virtuaalisten pistevalojen tallentamiseen, joiden lisäksi artikkelissa (Lin ja Yuksel 2020) esitelty valohierarkian käyttö esitellään epäsuoran valaistuksen approksimointiin virtuaalisten pistevalojen avulla. Myös artikkelin (Moreau, Pharr ja Clarberg 2019) valohierarkiaa voidaan käyttää virtuaalisille pistevaloille, mutta paremmin se soveltuu pinta-alallisten valonlähteiden, kuten emissoivien kolmioverkkojen tallentamiseen. Alla esitellään valohierarkioita käsittelevät aineiston artikkelit.

Artikkelissa (Moreau, Pharr ja Clarberg 2019) esitellään muutoksia artikkelissa (Conty Estevez ja Kulla 2018) esiteltyyn ATS (Adaptive Tree Splitting) valohierarkiaan mahdollistaen valohierarkian muodostaminen ja päivittäminen reaaliajassa, mikä puolestaan mahdollistaa dynaamisten valonlähteiden sisällyttämisen hierarkiaan. Alkuperäisessä kuin myös uudistetussa menetelmässä valonlähteet tallennetaan binääriseen rajaavientilavuuksien hierarkiaan, johon valonlähteet sijoittautuvat valonlähteen suunnatun avaruuskulman ja sijainnin perusteella. Hierarkiassa lehtisolmuihin tallennetaan yksittäiset valonlähteet, kun taas hierarkian sisäsolmuihin tallennetaan niin kutsutut valoklusterit (engl. *light cluster*). Valoklusteri muodostuu kyseisen solmun alipuiden lehtisolmujen sisältämille valonlähteille muodostetusta rajaavasta tilavuudesta, sekä valonlähteiden yhdistetystä suunnatusta avaruuskulmasta ja valovoimakkuuksien summasta. Pinta-ala-alkiolla näytteistettävien valonlähteiden löytämiseksi menetelmässä hierarkiaa matkataan stokastisesti painottaen hierarkian solmuja solmuun tallennetun suunnatun avaruuskulman ja pinta-ala-alkion suuntien, rajaavan tilavuuden keskipisteen ja pinta-ala-alkion välisen etäisyyden, kuin myös solmuun tallennetun valovoimakkuuden mukaisesti. Koska ylhäällä hierarkiassa eli lähellä juurisolmua valoklustereiden yhdistetyt avaruuskulmat ovat tyypillisesti suuria, saattaa edellä kuvattu alipuiden painotus johtaa mielivaltaisiin valintoihin ylhäällä hierarkiassa johtaen epäsuotuisien valonlähteiden näytteistämiseen. Tämän estämiseksi menetelmissä hierarkian matkaaminen voidaan hajaut-

taa (engl. *split*)[6], jolloin hierarkiassa matkaamista jatketaan väliaikaisesti molempien haarojen osalta. Hajautuminen määräytyy valoklusterin sisältämien valonlähteiden valovoimakkuuksien ja etäisyyksien varianssin perusteella, jolloin varianssin ollessa suurta, jatketaan molempia haaroja tilapäisesti. Lopullinen näytteistäminen menetelmissä suoritetaan lehtisolmujen valonlähteistä.

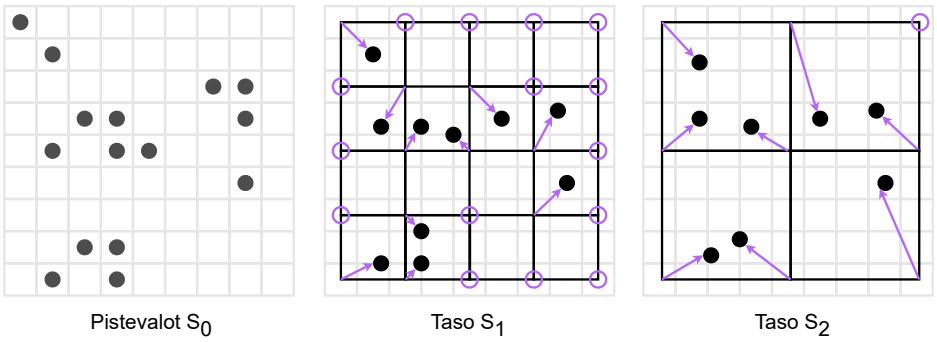
Siinä, missä alkuperäinen AT-menetelmä kuvataan yksitasoisena hierarkiana, ehdotetaan artikkelissa (Moreau, Pharr ja Clarberg 2019) valohierarkian jakamista pohja- ja ylätason rajaavien tilavuuksien hierarkioihin eli kiihdytysrakenteisiin. Pohjatason kiihdytysrakenteisiin tallennetaan valonlähteet kokonaisuudessaan eli esimerkiksi kolmioverkko, joka sisältää emissoivia kolmioita. Ylätason kiihdytysrakenteessa pohjatason kiihdytysrakenteet sijoittautuvat lehtisolmuihin. Matkaaminen hierarkioissa eli ylätason kiihdytysrakenteessa, kuin myös pohjatason kiihdytysrakenteissa suoritetaan painottaen alipuita samoin kuin alkuperäisessä ATS-menetelmässä. Kaksitasoisen valohierarkian hyöty tulee siitä, että vain animoitu pohjatason kiihdytysrakenne, sekä ylätason kiihdytysrakenne pohjatason kiihdytysrakenteiden muuttuessa tai liikkuessa tulee päivittää sen sijaan, että valohierarkia rakennettaisiin kokonaan uudelleen. Artikkelissa esitellään kiihdytysrakenteiden nopea päivittäminen näytönohjaimella, kun taas kiihdytysrakenteiden uudelleenrakentaminen suoritetaan CPU:lla[7]. Artikkelissa kuvatun valohierarkian voidaan katsoa soveltuvan erityisesti monimutkaisempien dynaamisten emissoivia kolmioita sisältävien kolmioverkkojen tallentamiseen[8].

Artikkelissa (Lin ja Yuksel 2019) puolestaan esitellään epäsuoran valaistuksen laskenta reaaliajassa virtuaalisten pistevalojen avulla hyödyntäen artikkelissa (Yuksel ja Yuksel 2017) eireaaliaikaisen renderöinnin yhteydessä esiteltyä hierarkkista valaistushilaa (engl. *lightning grid*). Valaistushila on kolmiulotteinen säännöllinen hilarakenne kuvattuna usealla tarkkuustasolla siten, että jokaisella tasolla kuvataan virtuaalimaailman kaikki pistevalot kokonaisuu-

6. Tästä tulee menetelmän nimi.

7. Artikkelissa esitetään pohjatason ja ylätason kiihdytysrakenteiden päivittäminen näytönohjaimella mahdollistaen varhaisen säteenseurannan aloittamisen, kun taas CPU:lla suoritetaan ylätason kiihdytysrakenteen uudelleenrakentaminen asynkronisesti edellisen kehyksen datasta.

8. Kaksitasoisuuden tuoman hyödyn voidaan kuvitella kasvavan valonlähteiden kolmioverkkojen kompleksisuuden myötä, jolloin pistevaloille käytettynä hierarkian kaksitasoisuus ei tarjoa hyötyä, mikä on havaittavissa myös artikkelin (Lin ja Yuksel 2020) vertailussa.




Kuvio 39. Valaistushilan (Yuksel ja Yuksel 2017) syöte $S_0$, sekä kaksi alinta tasoa $S_1$ ja $S_2$.

dessaan, mutta eri tarkkuuksilla. Näin ollen valaistushila mahdollistaa pistevalojen näytteistämisen eri tarkkuuksilla eli eri tasoilta esimerkiksi pinta-ala-alkion ja pistevalojen välisten etäisyyksien perusteella. Valaistushilan syötteenä $S_0$ toimivat pistevalot, kun taas valaistushilan alimmalla eri korkeimmalla tarkkuustasolla $S_1$ ja sitä ylemmillä tasoilla solun sisältämien pistevalojen vaikutus interpoloidaan hilan vertekseihin[9] muodostaen hilan vertekseille niin kutsuttuja valaistuskeskittymiä (engl. *illumination center*). Valaistuskeskittymä tallennetaan pistevalojen valovoimakkuuksien summana, sekä valovoimakkuuksien mukaisesti painotettuna poikkeamana verteksin sijainnista. Jokaiseen hilan verteksiin voidaan tallentaa enimmillään[10] yksi valaistuskeskittymä, jolloin yhdessä hilan solussa voi enimmillään olla kahdeksan valaistuskeskittymää. Solujen mittojen kaksinkertaistuessa tasojen välillä hierarkiassa ylöspäin mentäessä, laskee solujen lukumäärä kahdeksasosaan alempaan tasoon nähden. Hierarkian ylimmällä eli karkeimmalla tarkkuustasolla on vain yksi solu ja siten enimmillään kahdeksan valaistuskeskittymää. Valaistushila esitetään kuviossa 39.

Siinä, missä alkuperäisessä artikkelissa valaistushila kuvaillaan ei-reaaliaikaisen renderöinnin yhteydessä, esitellään artikkelissa (Lin ja Yuksel 2019) valaistushilan reaaliaikainen muodostaminen ja epäsuoran valaistuksen kuvaaminen hilaan sisällytettyjen virtuaalisten pistevalojen avulla. Reaaliaikainen valaistushilan muodostaminen perustuu hilan ensimmäi-

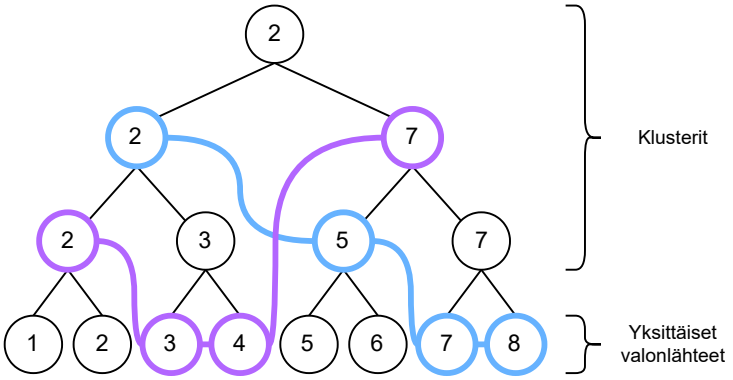
9. Näin ollen yksittäisen pistevalon vaikutus voi jakaantua useampaan hilan verteksiin.

10. Kaikkiin vertekseihin ei välttämättä tallenneta valaistuskeskittymää, mikä voi olla tyypillisitä virtuaalisten pistevalojen yhteydessä virtuaalisten pistevalojen sijoittuessa virtuaalimaailman geometrioiden pinnoille. Toisaalta hilaan tallennettavien elementtien epätasainen jakautuminen voidaan yleisemminkin katsoa säännöllisen hilan haasteeksi.

sen tason $S_1$ käyttämiseen syötteenä sitä ylempien tasojen muodostamiseen todellisten pistevalojen sijasta. Artikkelissa epäsuoran valaistuksen kuvaaminen tapahtuu rasterointitekniikalla siten, että pinta-ala-alkiota lähellä olevia virtuaalisia pistevaloja näytteistetään hierarkiasta matalalla tasolla eli korkealla tarkkuudella ja kauempana pinta-ala-alkiosta sijaitsevia virtuaalisia pistevaloja korkeammalta tasolta eli matalammalla tarkkuudella, jolloin approksimaatio epäsuorasta valosta muodostetaan eri tasoilta kerättyjen näytteiden painotettuna summana. Epäsuoran valaistuksen kuvaaminen tapahtuu näytteistäen alimmillaan eli tarkkuudeltaan korkeimmillaan tason $S_1$ valaistuskeskittymiä, jolloin todellisia virtuaalisia pistevaloja ei näytteistetä ollenkaan. Epäsuoran valaistuksen kuvaamisen yhteydessä valaistushilasta stokastisesti valitaan pinta-ala-alkion ja valaistuskeskittymän välisen etäisyyden ja valaistuskeskittymän valovoimakkuuden mukaisesti painottaen valaistuskeskittymiä epäsuorien varjojen kuvaamiseksi. Epäsuorien varjojen kuvaamiseen käytetään puolestaan säteenseurantaa ja näytteistettävät valaistuskeskittymät valitaan alimmillaan eli tarkkuudeltaan korkeimmillaan tasolta $S_2$. Koska epäsuoran valaistuksen kuvaamisessa valaistuskeskittymien näkyvyyttä ei tarkasteta, on menetelmä altis valovuodoille samoin kuin luvussa 3.1 esitettyihin irradianssiluotaimiin perustuva epäsuoran valaistuksen kuvaaminen rasterointitekniikalla.

Artikkelissa (Lin ja Yuksel 2020) esitellään artikkelissa (Yuksel 2019) esitetyn ei-reaaliaikaiseen renderöintiin tarkoitetun stokastinen valoleikkaukset (engl. *light cuts*) menetelmän uudistaminen reaaliaikaiseksi menetelmäksi. Stokastinen valoleikkaukset menetelmä perustuu artikkelissa (Walter ym. 2005) esiteltyyn valoleikkaukset menetelmään. Valoleikkaukset menetelmässä valohierarkiana käytetään binääristä puurakennetta, jonka lehtisolmut koostuvat virtuaalimaailman valonlähteistä ja sisäsolmut alipuiden lehtisolmujen sisältämien valonlähteiden muodostamista klustereista. Sisäsolmuun tallennettava klusteri koostuu klusterin sisältämille valonlähteille muodostetusta minimaalisesta rajaavasta tilaavuudesta, valonlähteiden yhteenlasketusta valovoimakkuudesta, suunnattuja valonlähteitä käytettäessä valonlähteiden määräämästä suunnatusta avaruuskulmasta, sekä niin kutsutusta edustavasta valonlähteestä. Alkuperäisessä valoleikkaukset menetelmässä sisäsolmun klusterin edustavan valonlähteen valinta tapahtuu satunnaisesti sisäsolmun kahden alisolmun, sisä- tai lehtisolmun, edustavien valonlähteiden[11] välillä painottaen edustavia valonlähteitä niiden valovoimak-

11. Lehtisolmun edustavana valonlähteenä käytettään lehtisolmuun tallennettua valonlähdettä.




Kuvio 40. Kahdeksasta valonlähteestä koostuva valoleikkaukset (Walter ym. 2005) menetelmässä käytettävä hierarkia ja kaksi hierarkian valoleikkausta.

kuuksilla.

Näytteistämisessä alkuperäisessä valoleikkaukset (Walter ym. 2005) menetelmässä käytetään joko sisäsolmun klusterin edustavaa valonlähdettä[12] ja klusterin yhteenlaskettua valovoimakkuutta tai lehtisolmuissa sijaitsevia valonlähteitä. Näytteistettävät valonlähteet määräytyvät käytettävän valoleikkauksen perusteella. Valoleikkaus määritellään solmujen joukoksi, joka sisältää täsmälleen yhden solmun, sisä- tai lehtisolmun, jokaiselta polulta juurisolmulta lehtisolmuille[13]. Valoleikkaus on havainnollistettu kuviossa 40. Kuviossa turkoosilla ja violetilla esitetään kaksi erilaista valoleikkausta. Valoleikkaus määräytyy jokaiselle ratkaistavalle pinta-ala-alkiolle erikseen käyttäjän määrittelemän enimmäisvirheen ja sisäsolmun klusterille lasketun virheen ylärajan mukaisesti. Virheen yläraja klusterille lasketaan edustavan valon ja klusteriin sisällytettyjen valonlähteiden tuottamien suoran valaistuksen yhtälön approksimaatioiden erotuksen ylärajana. Tällöin, mikäli sisäsolmun klusterin virheen yläraja on alle käyttäjän määrittelemän sallitun enimmäisvirheen, sisällytään kyseinen sisäsolmu valoleikkaukseen. Muussa tapauksessa jatketaan alipuiden tutkimista, kunnes löydetään sisäsolmu, jonka klusterin enimmäisvirhe jää alle sallitun enimmäisvirheen tai päädytään lehtisolmulle. Näin ollen menetelmässä sallitun enimmäisvirheen toteutumista tarkastellaan alipuu kohtaisesti[14].

12. Tällöin valaistuslaskennassa käytetään muun muassa edustavan valon sijaintia ja näkyvyyttä pinta-ala-alkiolta.

13. Näin ollen valoleikkaus sisältää aina kaikkien valonlähteiden valovoimakkuuden.

14. Tällöin valoleikkauksen kokonaisvirhe voi merkittävästi vaihdella valoleikkauksien välillä.

Artikkelissa (Yuksel 2019) tuodaan alkuperäisen valoleikkaukset menetelmän huonot puolet esille, joista merkittävin on hierarkian muodostamisen yhteydessä valittavat edustavat valonlähteet. Erityisen ongelmallisia ovat korkealla hierarkiassa sijaitsevat edustavat valonlähteet, joilla on muita valonlähteitä merkittävästi suurempi todennäköisyys sisältyä valoleikkaukseen. Todennäköisyys kasvaa hierarkiassa ylöspäin mentäessä ja juurisolmun edustava valonlähde sisältyy aina valoleikkaukseen, mikä voidaan havaita myös kuviosta 40. Lisäksi, koska valoleikkauksen muodostuminen tapahtuu deterministisesti sallitun enimmäisvirheen mukaisesti, aiheuttaa se usein korrelaation valoleikkaukseen sisällytettävien edustavien valonlähteiden välille. Artikkelissa esitetyssä stokastinen valoleikkaukset menetelmässä sen sijaan, että käytettäisiin ennalta valittua edustavaa valonlähdettä, valitaan edustava valonlähde alipuiden lehtisolmujen valonlähteistä käyttäen painotettua näytteistystä. Painotus perustuu pinta-ala-alkion ja solmuun tallennetun rajaavan tilavuuden välisiin vähimmäis- ja enimmäisetäisyyksiin, pinta-ala-alkion normaaliin suhteessa solmun sijaintiin, sekä solmuun tallennettuun kokonaisvalovoimakkuuteen.

Artikkelissa (Lin ja Yuksel 2020) esitellään stokastinen valoleikkaus menetelmän uudistaminen reaaliaikaiseksi. Menetelmässä reaalikaisuus mahdollistuu käyttämällä hierarkiana täydellistää binääripuuta[15], mikä merkittävästi nopeuttaa niin hierarkian muodostamista kuin myös näytteistettävien valonlähteiden löytämistä. Artikkelissa esitellään valohierarkian nopea muodostaminen näytönohjaimella hyödyntäen Morton-käyrää (engl. *Lebesgue curve*, *z-order curve*)[16]. Toinen artikkelissa esitetty menetelmää nopeuttava uudistus on niin kutsuttu

15. Täydelliseksi binääripuuksi artikkelissa kutsutaan binääripuuta, jonka jokaisella sisäsolmulla on täsmälleen kaksi lapsisolmua ja puun lehtisolmut sijaitsevat samalla tasolla hierarkiassa. Tämä vaatii tekaistujen valonlähteiden, joiden valovoimakkuus on nolla, lisäämisen hierarkiaan, mikä tulee huomioida näytteistämisessä.

16. Valohierarkian nopea muodostaminen pohjautuu artikkelissa (Lauterbach ym. 2009) esiteltyyn LBVH (Linear Bounding Volume Hierarchy) menetelmään. LBVH-menetelmässä $2^k x 2^k x 2^k$ kokoisen hilan muodostamista varten tulee tuntea kaikkien valonlähteiden minimaalinen rajaava AABB, sekä jokaisen yksittäisen valonlähteen rajaava ABBB. Tällöin jokaiselle valonlähteelle voidaan muodostaa Mortonin koodi kvantisoimalla valonlähteen rajaavan AABB:n keskipisteen koordinaatit $k$-bittisiksi ja lomittamalla kvantisoidut koordinaatit $3k$-bittiseksi jonoksi. Morton koodien muodostamisen jälkeen valonlähteet lajitellaan Morton koodien mukaiseen järjestykseen. Puun muodostaminen tapahtuu lokeroimalla lajitellut valonlähteet rekursiivisesti bittien mukaisesti aloittaen merkittävimmästä bitistä. Menetelmän laskenta voidaan helposti rinnakkaistaa, joten se soveltuu erinomaisesti näytönohjaimella suoritettavaksi. Kattavasti menetelmiä rinnakkaiseen puun muodostamiseen esitellään artikkelissa (Lauterbach ym. 2009).

leikkauksien jakaminen (engl. *cut sharing*), jolla tarkoitetaan saman valoleikkauksen hyödyntämistä määritellyn kokoisissa pikselien muodostamissa tiilissä[17]. Artikkelissa menetelmä esitetään käytettäväksi epäsuoran valaistuksen approksimointiin virtuaalisten pistevalojen avulla, jolloin sekä hierarkia että virtuaaliset pistevalot luodaan jokaisella kehyksellä uudelleen.

Perustuen stokastiset valoleikkaukset menetelmään artikkelissa (Tatzgern ym. 2020) esitetään virtuaalisille pistevaloille tarkoitettu stokastinen korvaava puu (engl. *stochastic substitute tree*, *SST*) menetelmä. SST-menetelmää voidaan käyttää sekä diffuusi heijastusten kuvaamiseen, kuin myös sumeiden spekulaari[18] heijastusten kuvaamiseen. Siinä, missä stokastinen valoleikkaukset menetelmässä sisäsolmun sisältyessä valoleikkaukseen, suoritetaan näytteistäminen sisäsolmulle valittavalle valonlähteelle seuraten hierarkiaa lehtisolmulle, voidaan SST-menetelmässä näytteistäminen suorittaa suoraan sisäsolmun eli klusterin määräämälle normaalijakaumalle. Klusterin normaalijakauma määräytyy klusterin sisältämien valonlähteiden valovoimakkuuksilla painotettujen sijaintien varianssin perusteella[19]. Klusteriin sijoittautuminen tapahtuu virtuaalisen pistevalon sijainnin ja suunnan eli sen pinta-ala-alkion normaalin mukaisesti, jonka pinnalla virtuaalinen pistevalo sijaitsee. Hierarkian muodostamisen yhteydessä sisäsolmuun tallennetaan sisäsolmun klusterin soveltuvuus näytteistämiseen. Soveltuvuus riippuu sekä klusterin sisältämien virtuaalisten pistevalojen samansuuntaisuudesta että pistevalojen sijainnin varianssista klusterin normaalin suunnassa, jolloin näytteistettäviksi soveltuvat klusterit ovat tyypillisesti tasomaisia. Menetelmässä ei varsinaisesti suoriteta valoleikkauksia, vaan näytteistettäessä jokaista näytettä varten hierarkiaa matkataan juurisolmusta kohti lehtisolmuja painottaen alisolmuja samoin kuin stokastinen valoleikkaukset menetelmässä. Hierarkiassa matkaaminen diffuusi materiaalien yhteydessä py-

17. Valoleikkauksen uudelleenkäyttöä voidaan pitää perusteltuna sillä stokastinen valoleikkaukset menetelmässä, toisin kuin alkuperäisessä valoleikkaukset menetelmässä, näytteistettävä valonlähde valitaan stokastisesti valoleikkauksien sisäsolmuissa, jolloin näytteistettävät valonlähteet voivat poiketa toisistaan samaa valoleikkausta käytettäessä.

18. Menetelmä ei kykene tarkkoja peiliheijastuksia kuvaamaan, joiden kuvaamiseksi säteenseuranta soveltuu paremmin.

19. Tyypillisesti klusteri koostuu vierekkäisten ja saman suuntaisten kolmioverkon kolmioiden pinnalla sijaitsevista virtuaalisista pistevaloista, jolloin näytteistettävä piste valitaan varianssin mukaisesti kyseiseltä tasolta. Näin ollen virtuaalisten pistevalojen kuvitellaan muodostavan jatkuvan pinnan tasolla.

säytetään ensimmäiseen klusteriin, joka soveltuu näytteistettäväksi, kun taas spekulaari heijasteisten materiaalien yhteydessä hierarkiaa matkataan lehtisolmuille asti, jolloin näytteistäminen suoritetaan lehtisolmun sisältämältä virtuaaliselta pistevalolta. Näytteistäminen sisäsolmusta tapahtuu näytteistämällä sisäsolmun klusterin muodostaman normaalijakauman mukaisesti valikoitunutta pistettä käyttäen klusterin yhteenlaskettua valovoimakkuutta. Samoin kuin stokastinen valoleikkaukset menetelmässä, hierarkian muodostaminen suoritetaan näytönohjaimella käyttäen Morton-käyrää. Kenties parhaiten SST-menetelmä soveltuu virtuaalisten pistevalojen avulla epäsuoran valaistuksen kuvaamiseen diffuusi pinnoilta virtuaalimaailmassa, jossa geometriat sisältävät suuria tasomaisia osuuksia, jolloin näytteistäminen voidaan suorittaa korkealla hierarkiassa.

### 8.1.2 Laskennan uudelleenkäyttö

Siinä, missä valohierarkioilla voidaan ratkaista monivalo-ongelma, voidaan myös luvussa 4.3 esiteltyä RIS-menetelmää hyödyntämällä ratkaista monivalo-ongelma. Toisin kuin painotetussa näytteistyksessä, jossa approksimoitava funktio tulee tuntea ainakin osittain, mahdollistaa RIS-menetelmä funktion approksimoinnin ilman, että todellista funktiota tunnetaan, mikä tekee RIS-menetelmästä vaihtoehdon monivalo-ongelman ratkaisemiseksi[20]. Verrattuna valohierarkioihin RIS-menetelmä ei vaadi monimutkaisten hierarkioiden ylläpitoa, joiden ylläpito dynaamisessa virtuaalimaailmassa saattaa viedä suuren osan kehykseen käytettävästä ajasta. Aineistossa RIS-menetelmää hyödynnetään osana luvussa esiteltävää kandidaattinäytteitä ajallisesti ja spatiaalisesti uudelleen käyttävää ReSTIR-menetelmää. ReSTIR-menetelmän ohella luvussa esitellään neuroverkkoa hyödyntävä menetelmä useamman kimmokkeen epäsuoran valaistuksen kuvaamiseksi. Siinä, missä häiriönpoiston yhteydessä näytteitä yhdistellään usein spatiaalisesti ja ajallisesti osana jälkikäsittelyä, katsotaan tutkielmassa näytteistämisen menetelmiksi ne laskentaa uudelleen käyttävät menetelmät, joissa lopullisten näytteiden sijasta uudelleen käytetään esimerkiksi näytteiden todennäköisyysjakaumia

20. Monivalo-ongelman yhteydessä virtuaalimaailman sisältäessä satoja tai tuhansia valonlähteitä ei useinkaan ole suoraa tapaa painottaa valonlähteitä suoran valaistuksen kuvaamisessa eli toisin sanoen approksimoitavaa funktiota ei sen kompleksisuuden vuoksi tunneta. Näin ollen painotettu näytteistäminen sellaisenaan soveltuu hyvin virtuaalimaailmoihin, jotka sisältävät korkeintaan muutamia valonlähteitä, kun taas RIS-menetelmää voidaan käyttää approksimoitavan funktion löytämiseksi monimutkaisemmassa virtuaalimaailmassa.

tai osaa näytteen polusta ennen lopullisen näytteen muodostamista[21]. Alla esitellään laskennan uudelleenkäyttöä käsittelevät aineiston artikkelit.

Artikkelissa (Bitterli ym. 2020) esitellään RIS-menetelmää, monipainotettua näytteistystä, sekä niin kutsuttua painotettua varastoivaa näytteistystä (engl. *weighted reservoir sampling*, *WRS*) ja näytteiden tila-ajallista uudelleenkäyttöä hyödyntävä ReSTIR (reservoir-based spatiotemporal importance resampling) -menetelmä monivalo-ongelman ratkaisemiseksi suoran valaistuksen kuvaamisessa. WRS on algoritmien perhe lukumäärältään $N$ näytteen valitsemiseen painotettujen näytteiden virrasta (engl. *stream*) painotusten mukaisesti säilyttäen muistissa eli varastoiden ainoastaan $N$ näyttettä ja tuntematta virran pituutta $M$. ReSTIR-menetelmässä varastoon näytteet valitaan riippumatta toisistaan eli takaisinpanolla (engl. *with replacement*). Tällöin jokaisen näytteen $X_i$ todennäköisyys tulla sisällytetyksi varastoon on

$$P(X_i) = \frac{w(X_i)}{\sum_{j=1}^{M} w(X_j)}, \tag{8.1}$$

missä $w(X)$ on näytteen $X$ painokerroin. Olettaen, että $N = 1$ ja virrasta ollaan kulutettu $m$ näytettä, sisällytetään tällöin näyte $X_i$ varastoon todennäköisyydellä

$$P(X_i) = \frac{w(X_i)}{\sum_{j=1}^{m} w(X_j)} \tag{8.2}$$

ja sitä seuraava näyte $X_{m+1}$ todennäköisyydellä

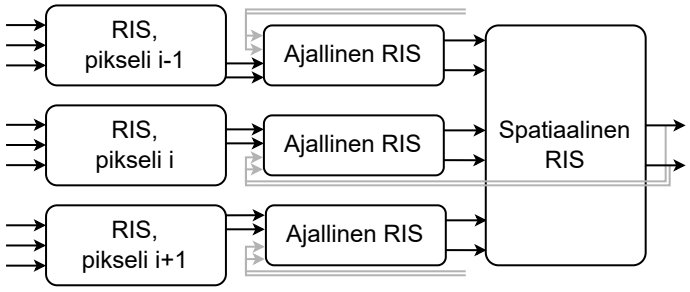
$$P(X_{m+1}) = \frac{w(X_{m+1})}{\sum_{j=1}^{m+1} w(X_j)}. \tag{8.3}$$

Tällöin näyte $X_{m+1}$ on halutun todennäköisyyden mukaisesti varastossa kuten on myös mikä tahansa edellinen näyte $X_i$[22]. ReSTIR-menetelmässä näytteiden virtana käytetään RIS-menetelmällä kerättyjä ja painotettuja kandidaattinäytteitä. Menetelmässä kandidaattinäytteet painotetaan suoran valaistuksen yhtälön mukaisesti ilman näkyvyystermiä.

21. Näin ollen myös virtuaaliset pistevalot voidaan tulkita laskennan uudelleenkäytöksi.

22. $P(X_i) = \frac{w(X_i)}{\sum_{j=1}^{M} w(X_j)} \left(1 - \frac{w(X_{m+1})}{\sum_{j=1}^{m+1} w(X_j)}\right) = \frac{w(X_i)}{\sum_{j=1}^{m+1} w(X_j)}$.

RIS-menetelmää käytettäessä hyviä tuloksia saavutetaan, kun kandidaattinäytteiden lukumäärä $M$ on korkea, mistä syystä ReSTIR-menetelmässä efektiivistä kandidaattinäytteiden lukumäärää kasvatetaan uudelleen käyttämällä kandidaattinäytteitä tila-ajallisesti kuviossa 41 esitetyllä tavalla. Ennen spatiaalista uudelleenkäyttöä menetelmässä suoritetaan kandidaattinäytteiden ajallinen uudelleenkäyttö pikseleittäin, jonka jälkeen varastoon sisällytetyille kandidaattinäytteille seurataan varjosäde[23]. Spatiaalisessa uudelleenkäytössä uudelleenkäytettävät pikselit valitaan satunnaisesti ennalta määrätyn etäisyyden sisältä painottaen tarkasteltavan pikselin kanssa syvyys- ja normaaliarvoiltaan samankaltaisia pikseleitä voimakkaammin. Sekä spatiaalinen, että ajallinen kandidaattinäytteiden uudelleenkäyttö tapahtuu varastoja yhdistämällä, jolloin edellisten kehysten varastot ja lähekkäisten pikselien varastot muodostavat uuden kandidaattinäytteiden virran, joista näytteistettävä näyte tai näytteet valitaan edellä kuvattua painotettua varastoivaa näytteistämismenetelmää hyödyntäen. Painona varastoon sisällytetylle kandidaattinäytteelle käytetään varastoon liitetyn kandidaattinäytteiden virran läpikäytyjen kandidaattinäytteiden painojen summaa $w_{sum} = \sum_{j=1}^{m+1} w(X_j)$, jolloin sen sijaan, että jokaiselle pikselille tallennettaisiin yksittäiset kandidaattinäytteet ja niiden painokertoimet, riittää menetelmässä ylläpitää jokaisella pikselillä varastoon sisällytettyjä kandidaattinäytteitä ja läpikäytyjen kandidaattinäytteiden painojen summaa.




Kuvio 41. ReSTIR (Bitterli ym. 2020) menetelmän toimintaperiaate. Harmaat nuolet tarkoittavat edelliseltä kehykseltä uudelleenkäytettäviä näytteitä.

Vaikka lähekkäisten ja syvyys- ja normaaliarvoiltaan samankaltaisten pikselien suoran valaistuksen ratkaisuavaruudet ovat tyypillisesti likimain samat, eivät ne useinkaan ole täysin identtiset, jolloin spatiaalinen kandidaattinäytteiden uudelleenkäyttö saattaa aiheuttaa har-

23. Jos valonlähteeseen ei ole näkyvyyttä, ei sen spatiaalista uudelleenkäyttöä mahdollisteta muilla pikseleillä ReSTIR-menetelmän harhallisessa versiossa.

han. Harha aiheutuu, mikäli varastoon sisällytetään spatiaalisen uudelleenkäytön yhteydessä lähekkäisen pikselin kandidaattinäyte, joka saa todellisuudessa arvokseen nolla tarkasteltavassa pikselissä tai estetään sellaisen lähekkäisen pikselin kandidaattinäytteen spatiaalinen uudelleenkäyttö, joka tarkasteltavassa pikselissä saisi nollasta poikkeavan arvon, perustuen kandidaattinäytteen nolla-arvoon alkuperäisessä pikselissä[24]. Artikkelissa ReSTIR-menetelmästä esitellään sekä harhaton että harhallinen versio. Siinä, missä harhallisessa versiossa ei spatiaalisesti uudelleen käytetä niitä kandidaattinäytteitä, joihin ei näytteen alkuperäisessä pikselissä ole näkyvyyttä, niin harhattomassa versiossa kaikkien spatiaalisesti uudelleenkäytettävien kandidaattinäytteiden näkyvyys tarkastetaan niitä hyödyntävässä pikselissä seuraamalla varjosäde, jonka lisäksi harhattomassa versiossa käytetään monipainotettua näytteistystä eri ratkaisuavaruuksista tulleiden näytteiden yhdistämiseksi. Tämä hidastaa menetelmää verrattuna harhalliseen versioon, jossa harha tummentaa valaistusta. Siinä, missä ReSTIR-menetelmä ei vaadi monimutkaisen hierarkian ylläpitämistä monivalo-ongelman ratkaisemiseksi[25], on ReSTIR-menetelmä kuten muutkin ajallisesti näytteitä uudelleen käyttävät menetelmät parhaimmillaan, kun virtuaalinäkymä on ajallisesti mahdollisimman vakaa.

ReSTIR-menetelmän alkuperäisen julkaisun jälkeen menetelmää on esitetty hyödynnettäväksi niin epäsuoran valaistuksen kuvaamisessa (Ouyang ym. 2021), kuin myös tilavuusrenderöinnissä[26] (engl. *volume rendering*) (Lin, Wyman ja Yuksel 2021) uudelleen käyttämällä kandidaattinäytteiden sijasta polkuja. Artikkelissa (Boissé 2021) puolestaan esitellään ReSTIR-menetelmän käyttö yhden kimmokkeen epäsuoran valaistuksen kuvaamiseen, mutta toisin kuin artikkelissa (Ouyang ym. 2021) esitellyssä epäsuoran valaistuksen kuvaamiseen tarkoitetussa ReSTIR-menetelmässä, spatiaalista uudelleenkäyttöä varten varastot tallenne-

24. Toisin sanoen harha johtuu valonlähteen näkyvyyden vaihtelusta lähekkäisillä pikseleillä. Esimerkiksi valonlähteen sijaitessa pikselin sisältämän pinta-ala-alkion normaalin määräämän puoliavaruuden sisällä, ei se välttämättä sijaitse lähekkäisen pikselin sisältämän pinta-ala-alkion normaalin määräämän puoliavaruuden sisällä tai näkyvyys valonlähteeseen vierekkäisillä pikseleillä voi vaihdella muun geometrian varjostaessa vain toista pikselin sisältämää pinta-ala-alkiota.

25. ReSTIR-menetelmää ei poissulje valohierarkian käyttöä, vaan valohierarkiaa voitaisiin käyttää alkuperäisten kandidaattinäytteiden muodostamiseen.

26. Tilavuusrenderöinti kattaa esimerkiksi osallistuvan median (engl. *participating media*), kuten sumun ja savun, renderöinnin. Osallistuvan median kuvaaminen säteenseurantaa käyttäen tapahtuu satunnaisesti hajaannuttamalla (engl. *scatter*) säde seuraamisen aikana suoraviivaisen etenemisen sijasta.

taan virtuaalimaailmassa adaptiiviseen hilarakenteeseen. Adaptiivisuudella tarkoitetaan hilan solun koon vaihtelua solun sijainnin ja virtuaalikameran välisen etäisyyden myötä, jolloin solujen koot kasvavat siirryttäessä etäämmäksi virtuaalikamerasta. Hilarakenteen solu, johon varasto tallennetaan tai luetaan, määräytyy sijainnista, sekä sijainnin ja virtuaalikameran ominaisuuksien avulla lasketusta solun koosta muodostettavan tiivisteen perusteella. Hilarakenteeseen tallennettaessa, kuin myös hilarakenteesta luettaessa uudelleenkäytön yhteydessä, sijaintia poikkeutetaan korrelaation vähentämiseksi[27]. Menetelmässä ajallinen uudelleenkäyttö puolestaan perustuu edellisellä kehyksellä muodostetun hilan käyttämiseen.

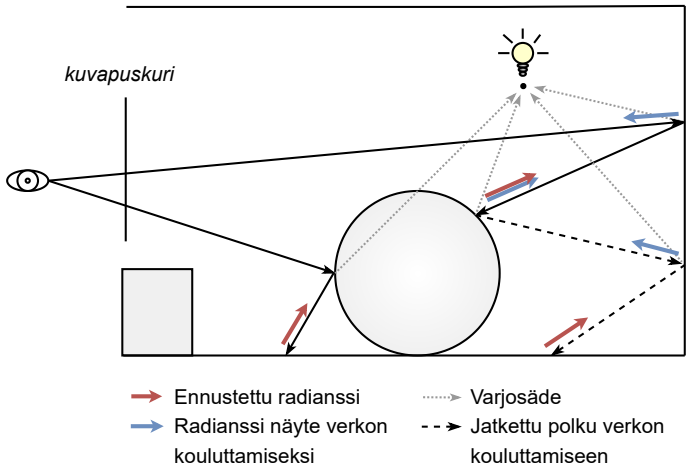
Siinä, missä alkuperäinen ReSTIR-menetelmä ei takaa approksimaation lähentymistä kohti funktion todellista arvoa tila-ajallisen näytteiden uudelleenkäytön myötä[28] eli menetelmä ei ole tarkentuva, esitellään artikkelissa (Lin ym. 2022) teoreettinen pohja GRIS (engl. *generalized resampled importance sampling*) RIS-menetelmän yleistämiseksi eri ratkaisuavaruuden näytteiden kuvaamiseksi uudelleenkäytettävässä ratkaisuavaruudessa (engl. *shift mapping*), mikä takaa approksimaation lähentymisen kohti funktion todellista arvoa. Samassa artikkelissa esitetään GRIS teoriaan pohjautuva tarkentuva ja harhaton ReSTIR polunseuranta (ReSTIR PT) -menetelmä useamman kimmokkeen epäsuoran valaistuksen kuvaamiseen kuin myös valon kulkeutumiseen läpinäkyvissä aineissa lähes reaaliaikaisesti.

Artikkelissa (Müller ym. 2021) esitellään useamman kimmokkeen epäsuoran valaistuksen diffuusi ja spekulaari heijastusten kuvaamiseen tarkoitettu monikerroksista perseptroniverkkoa hyödyntävä radianssin varastointimenetelmä (engl. *neural radiance caching*, *NRC*). NRC-menetelmä perustuu artikkelissa (Krivanek ym. 2005) esiteltyyn radianssin varastointimenetelmään (engl. *radiance caching*), joka puolestaan on artikkelissa (Ward, Rubinstein ja Clear 1988) esitellyn irradianssin varastointimenetelmän (engl. *irradiance caching*) yleistys kuvaamaan Lambertin diffuusin lisäksi spekulaari heijastuksia[29]. Irradianssin varastointimenetelmä perustuu oivallukseen, että siinä, missä suora valaistus saattaa muuttua merkittävästikin

27. Tämä häivyttää hilan solujen näkymistä kuvassa.

28. Menetelmässä näytteet ovat yleisesti korreloituneita eli näytteet ovat jossain määrin samoja kuin edellisellä kehyksellä ja lähekkäisillä pikseleillä, mikä hidastaa tai estää approksimaation lähentymisen kohti funktion todellista arvoa.

29. Radianssin varastointimenetelmä ei kuitenkaan sovellu peili- tai lähes peiliheijastuksien kuvaamiseen, vaan ne tulee kuvata esimerkiksi polunseurantamenetelmää hyödyntäen.




Kuvio 42. NRC menetelmän (Müller ym. 2021) toimintaperiaate.

viereisillä pinta-ala-alkiolla, epäsuora valaistus muuttuu diffuusi heijasteisilla pinnoilla huomattavasti hitaammin, mikä mahdollistaa laskennan uudelleenkäytön vierekkäisillä pinta-ala-alkiolla. Irradianssin varastointimenetelmässä, samoin kuin radianssin varastointimenetelmässä, epäsuoran valaistuksen kuvaaminen tapahtuu interpoloimalla pinta-ala-alkiota lähimpänä olevia varastoja. Uusi varasto luodaan, mikäli tarkasteltavaa pinta-ala-alkiota lähellä ei sijaitse varastoja. Menetelmissä varastot tallennetaan kasipuuhun (engl. *octree*) lähimpien varastojen löytämiseksi. Erona menetelmien välillä on, että radianssin varastointimenetelmässä kasipuuhun tallennetaan pisteeseen tietyistä suunnista saapuva radianssi sen sijaan, että tallennettaisiin pisteeseen saapuva irradianssi yksittäisenä arvona. Luvussa 3.1 esitelty irradianssitilavuusmenetelmä pohjautuu irradianssi varastointimenetelmään, mutta merkittävänä erona puolestaan näiden kahden menetelmän välillä voidaan pitää sitä, että irradianssi varastointimenetelmässä varastot sijaitsevat geometrioiden pinnoilla, kun taas irradianssitilavuusmenetelmässä luotaimet sijoittautuvat tyhjään tilaan, mikä tekee luotaimista paremman vaihtoehdon epäsuoran valaistuksen kuvaamiseen virtuaalimaailmassa, jossa geometriat ovat hyvin kompleksisia. Siinä, missä irradianssitilavuusmenetelmässä irradianssiluotaimien näytteistämisessä usein käytetään palloharmonisia funktioita, käytetään varastointimenetelmässä puolestaan puolipalloharmonisia funktiota (engl. *hemispherical harmonics*)[30].

30. Koska irradianssi varastointimenetelmässä varastot sijaitsevat geometrioiden pinnoilla, tulee irradianssi kuvata vain puoliavaruudesta.

Pohjautuen radianssin varastointimenetelmään, NRC-menetelmässä varastoidaan radianssia, mutta puolipalloharmonisten funktioiden vakioiden muodostamisen sijasta, radianssi varastoidaan yksittäiselle neuroverkolle, joka ennustaa radianssin annetulle sijainnille ja suunnalle. Neuroverkko koulutetaan ohjelman ajon aikana (engl. *online*) mahdollistaen dynaamiset ja monimuotoiset virtuaalimaailmat. Neuroverkon kouluttaminen tapahtuu suorittamalla neljä gradienttilaskeumaa kehyksellä, mikä sellaisenaan johtaa suureen vaihteluun kehyksien välillä, mistä syystä gradienttilaskeumien tuottamille painoille käytetään eksponentiaalista liikkuvaa keskiarvoa (engl. *exponential moving average*, *EMA*)[31]. Tappiona (engl. *loss*) menetelmässä käytetään heijastuvuusyhtälön 2.22 neliövirhettä. NRC-menetelmässä seurataan lyhyitä polkuja päättäen polut neuroverkon ennustamaan radianssiin[32], kuten esitetään punaisin nuolin kuviossa 42. Neuroverkon kouluttamiseksi pientä osaa[33] poluista jatketaan useamman kimmokkeen epäsuoran valaistuksen kuvaamiseksi päättäen myös jatketut polut neuroverkon ennustamaan radianssiin. Kuviossa neuroverkon kouluttamiseen käytettävä jatkettu polku esitetään sinisin nuolin. Syötteenä neuroverkolle viedään pinta-ala-alkion sijainti, avaruuskulma, pinnan normaali, sekä pinnan materiaalien ominaisuuksia muodostaen koodauksen jälkeen 64-ulotteisen syötteen[34]. Reaaliaikaisuuden mahdollistamiseksi artikkelissa esitellään neuroverkon toteutus yksittäisenä CUDA ytimenä ilman tekoälykehyksen tai -kirjaston käyttöä. Artikkelissa menetelmä esitellään käytettäväksi yhdessä ReSTIR-menetelmän kanssa.

31. Eksponentiaalista liikkuvaa keskiarvoa käytettäessä viimeisintä arvoa painotetaan voimakkaimmin ja viimeisintä arvoa aikaisempien arvojen paino laskee eksponentiaalisesti.

32. Polun seuraaminen lopetetaan polulla kohdattujen materiaalien karheuksien mukaan eli heijastus avaruuskulmien summan kasvaessa raja-arvon yli, jolloin mahdollinen neuroverkon ennustaman radianssin mahdollinen epätarkkuus ei näy kuvassa.

33. Artikkelissa esitellyssä toteutuksessa 3 %:ia poluista jatketaan verkon kouluttamiseksi.

34. Pinnan materiaaleista syötteenä viedään pinnan karheus, sekä diffuusi ja spekulaari heijastuvuudet. Korrelaation löytämisen helpottamiseksi syötteenä vietävät sijainti, avaruuskulma, pinnan normaali ja karheus koodataan useampi ulotteiseksi korrelaatioiden saattamiseksi lineaarisemmiksi. Koneoppimisen yhteydessä tämä tunnetaan termillä ydintemppu (engl. *kernel trick*). Yhteensä koodatut parametrit muodostavat 62 ulottuvuutta, joka täytetään vakioilla muodostaen 64 ulottuvuutta.

### 8.1.3 Näytteistäminen - muut

Siinä, missä edellä esiteltiin laajemmin valohierarkioihin perustuvaa hierarkkista näytteistämistä ja laskennan uudelleenkäyttöä yleisemmin suoran ja epäsuoran valaistuksen kuvaamiseen, käsittelevät tässä luvussa esiteltävät artikkelit spesifisempiä näytteistämisen menetelmiä erilaisten valonlähteiden näytteistämiseen ja valon ilmenemismuotojen kuvaamiseen, kuin myös näytteistämiseen korkeaa kehysnopeutta vaativissa sovelluksissa. Kirjallisuuskatsauksen aineiston näytteistämistä käsittelevistä artikkeleista neljä luokiteltiin muu kategoriaan, joista artikkelissa (Andersson ym. 2019) käsitellään näytteistämistä korkean kehysnopeuden sovelluksissa, artikkelissa (Peters ja Dachsbacher 2019) pallomaisen valonlähteen näytteistämistä, artikkelissa (Peters 2021) polygonisen pinta-alallisen valonlähteen näytteistämistä ja artikkelissa (Xu ym. 2022) kontaktivarjojen kuvaamista ympäristöstä saapuvalle valolle. Alla esitellään tarkemmin edellä mainitut artikkelit.

Artikkelissa (Andersson ym. 2019) esitetään menetelmä säteenseurannan hyödyntämiseen korkean kehysnopeuden (240 Hz) sovelluksissa, kuten esimerkiksi elektroniseen urheiluun suunnatuissa videopeleissä. Menetelmässä yhdistyy kehyksetön renderöinti (engl. *frameless rendering*)[35], lomitettu näytteistys (engl. *interleaved sampling*)[36] ja ajallinen reunanpehmennys. Menetelmässä kuvapuskuri jaetaan 2 x 2 pikseleiden muodostamiin tiiliin, joista ainoastaan yhtä pikseliä näytteistetään kehystä kohden, jolloin joka kehyksellä päivitetään ainoastaan neljäosa koko kuvapuskurista. Artikkelissa näin muodostetusta kuvapuskurista käytetään termiä alikehys (engl. *subframe*). Aliasoitumisen välttämiseksi näytteistämisjärjestystä 2 x 2 pikseleiden tiileissä vaihdellaan adaptiivisesti, kuitenkin säilyttäen saman näytteistämisjärjestyksen neljän alikehyksen ylitse. Esitettävän kehyspuskurin muodostamista varten pikselit jaetaan dynaamisiin ja staattisiin. Dynaamisille pikseleille, joita ei nykyisessä alikehyksessä näytteistetty, haetaan arvot edellisiltä alikehyksiltä. Staattisilla pikseleillä puoles-

35. Kehyksettömässä renderöinnissä (Bishop ym. 1994) pikseliä tai näytettä ei varsinaisesti sidota tiettyyn kehykseen, vaan kehyspuskurin päivitys voidaan tehdä esimerkiksi pikseleittäin sen sijaan, että kaikki kehyspuskurin pikselit päivitettäisiin kerralla ennen kehyksen esittämistä näytöllä.

36. Lomitetussa näytteistyksessä (Keller ja Heidrich 2002) samoin kuin luvussa 3.1 esitellyssä erotellussa lomitetussa näytteistyksessä tarkoituksena on näytteistää vain osaa pikseleistä, mutta siinä, missä erotellussa lomitetussa näytteistyksessä ajatuksena on tuottaa kehyspuskuri kokonaisuudessaan osissa, tuotetaan lomitetussa näytteistyksessä vain osa kokonaisesta kehyspuskurista.

taan kehyspuskuriin tallennetaan arvo keskiarvoistamalla ajallisesti enintään neljä pikselin sisällä poikkeutettua näytettä huolimatta siitä näytteistettiinkö kyseistä pikseliä nykyisessä alikehyksessä vai ei. Näin ollen dynaamisten pikseleiden efektiivinen näytemäärä on yksi, kun taas staattisten pikseleiden on neljä. Menetelmän arvioimiseksi suoritettiin käyttäjätutkimus.

Artikkelissa (Peters ja Dachsbacher 2019) esitellään reaaliaikainen harhaton menetelmä pallomaisten valonlähteiden näytteistämiseen diffuusi heijasteisten materiaalien pinnoilta. Menetelmässä tasajakautuneet näytteistettävät infitesimaaliset avaruuskulmat muodostetaan perustuen pallomaisen valonlähteen ja pinta-ala-alkion välisen projektioavaruuskulman geometriaan. Menetelmässä näytteistäminen perustuu projektioavaruuskulman jakamiseen segmentteihin (engl. *cut disk*), näytteistettävien pisteiden muodostamiseen segmenteillä tasajakauman mukaisesti ja näitä vastaavien infitesimaalisten avaruuskulmien muodostamiseen ja näytteistämiseen. Menetelmässä projektioavaruuskulman jakaminen segmentteihin käsitellään kolmessa erillisessä tapauksessa: pinta-ala-alkion ja pallomaisen valonlähteen välinen avaruuskulma kokonaan puoliavaruuden sisällä, jolloin projektioavaruuskulma muodostaa ellipsin pinta-ala-alkion puoliavaruuden projektiotasolle, kuten kuviossa 8, avaruuskulma enimmäkseen projektiotason yläpuolella ja avaruuskulma enimmäkseen projektiotason alapuolella.

Perustuen algebralliseen geometriaan artikkelissa (Peters 2021) esitellään puolestaan reaaliaikainen harhaton, sekä siitä johdettu harhallinen menetelmä tasomaisten (engl. *coplanar*) konveksisten polygonisten[37] valonlähteiden näytteistämiseen diffuusi ja spekulaari heijasteisten materiaalien pinnoilta. Siinä, missä edellä kuvatussa menetelmässä pallomaisen valonlähteen projektioavaruuskulma jaetaan segmentteihin, jaetaan artikkelin (Peters 2021) esittelemässä menetelmässä konveksin polygonisen valonlähteen projektioavaruuskulma pinta-ala-alkiosta katsoen sektoreihin, joiden kyljet piirtyvät polygonisen valonlähteen projisoitujen verteksien lävitse. Näytteiden muodostaminen sektoreilla avaruuskulman kattamissa osissa tapahtuu kahdessa tapauksessa riippuen siitä, osoittaako pinta-ala-alkion normaali kohti polygonista valonlähdettä. Tapauksista haastavampi on pinta-ala-alkion normaalin osoittaessa polygonisen valonlähteen ulkopuolelle, jolloin näytteiden muodostaminen tapah-

37. Ei-konveksit polygonit tulee jakaa konvekseihin ennen menetelmän soveltamista.

tuu iteratiivisesti. Siinä, missä diffuusi heijasteisilta pinnoilta näytteet muodostetaan tasajakauman mukaisesti projektioavaruuskulman sisällä, esitellään artikkelissa LTC (engl. *linearly transformed cosines*, *LTC*) (Heitz ym. 2016) jakaumien[38] käyttö näytteiden muodostamiseen spekulaari heijasteisilta pinnoilta. Tyypillisesti materiaalien kuvaamisessa käytetyt kaksisuuntaiset heijastusjakaumafunktiot voidaan jakaa diffuusi ja spekulaari komponentteihin, jolloin LTC jakauman mukaisesti painotettujen ja tasajakauman mukaisten näytteiden yhdistämiseen artikkelissa esitellään käytettäväksi monipainotettua näytteistystä.

Artikkelissa (Xu ym. 2022) esitellään reaaliaikainen menetelmä dynaamisten geometrioiden ja ympäristön valon aiheuttamien kontaktivarjojen kuvaamiseen. Menetelmässä seurattavat säteet rajoitetaan dynaamisia geometrioita rajaavien pallojen ja pinta-ala-alkion välille muodostettujen kartioiden alueelle[39], mistä artikkelista käytetään termiä suuntien karsiminen (engl. *direction culling*). Toisaalta säteen seuraaminen voidaan lopettaa viimeistään säteen suunnalla kauimmaisen rajaavan pallon kohdalla, mistä artikkelista käytetään termiä pituus karsinta (engl. *length culling*). Menetelmässä suunniltaan sisäkkäiset kartiot yhdistetään, jolloin yhdistetyn kartion korkeutena käytetään alkuperäisistä kartioista korkeamman kartion korkeutta. Menetelmä soveltuu tilanteisiin, joissa dynaamisten geometrioiden lukumäärä on alhainen, jolloin dynaamiset geometriat eivät kata pinta-ala-alkion puoliavaruutta kokonaisuudessaan. Dynaamisten geometrioiden lukumäärän kasvaessa säteiden karsimisen esittelemä hyöty vähenee ja samaan aikaan menetelmässä suoritettava kartioiden muodostaminen ja yhdistäminen vie enemmän aikaa, jolloin menetelmä soveltuu heikosti useamman dynaamisen geometrian aiheuttamien kontaktivarjojen kuvaamiseen[40].

38. LTC jakaumat käsittävät kosini jakaumasta lineaarimuunnoksien avulla tuotettuja variaatioita erilaisten fysikaalisten kaksisuuntaisten heijastusjakaumafunktioiden kuvaamiseksi (Heitz ym. 2016).

39. Toisin sanoen säteenseuranta kohdistetaan ainoastaan dynaamisia geometrioita rajaavien pallojen ja pinta-ala-alkion välisille avaruuskulmille.

40. Artikkelissa ei esitetä dynaamisten geometrioiden sisällyttämistä hierarkkiseen rakenteeseen niiden esikarsimiseksi esimerkiksi etäisyyden perusteella, mikä voisi nopeuttaa menetelmää virtuaalimaailman sisältäessä useampia dynaamisia geometrioita.

## 8.2 Häiriönpoisto

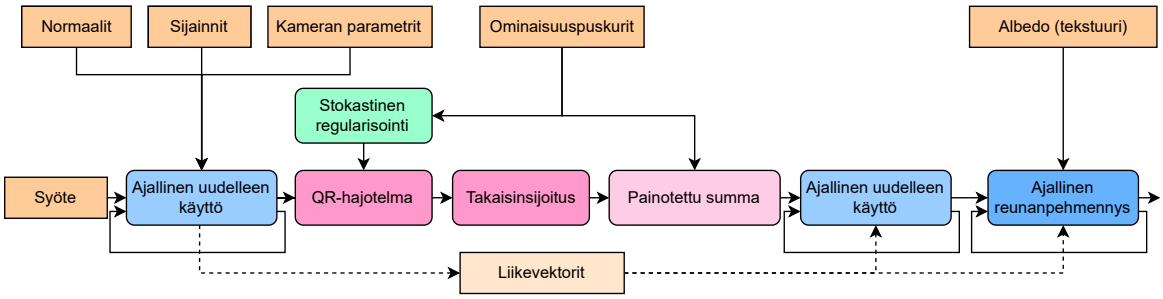
Kirjallisuuskatsauksen aineiston neljä artikkelia luokiteltiin häiriönpoisto kategoriaan, joista artikkelissa (Koskela ym. 2019) esitellään regressiopohjainen häiriönpoistomenetelmä, artikkelissa (Oberberger, Chajdas ja Westermann 2022) säteenseurantaa ja SVGF-menetelmää hyödyntävä menetelmä liike-epätarkkuuden (engl. *motion blur*) kuvaamiseen, artikkelissa (Xu, Ren ja Wu 2019) menetelmä filtteröinnissä käytettävän tunnetun jakauman ja kuvan välisen konvoluution approksimointiin ja artikkelissa (Thomas ym. 2022) neuroverkkoon perustuva häiriönpoisto- ja ylösskaalausmenetelmä. Lukuun ottamatta artikkelissa (Xu, Ren ja Wu 2019) esiteltyä filtterin approksimointiin soveltuvaa menetelmää, yhdistää menetelmiä kuvan muodostaminen yhdestä näytteestä pikseliä kohden, kuin myös G-puskuriin sisällytettyjen pikseleiden ominaisuuksia tallentavien häiriöttömien kuvapuskurien hyödyntäminen häiriönpoistossa, sekä tila-ajallinen näytteiden uudelleenkäyttö. Alla esitellään häiriönpoistoa käsittelevät aineiston artikkelit.

Artikkelissa (Koskela ym. 2019) esitellään ensimmäinen reaaliaikainen regressio-pohjainen häiriönpoistomenetelmä, BMFR (Blockwise Multi-Order Feature Regression). BMFR-menetelmässä lineaarisella regressioanalyysilla pyritään löytämään paras sovite ominaisuuspuskurien (engl. *feature buffer*) eli selittävien muuttujien ja kuvan RGB-värin värikomponenttien eli vastemuuttujien välille. Menetelmässä rekonstruktio luodaan sovitteen avulla. Ominaisuuspuskureilla tarkoitetaan mitä tahansa häiriöttömiä kuvapuskureita, kuten G-puskuriin sisällytettyjä kuvapuskureita. Menetelmässä sovite muodostetaan lohkoittain eli esimerkiksi 32 x 32 ja 64 x 64 pikselien tiilissä. Tiilien välisiä rajoja häivytetään poikkeuttamalla tiiliä peräkkäisillä kehyksillä, sekä menetelmän lopussa suoritettavalla ajallisella reunanpehmennyksellä. Menetelmän vaiheet ovat kuvattu kuviossa 43.

Syötteenä menetelmässä käytetään säteenseurannalla tuotettua kuvapuskuria sisältäen yhden näytteen pikseliä kohden[41]. Ennen lineaarista regressioanalyysia, menetelmässä muodostetaan ajallista uudelleenkäyttöä varten liikevektorit, poistetaan säteenseurannalla tuotetusta datasta tekstuurit, sekä suoritetaan stokastinen regularisointi (engl. *stochastic regularization*) ominaisuuspuskureille. Säteenseurannalla tuotetun datan lisäksi menetelmässä ole-

41. Artikkelissa esitellyssä toteutuksessa syötteenä käytetään NEE-menetelmän avulla kuvattua suoraa ja yhden kimmokkeen epäsuoraa valoa.

tetaan G-puskurin sisältävän mahdollisten selittävien muuttujien ohella ainakin pikseleiden normaalit ja sijainnit tallentavat kuvapuskurit, joita hyödynnetään muun muassa ajallisen uudelleenkäytön vaatimassa uudelleenprojisoinnissa. Ajallisessa uudelleenkäytössä menetelmässä sovelletaan kumulatiivista liikkuvaa keskiarvoa (engl. *cumulative moving average*, *CMA*) vähäisellä lukumäärällä ajallisesti uudelleenkäytettäviä näytteitä, kun taas eksponentiaalista liikkuvaa keskiarvoa käytetään silloin, kun ajallisesti uudelleenkäytettävien näytteiden lukumäärä on korkeampi. Stokastisella regularisoinnilla tarkoitetaan häiriön lisäämistä lineaarisen regressioanalyysin *syötteenä* käytettäviin ominaisuuspuskureihin mahdollisten selittävien muuttujien välisten lineaaristen riippuvuuksien poistamiseksi. Sen sijaan, että menetelmässä ominaisuuspuskureita käytettäisiin välttämättä sellaisenaan selittävinä muuttujina, muodostetaan ominaisuuspuskureista niin kutsuttu jatkettu ominaisuuspuskurien joukko (engl. *extended set*). Jatketulla ominaisuuspuskurien joukolla tarkoitetaan alkuperäisistä ominaisuuspuskureista muodostettua joukkoa, jossa ominaisuuspuskurit voivat esiintyä eriulotteisina[42].




Kuvio 43. Artikkelissa (Koskela ym. 2019) esitellyn BMFR-menetelmän häiriönpoisto liukuhihnan yleisnäkymä.

Merkitsemällä ominaisuuspuskureita $F = [F_1, ..., F_N]$, voidaan lukumäärältään $M$ ominaisuuspuskurista koostuva laajennettu ominaisuuspuskurien joukko[43] esittää muodossa $T = [F_{n_1}^{\gamma_1}, ..., F_{n_m}^{\gamma_m}, ..., F_{n_M}^{\gamma_M}]$, missä $n_m \in \{1, ..., N\}$, $\gamma_1 = 0$ ja $\gamma_m > 0$ ja $m = 2, ..., M$, jolloin ensimmäinen ominaisuuspuskuri on vakiopuskuri $F_{n_1}^0 = 1$. Kun merkitään tiilen $S$ sisältämää pik-

42. Tämän tarkoituksena on saattaa selittävien muuttujien ja vastemuuttujien väliset korrelaatiot lineaarisemmiksi lineaarisen regressioanalyysin onnistumiseksi.

43. Artikkelissa esitellyssä toteutuksessa käytetään laajennettua ominaisuuspuskuria $T = [1, n_x, n_y, n_z, w_x, w_y, w_z, w_x^2, w_y^2, w_z^2]$, missä $n$ ja $w$ ovat pikselien normaalit ja sijainnit tallentavat kuvapuskurit vastaavassa järjestyksessä.

seliä **p**:llä, voidaan lineaarisen regressioanalyysin ratkaisu esittää pienimmän neliösumman lausekkeena

$$\hat{\alpha}^{(c)} = \underset{\alpha^{(c)} \in \mathbb{R}^M}{\operatorname{argmin}} \sum_{\mathbf{p} \in S} \left( Z^{(c)}(\mathbf{p}) - \sum_{m=1}^{M} \alpha_m^{(c)} F_{n_m}^{\gamma_m}(\mathbf{p}) \right)^2, \tag{8.4}$$

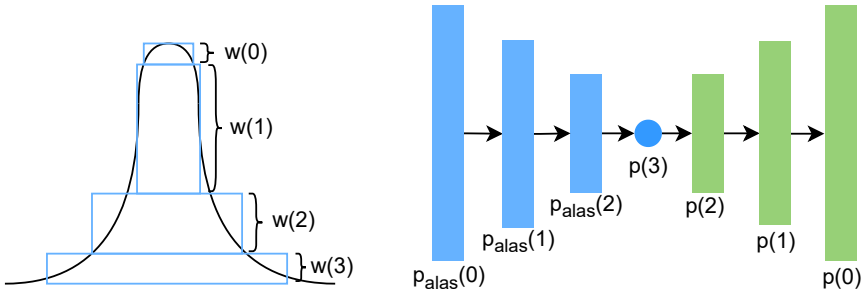
missä $Z^{(c)}$ on syötteenä käytettävän säteenseurannalla tuotetun kuvapuskurin RGB-värin värikomponentti $c$. Menetelmässä pienimmän neliösumman lauseke ratkaistaan Householderin QR-hajotelmaa käyttäen. Rekonstruktio $Y$ värikomponentille $c$ voidaan sovitteen avulla esittää tällöin painotettuna summana

$$Y^{(c)}(\mathbf{p}) = \sum_{m=1}^{M} \hat{\alpha}_m^{(c)} F_{n_m}^{\gamma_m}(\mathbf{p}). \tag{8.5}$$

Rekonstruktion jälkeen uudelleen käyttäen menetelmässä aikaisemmin luotuja liikevektoreita, suoritetaan menetelmässä ajallinen näytteiden uudelleenkäyttö ja reunanpehmennys tiilimäisyyden häivyttämiseksi, sekä kehysten välisen varianssin vähentämiseksi. Ilman ajallisesti uudelleenkäytettäviä näytteitä, ovat menetelmässä tiilet havaittavissa.

Siinä, missä luvussa 4.4 bilateraalisten filttereiden approksimointiin esiteltiin à-trous-aallokemuunnos, esitellään artikkelissa (Xu, Ren ja Wu 2019) menetelmä filtteröinnissä käytettävän tunnetun jakauman ja kuvan välisen konvoluution approksimointiin. Artikkelin esittelemässä menetelmässä approksimaatio filtterin ytimen ja kuvan välisestä konvoluutiosta muodostetaan kuvan alasnäytteistettyjen kuvien eli tasojen ja alasnäytteistetyistä tasoista tuotettujen ylösnäytteistettyjen tasojen painotettuna summana. Alasnäytteistäminen ja sitä seuraava ylösnäytteistäminen esitetään kuviossa 44 oikealla. Menetelmässä alasnäytteistäminen perustuu nopeisiin laatikko filttereihin (engl. *box filter*)[44], joilla pyritään tuottamaan approksimaatio tavoite filtterin jakaumasta, kuten esitellään kuviossa 44 vasemmalla. Mitä useampaa tasoa käytetään, sitä tarkempi approksimaatio filtterin jakaumasta kyetään muodostamaan. Ylösnäytteistys puolestaan tapahtuu lineaarisesti interpoloimalla edellä ylösnäytteistettyä karkeampaa eli matalamman resoluution kuvaa ja saman tason eli saman resoluution

44. Laatikko filtterissä jokaisen ytimen elementin arvo on sama, jolloin pikselille muodostettu arvo on pikselin ja sen vierekkäisten pikselien painottamaton keskiarvo.




Kuvio 44. Menetelmän (Xu, Ren ja Wu 2019) jakauman approksimointi käyttäen laatikko filttereitä (vasemmalla), sekä menetelmän liukuhihna koostuen monivaiheisesta alasnäytteistämisestä ja sitä seuraavasta monivaiheisesta ylösnäytteistämistä (oikealla).

alasnäytteistettyä kuvaa. Merkitsemällä pikselin arvoa $p(L)$ tasolla $L$ ja karkeammalla tasolla sen vastinetta $p(L+1)$, sekä merkitsemällä $p_{alas}(L)$:llä puolestaan pikselin arvoa samalla tasolla alasnäytteistystä kuvasta, voidaan ylösnäytteistys esittää muodossa

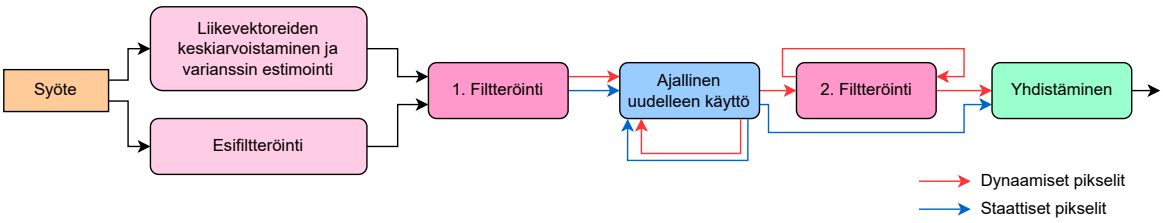
$$p(L) = (1-\alpha(L))p(L+1) + \alpha(L)p_{alas}(L), \tag{8.6}$$

missä merkitsemällä tasojen lukumäärää $m$:llä voidaan tasojen painotus $\alpha$ esittää muodossa

$$\alpha(L) = \frac{w(L)}{\int_L^m w(l)dl}, \tag{8.7}$$

missä painokertoimet $w$ riippuvat approksimoitavasta jakaumasta, mistä syystä jakauma tulee tuntea sekä painokertoimet muodostaa etukäteen. Artikkelissa menetelmä esitellään käytettäväksi Gaussin sumentamiseen, irradianssiluotaimien generointiin, kuin myös säteenseurannalla tuotettujen spekulaari heijastusten kuvaamiseen yhdestä näytteestä pikseliä kohden.

Artikkelissa (Oberberger, Chajdas ja Westermann 2022) esitellään SVGF-menetelmän hyödyntäminen liike-epätarkkuuden kuvaamiseksi yhdestä näytteestä pikseliä kohden. Menetelmä on kuvattu kuviossa 45. Menetelmässä syötteenä käytetään säteenseurannalla seurattuja primäärisäteitä ja primäärisäteiden lähtöpisteen ja suunnan määrääviä satunnaisesti luotuja aikaleimoja. Aikaleima kertoo pikselissä seurattavan primäärisäteen muodostamisen ajan-

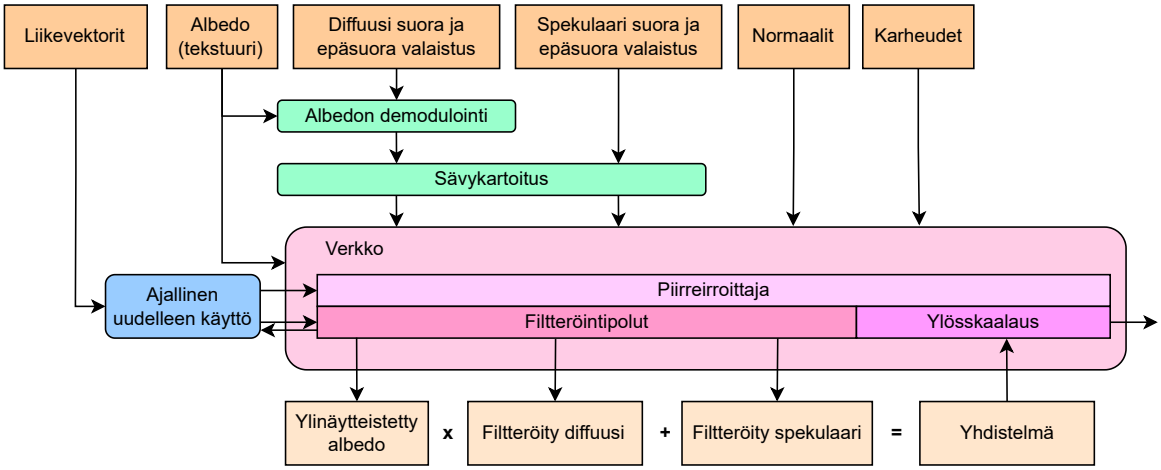



Kuvio 45. Artikkelissa (Oberberger, Chajdas ja Westermann 2022) esitellyn liike-epätarkkuuden kuvaamiseen tarkoitetun menetelmän liukuhihnan yleisnäkymä.

hetken kehyksen aloituksen ja päättymisen väliltä[45]. Primäärisäteiden ja aikaleimojen lisäksi menetelmässä syötteenä käytetään G-puskuriin sisällytettyjä pikseleiden liikevektorit ja syvyysarvot tallentavia kuvapuskureita. Siinä, missä SVGF-menetelmässä pikseleiden luminanssi arvoja käytetään varianssin estimointiin filtteröinnin ohjaamiseksi, käytetään artikkelin menetelmässä pikseleiden liikevektoreita varianssin estimointiin filtteröinnin ohjaamiseksi. Varianssin estimoinnin yhteydessä liikevektorit filtteröidään ja keskiarvoistetaan. Keskiarvoistettuja liikevektoreita käytetään menetelmän myöhemmissä vaiheissa filtteröinnin yhteydessä. Varianssin estimointia ennen ei suoriteta näytteiden ajallista uudelleenkäyttöä kuten SVGF-menetelmässä.

Varianssin vähentämiseksi niissä pikseleissä, joiden liikevektorien suuruudet ovat yli ennalta määritetyn raja-arvon, suoritetaan à-trous-aallokemuunnokseen perustuva esifiltteröinti, jossa käytettävän ytimen nollaelementtien lukumäärä riippuu liikevektorin suuruudesta. Siinä, missä SVGF-menetelmässä à-trous-aallokemuunnoksessa käytettävä painofunktio perustuu pikseleiden luminanssi-, syvyys- ja normaaliarvoihin reunojen säilyttämiseksi, perustuu artikkelin menetelmässä dynaamisten pikseleiden painofunktio pikselin $p$ ja ytimeen kuuluvan

45. Näin ollen kehyksen kesto tulee tuntea, mikä vaatii kiinteän kehysnopeuden. Eri aikoina seurattavat säteet antavat todellista tietoa virtuaalinäkymästä eri ajanhetkillä, jolloin osassa pikseleillä nähdään esimerkiksi liikkuvan geometrian taakse, mitä ei jälkikäsittelynä tuotetussa liike-epätarkkuuden kuvaamisessa ole saatavilla. Artikkelissa ei täsmällisemmin kuvata säteenseurantaa, mutta eri ajanhetkien kuvaamiseksi tulee kameran muunnosmatriisia ja liikkuvien ilmentymien muunnosmatriiseja interpoloida virtuaalimaailman esittämiseksi kyseisellä ajanhetkellä, mikä voi tehdä säteenseurannasta laskennallisesti vaativaa. Vulkan API:ssa NVIDIA Ampere ja Ada Lovelace arkkitehtuurien näytönohjaimilla säteenseurannalla suoritettavan liike-epätarkkuuden kuvaamissa voidaan hyödyntää VK_NV_ray_tracing_motion_blur laajennosta (“Vulkan® 1.3.224 - A Specification” 2022).



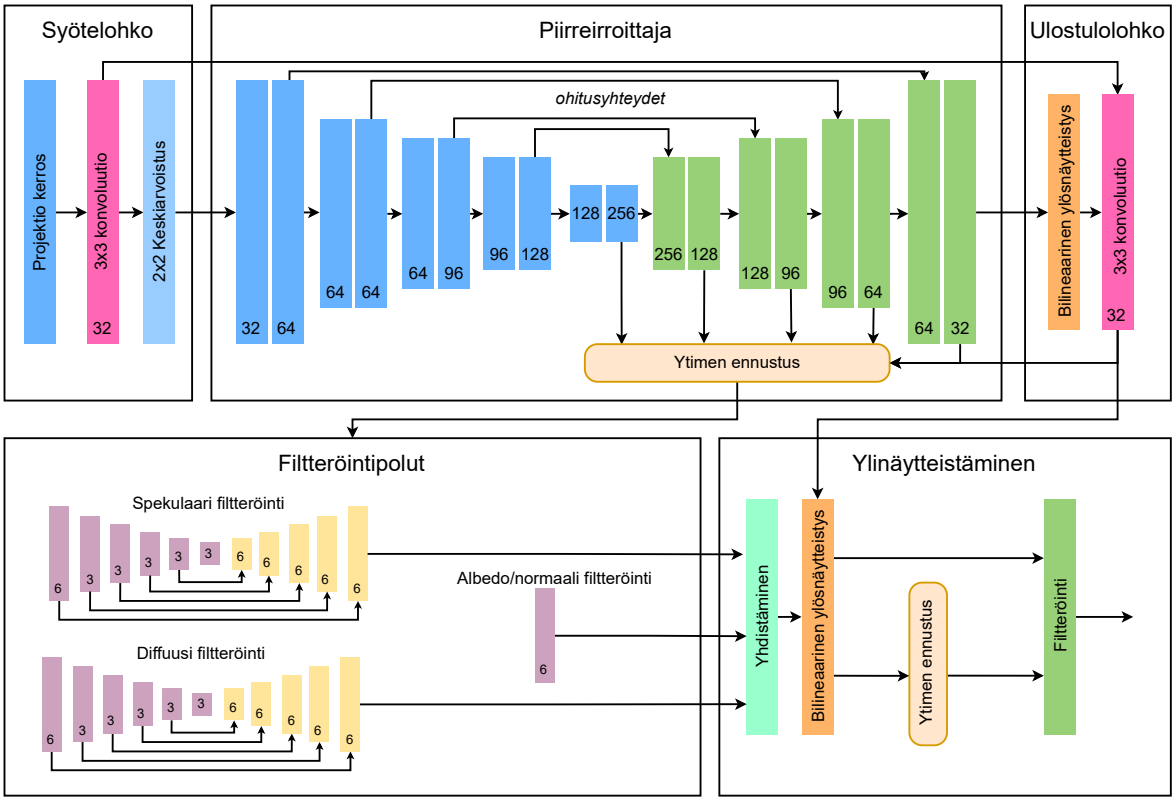

Kuvio 46. Artikkelissa (Thomas ym. 2022) esitellyn häiriönpoisto- ja ylösskaalausmenetelmän liukuhihnan yleisnäkymä.

pikselin $q$ välillä pikselin $q$ liikevektorin ja aikaleiman määrittelemän kehyksen aikana tapahtuvaa siirtymää kuvaavan segmentin ja pikselin $p$ väliseen vähimmäisetäisyyteen[46]. Esifiltteröinnin jälkeen menetelmässä suoritetaan à-trous-aallokemuunnos kahdessa vaiheessa, joista ensimmäisessä vaiheessa filtteröidään sekä dynaamiset että staattiset pikselit erikseen ja toisessa vaiheessa ainoastaan dynaamiset pikselit. Staattisille pikseleille ei käytetä edellä kuvattua painofunktiota. Filtteröinti muuttaa sekä pikseleiden väriarvoja että liikevektoreita. Ennen toista ja ainoastaan dynaamisille pikseleille suoritettavaa filtteröintivaihetta, suoritetaan sekä dynaamisille että staattisille pikseleille ajallinen näytteiden uudelleenkäyttö. Toisessa vaiheessa tapahtuvassa fitteröinnissä ydintä skaalataan ja kierretään keskiarvoistetun liikevektorin mukaisesti. Menetelmän lopussa dynaamiset pikselit ja staattiset pikselit yhdistetään.

Artikkelissa (Thomas ym. 2022) esitellään neuroverkko häiriönpoistoon ja kehyspuskurin ylösskaalaamiseen ylinäytteistämisen (engl. *supersampling*) avulla. Häiriönpoisto ja ylösskaalaus tapahtuvat samaa neuroverkkoa ja piirreirroittajaa (engl. *feature extractor*) käyttäen[47]. Korkealla tasolla menetelmä on kuvattu kuviossa 46. Menetelmässä syötteenä käyte-

46. Tarkoituksena filtteröinnissä on käyttää niitä pikseleitä $q$, jotka ovat voineet jollain ajanhetkellä kehyksen sisällä sijaita likimain pikselin $p$ sijainnissa.

47. Tämä on toisin kuin esimerkiksi NVIDIA:n DLSS ja Intelin X$^{e}$SS ylösskaalausmenetelmissä, joissa oletetaan syötteen olevan häiriötöntä.




Kuvio 47. Artikkelissa (Thomas ym. 2022) esitellyn häiriönpoisto ja ylösskaalausmenetelmän neuroverkon vaiheet ja arkkitehtuuri.

tään eri kuvapuskureihin tallennettuja diffuusi ja spekulaari heijasteisille pinnoille kuvattuja suoraa ja epäsuoraa valoa eli diffuusi ja spekulaari komponentteja, sekä G-puskuriin sisällytettyjä pikseleiden normaalit, karheudet ja liikevektorit tallentavia kuvapuskureita. Häiriönpoistoa varten tekstuuri erotellaan valaistuksesta (engl. *demodulate*), jonka lisäksi HDR (High-Dynamic Range) väriarvot katkaistaan (engl. *clamp*) 16-bittiin, jonka jälkeen väriarvoille suoritetaan logaritminen[48] sävykartoitus (engl. *tonemapping*). Menetelmässä diffuusi ja spekulaari komponentit filtteröidään erikseen. Diffuusi ja spekulaaari komponenttien filtteröinnissä käytettävien spatiaalisten ja ajallisten filttereiden ytimien painokertoimet perustuvat piirreirroittajan ennustukseen, samoin kuin ylinäytteistämisen jälkeen suoritettavassa filtteröinnissä käytettävän ytimen.

Menetelmän neuroverkko on esitetty kuviossa 47. Verkon piirreirroittaja perustuu artikke-

48. Menetelmässä sävykartoitukseen käytetään artikkelissa (Hasselgren ym. 2020) esiteltyä funktiota väriarvolle $x$: $(log(x+1)^{\frac{1}{2.2}})$.

lissa (Ronneberger, Fischer ja Brox 2015) esitettyyn U-Net arkkitehtuuriin[49] koostuen enkoodereista (engl. *encoder*) ja dekoodereista (engl. *decoder*). Kuviossa sinisellä kuvatut enkooderi lohkot koostuvat kahdesta 3x3 konvoluutiokerroksesta, ReLU aktivaatiosta ja max pool operaatiosta. Kuviossa vihreällä kuvatut dekooderi lohkot puolestaan koostuvat edellisen lohkon bilateraalisesta ylösnäytteistyksestä, sekä kahdesta 3x3 konvoluutiokerroksesta ja ReLU aktivaatiosta. Enkooderien tarkoitus on irroittaa kuvista oleellisia piirteitä, kun taas dekooderien tarkoitus on tuottaa semanttinen segmentointi (engl. *semantic segmentation*)[50] pikseleittäin. Kuviossa esitettyjen ohitusyhteyksien (engl. *skip connection*) tarkoitus on säilyttää yksityiskohdat, sekä nopeuttaa verkon kouluttamista. Tappiofunktiona (engl. *loss function*) verkossa käytetään diffuusi ja spekulaari komponenttien, G-puskuriin tallennettujen häiriöttömien kuvapuskureiden, sekä ylösskaalauksen tappiofunktioiden painotettua summaa

$$L := w_{diffuusi} \cdot L_{diffuusi} + w_{spekulaari} \cdot L_{spekulaari} + w_{apu} \cdot L_{apu} + w_{yskaalaus} \cdot L_{yskaalaus}, \quad (8.8)$$

missä kukin $w_i$ on vastaavan tappiofunktion painokerroin [51], kun taas yksittäiset tappiofunktiot ovat kukin muotoa

$$L_i = w_{spatiaalinen_i} \cdot L_{spatiaalinen_i} + w_{ajallinen_i} \cdot L_{ajallinen_i}, \quad (8.9)$$

missä $w_{spatiaalinen_i}$ ja $w_{ajallinen_i}$ ovat spatiaaliselle $L_{spatiaalinen_i}$ ja ajalliselle $L_{ajallinen_i}$ tappiofunktiolle asetetut painokertoimet[52]. Tappiofunktioista ajallinen tappiofunktio voidaan esittää $L_1$-normina[53] muodossa

49. Alkuperäisessä artikkelissa U-Net arkkitehtuuria esitellään käytettäväksi kuvan segmentointiin biolääketieteessä.

50. Semanttisella segmentoinnilla tarkoitetaan kuvan jakamista sisällöltään merkityksellisiin segmentteihin eli luokkiin muodostamalla segmentointi peite (engl. *segmentation mask*) pikseleittäin. Segmentointi peite kertoo jokaiselle pikselille luokan, johon se kuuluu.

51. Artikkelissa esitellyssä toteutuksessa jokaiselle tappiofunktiolle asetettiin painokertoimeksi yksi.

52. Artikkelissa esitellyssä toteutuksessa käytettiin $w_{spatiaalinen_i} = 0{,}2$ ja $w_{ajallinen_i} = 1 - w_{spatiaalinen_i}$.

53. $L_1$-normi tarkoittaa itseisarvojen summaa.

$$L_{ajallinen_i} = L_1(o_i - f_w(o_{i-1}), r_i - f_w(r_{i-1})), \tag{8.10}$$

missä $o_i$ ja $r_i$ ovat tuotettu kuva ja referenssikuva nykyiseltä kehykseltä, kun taas $f_w(o_{i-1})$ ja $f_w(r_{i-1})$ ovat edellisen kehyksen tuotettu kuva ja referenssikuva. Spatiaalinen tappiofunktio puolestaan määräytyy erillisen U-Net arkkitehtuuriin perustuvan havainnollisen tappioverkon (engl. *perceptual loss network*) [54] perusteella. Havainnollisen tappioverkon tappiofunktio kolmen ensimmäisen verkon kerroksen $j$ piirrekarttojen avulla voidaan esittää muodossa

$$L_{spatiaalinen_i} = \sum_{j=1}^{3} w_j \cdot L_1(f_j(o), f_j(r)), \tag{8.11}$$

missä $w_j$ on painokerroin tasolle $j$, $f_j(o)$ on tuotettu piirrekartta ja $f_j(r)$ referenssipiirrekartta. Havainnollinen tappioverkko koulutettiin segmentointi peitteen muodostamiseen käyttäen Applen Hypersim tietoaineistoa. Menetelmän varsinainen neuroverkko puolestaan koulutettiin käyttäen Unreal Engine 4.25 (UE4) pelimoottorilla kerättyä häiriöllistä dataa, UE4 pelimoottorin häiriönpoistomenetelmää käyttäen tuotettua dataa, sekä referenssidataa[55].

## 8.3 Muut aiheet

Näytteistäminen ja häiriönpoisto kategorioiden ulkopuolelle aineiston artikkeleista luokiteltiin kaksi artikkelia, joissa molemmissa käsitellään laitteiston, ajurin ja renderöinti-ohjelmointirajapinnan muutoksia. Artikkelissa (Lee ja Liktor 2020) ehdotetaan iteratiivista viivästettyä kiihdytysrakenteen rakentamista (engl. *multi-pass lazy build*, *MPLB*). MPLB-menetelmän tarkoituksena on vähentää kiihdytysrakenteiden, erityisesti animoitujen pohjatason kiihdytysrakenteiden, rakentamiseen ja päivittämiseen kuluvaa aikaa. Menetelmä perustuu artikkelissa (Lee, Liktor ja Vaidyanathan 2019) ehdotettuihin läpikulkuvarjostimeen (engl. *traversal*) ja ohjelmoitaviin ilmentymiin (engl. *programmable instance*, *PI*). Ohjelmoitavalla

54. Havainnollisella tarkoitetaan kuvan huomioimista kokonaisuutena, jolloin yksittäisten pikselien arvojen samankaltaisuuden sijasta havainnollisessa tappiofunktiossa arvioidaan kuvan sisältämien piirteiden samankaltaisuutta.

55. Referenssidata tuotettiin käyttäen 64-128 näytettä pikseliä kohden ja ylösnäytteistämällä 720p formaatista 1440p formaattiin.

ilmentymällä tarkoitetaan kiihdytysrakenteeseen sijoitettavaa tyhjää rajaavaa tilavuutta, johon säteen osuessa kutsutaan ilmentymälle määrättyä läpikulkuvarjostinta. Läpikulkuvarjostimesta voidaan määritellä se pohjatason kiihdytysrakenne, jossa säteenseurantaa jatketaan. Artikkelissa (Lee, Liktor ja Vaidyanathan 2019) läpikulkuvarjostinta esitetään käytettäväksi sopivan tarkkuustason ilmentymän valitsemiseksi.

MPLB-menetelmässä läpikulkuvarjostinta käytetään puolestaan pohjatason kiihdytysrakenteiden rakennuttamiseksi. Menetelmässä ylätason kiihdytysrakenne kehyksen alussa koostetaan edellisellä kehyksellä näkyvistä pohjatason kiihdytysrakenteista, sekä näihin kuulumattomista pohjatason kiihdytysrakenteista, joiden näkyvyys voidaan rasteroinnilla päätellä. Puolestaan ei-näkyvien pohjatason kiihdytysrakenteiden tilalle kiihdytysrakenteeseen asetetaan ohjelmoitavat ilmentymät. Sekundäärisäteen osuessa ohjelmoitavaan ilmentymään, merkitään ilmentymän pohjarakenne rakennettavaksi läpikulkuvarjostimessa, mikäli kiihdytysrakennetta ei ole vielä rakennettu. Menetelmässä iteratiivisesti seurataan säteitä jokaista pikseliä kohden, kunnes yksikään säde ei leikkaa ohjelmoitavaa ilmentymää, jolloin kaikki ne pohjatason kiihdytysrakenteet, joita tullaan tarvitsemaan ovat rakennettu. Tämän jälkeen ylätason kiihdytysrakenne muodostetaan jo rakennetuista kiihdytysrakenteista. Menetelmä hyödyntää erityisesti virtuaalimaailmassa, joka koostuu lukuisista animoiduista pohjatason kiihdytysrakenteista, jolloin MPLB-menetelmä vähentää pohjatason kiihdytysrakenteiden rakentamiseen kuluvaa aikaa verrattuna tilanteeseen, jossa kaikki pohjatason kiihdytysrakenteen rakennettaisiin joka kehyksellä, kunhan kaikkia pohjatason kiihdytysrakenteita ei tulla tarvitsemaan valon ilmenemismuotojen kuvaamiseen.

Artikkelissa Liu ym. 2021 puolestaan ehdotetaan säteen leikkauksen ennustavaa yksikköä nopeuttamaan erityisesti ympäristön okkluusion laskentaa, sekä näytetään yksikön toimivuus käyttäen GPGPU-Sim 4.0.1 simulaattoria, jota kirjoittavat laajensivat RT-ytimen toteutuksella. Yksikön tarkoitus on ylläpitää potentiaalisten sisä- ja lehtisolmujen taulukkoa, josta säteelle muodostetun tiivisteen avulla saadaan ne solmut, joiden kanssa säde todennäköisemmin tulee leikkaamaan. Mikäli tiivisteellä on olemassa potentiaalisia solmuja, aloitetaan leikkaavuustarkastelu potentiaalisista solmuista sen sijaan, että leikkaavuustarkastelu aloitettaisiin kiihdytysrakenteen juurisolmusta. Mikäli säde leikkaa potentiaalisen lehtisol-

mun, vältytään kiihdytysrakenteen matkaamiselta[56]. Jos säde ei leikkaa yhtään potentiaalista solmua tai tiivisteelle ei ole saatavilla potentiaalisia solmuja, aloitetaan säteen seuraaminen kiihdytysrakenteen juurisolmusta ja päivitetään säteen leikkaama solmu potentiaalisten solmujen taulukkoon.

56. Tämä silloin, kun ei olla kiinnostuneita lähimmän leikkaavan primitiivin löytämisestä, vaan leikkaavuudesta yleensä, kuten ympäristön okkluusion kuvaamisessa.

# 9 Yhteenveto

Tutkielmassa kartoitettiin laitteistokiihdytteisen säteenseurannan käyttöä ja haasteita reaaliaikaisessa renderöinnissä vastaten kahteen tutkimuskysymykseen. Tutkielman ensimmäisenä tutkimuskysymyksenä oli tutkia laitteistokiihdytteisen säteenseurannan suorituskykyä ja siihen vaikuttavia tekijöitä. Kysymykseen vastattiin tutkielman empiirisessä osiossa, jossa tutkittiin laitteistokiihdytteisen säteenseurannan suorituskyvyn ylärajaa Vulkan API:lla toteutetussa koeympäristössä, sekä kiihdytysrakenteen ominaisuuksien ja määrittelyjen vaikutusta ylärajaan. Vaikka empiirisen osion kokeissa koelaitteistolla päästiin näytönohjainvalmistajan ilmoittamaan 5 gigasäteeseen sekunnissa hyvin pelkistetyssä virtuaalimaailmassa, putoaa suorituskyky merkittävästi niin kolmioiden lukumäärän, kuin myös ilmentymien lukumäärän myötä, sekä kolmioiden ja ilmentymien sijoittautumisesta riippuen. Verrattuna säännöllisiin ja konvekseihin kolmioverkkoihin, epäsäännöllisillä ja epäkonvekseilla kolmioverkolla havaittiin huomattavasti alhaisempi säteenseurannan suorituskyky, samoin kuin ilmentymien sijoittuessa epäsäännöllisesti ylätason kiihdytysrakenteeseen. Kompleksisemmissa virtuaalimaailmoissa, kuten usein videopelien virtuaalimaailmoissa, laitteistokiihdytteisen säteenseurannan suorituskyvyn uskotaan jäävän empiirisistä kokeista saatua ylärajaa huomattavasti alhaisemmaksi, jonka lisäksi ei useinkaan ole mahdollista koko kehykseen käytettävää aikaa käyttää säteiden seuraamiseen, mikä myös vähentää seurattavien säteiden lukumäärää kehyksellä.

Suoritettujen kokeiden tulokset puoltavat matalamman tarkkuustason geometrioiden käyttöä epätarkkuutta sietävien valon ilmenemismuotojen, kuten diffuusi heijastusten kuvaamisessa, jolloin kiihdytysrakenteen sisältämien kolmioiden kokonaismäärä kuin myös usein epäsäännöllisyys ja epäkonveksisuus vähenee. Ylätason kiihdytysrakenteen sisältämien ilmentymien sijoittelun säännöllistäminen ei ehkä useinkaan ole mielekästä, jolloin säteenseurannan suorituskyvyn kasvattamiseksi monesti tarpeelliseksi tulee tarpeettomien ilmentymien karsiminen pois kiihdytysrakenteesta, mikä on huomattava haaste säteenseurannassa verrattuna esimerkiksi rasterointiin. Mielenkiintoinen lisäkoe olisi selvittää, miten lähekkäisten kolmioverkkojen sisällyttäminen samaan pohjatason kiihdytysrakenteeseen ilmentymien lukumäärän vähentämiseksi vaikuttaisi säteenseurannan suorituskykyyn. Kiihdytysrakenteen sisällön

lisäksi säteenseurannan suorituskykyyn voidaan vaikuttaa pohjatason kiihdytysrakenteiden muodostamisen yhteydessä annettavilla lippubiteillä, joiden soveltuvuus kuin myös Vulkan-spesifikaation ulkopuolelle ajurin vastuulle jäävien lippubittien toimivuus on hyvä varmistaa sovelluskohteessa. Myös ylätason kiihdytysrakenteen päivittäminen uudelleenrakentamisen sijasta voi vaikuttaa säteenseurannan suorituskykyyn alentavasti kiihdytysrakenteen sisältäessä dynaamisia ilmentymiä. Suorituskyvyn alentuminen kasvaa niin dynaamisten ilmentymien lukumäärän kuin myös dynaamisten ilmentymien poikkeamien myötä, mistä syystä kiihdytysrakenteen uudelleenrakentaminen voi usein osoittautua toimivammaksi ratkaisuksi kuin kiihdytysrakenteen päivittäminen.

Siinä, missä kokeet suoritettiin Turing-arkkitehtuurin RTX 2060 näytönohjaimella, joka sisältää 30 ensimmäisen sukupolven RT-ydintä, sisältävät puolestaan Turing-arkkitehtuurin RTX 2080 Ti -näytönohjain 68 RT-ydintä ja NVIDIA:n Ampere-arkkitehtuurin RTX 3080 Ti -näytönohjain 80 toisen sukupolven RT-ydintä, kun taas NVIDIA:n viimeisimmän kuluttajaluokan Ada Lovelace -arkkitehtuurin RTX 4090 -näytönohjain sisältää 128 kolmannen sukupolven RT-ydintä (Corporation 2023). RT-ytimien lukumäärän kasvun lisäksi myös ytimien suorituskyky on noussut kolmannen sukupolven RT-ytimien säde-kolmioleikkaavuustarkastelua kiihdyttävän yksikön suorituskyvyn ollessa nelinkertainen verrattuna ensimmäisen sukupolven RT-ytimiin ja kaksinkertainen verrattuna toisen sukupolven RT-ytimiin (Corporation 2023). Näin ollen kokeissa saatu yläraja on oletettavasti huomattavasti matalampi kuin mitä olisi kehittyneemmillä näytönohjaimilla. Ottaen huomioon, että laitteistokiihdytteinen säteenseuranta on vielä verrattain uusi teknologia, jää nähtäväksi kuinka paljon RT-ytimiä voidaan mahduttaa näytönohjaimeen ja kuinka paljon RT-ytimen sisältämien säde-kolmioleikkaavuustarkastelua ja kiihdytysrakenteen läpikäyntiä kiihdyttävien yksiköiden suorituskykyä voidaan vielä kasvattaa. Toisaalta yhtä lailla säteenseurannalle ominaisten haasteiden tullessa tutuiksi, voi säteenseurannan suorituskyky kehittyä myös laitteistotasolla tehtyjen muiden ratkaisujen avulla. Yhtenä esimerkkinä tästä voidaan pitää NVIDIA:n Ada Lovelace -arkkitehtuurin näytönohjaimissa toteutettua varjostinkutsujen uudelleenjärjestelemistä[1].

1. Varjostinkutsujen uudelleenjärjestelemisen ohella NVIDIA:n Ada Lovelace -arkkitehtuurin tuomia uudistuksia ovat osittain läpinäkyvien geometrioiden säteenseurantaa kiihdyttävän yksikön (engl. *Opacity Micromap Engine*) ja kompleksisten geometrioiden säteenseurantaa kiihdyttävän yksikön (engl. *Displaced Micro-*

Tutkielman toisena tutkimuskysymyksenä oli esitellä tapoja ja menetelmiä laitteistokiihdytteisen säteenseurannan hyödyntämiseen reaaliaikaisessa renderöinnissä, sekä siihen liittyviä haasteita. Tutkimuskysymykseen vastattiin tarkastelemalla säteenseurannan hyödyntämistä nykyisissä reaaliaikaisissa sovelluksissa, kuin myös kirjallisuuskatsauksen avulla. Tutkielmassa säteenseurannan hyödyntäminen reaaliaikaisissa sovelluksissa rajattiin videopeleihin aineiston koostuen pääsääntöisesti GDC konferenssitallenteista. Konferenssitallenteiden osittain epätarkka videopeleissä käytettyjen menetelmien kuvaaminen rajoitti menetelmien tarkempaa esittelyä, mistä syystä osiossa päädyttiin esittelemään aineiston videopeleissä ilmeneviä yleisiä piirteitä. Aineiston videopeleissä renderöinti toteutetaan hybriditekniikkana säteenseurantekniikan täydentäessä rasterointitekniikkaa tiettyjen valon ilmenemismuotojen kuvaamisessa. Vuorottelu rasterointitekniikan ja säteenseurantatekniikan välillä voidaan toteuttaa joko siten, että molempia käytetään saman valon ilmenemismuodon kuvaamiseen tai tekniikkoja käytetään kuvaamaan eri valon ilmenemismuotoja. Aineiston videopeleissä käytettyjä tyypillisiä heuristiikkoja säteenseurantatekniikan hyödyntämiselle yhdessä rasterointitekniikan kanssa saman valon ilmenemismuodon kuvaamiseen ovat säteenseurannan hyödyntäminen niissä pikseleissä, joissa rasterointipohjaisella menetelmällä ei kyetä luotettavasti kyseistä valon ilmenemismuotoa kuvaamaan, kuten saattaa käydä näkymäavaruuden menetelmissä, sekä säteenseurannan hyödyntäminen siellä, missä vaikutus on parhaiten havaittavissa, kuten esimerkiksi lähellä virtuaalikameraa. Aineiston videopeleissä yleisimmät säteenseurannan käyttökohteet ovat näkymäavaruuden heijastusten ja varjokarttojen avulla toteutettujen varjojen täydentäminen.

Puolestaan aineiston videopeleissä havaitut tyypillisimmät haasteet ovat useamman valonlähteen näytteistäminen, mistä syystä osassa aineiston videopeleissä näytteistäminen suoritetaan ainoastaan yhdestä valonlähteestä, kuten auringosta tai kuusta, säteenseurannan eriytyminen, sekä tarpeettomien ilmentymien karsiminen kiihdytysrakenteesta. Siinä, missä aika tulee näyttämään laitteistotason kehityksen, pätee se myös reaaliaikaisen renderöinnin sovelluksiin. Rasterointitekniikalla on pitkä historia reaaliaikaisessa renderöinnissä, mistä syystä pelimoottoreiden voidaan ajatella olevan pitkälle optimoituja rasterointitekniikan suorittamiseen, mikä ei välttämättä päde vielä säteenseurantaan, jolloin säteenseurannan tiukempi integrointi osaksi pelimoottoria ja grafiikka API:en uusimpien laajennosten ja ominaisuuksien

*Mesh Engine*) lisääminen RT-ytimiin.

käyttöönotto saattaa myös kasvattaa säteenseurannan suorituskykyä tulevissa reaaliaikaisissa sovelluksissa. Joka tapauksessa säteenseurannan kohdeyleisön kasvaessa viimeisemmän sukupolven Xbox ja Playstation -pelikonsolien tukiessa laitteistokiihdytteistä säteenseurantaa, voidaan peliyhtiöiltä odottaa aiempaa suurempaa panostusta säteenseurannan kehittämiseen.

Kirjallisuuskatsauksessa pyrkimyksenä oli kattavasti löytää menetelmiä laitteistokiihdytteisen säteenseurannan hyödyntämiseen reaaliaikaisessa renderöinnissä, mistä syystä menetelmien rajaamisen sijasta kirjallisuuskatsauksen laajuutta rajattiin suorittamalla artikkelien etsintä ainoastaan ACM Digital Library -tietokantaan. Toisena tietokonegrafiikan kannalta merkittävänä tietokantana voidaan pitää Wiley Online Library -tietokantaa, mikä jätettiin pois tutkielman laajuus huomioiden. Kaikkiaan kirjallisuuskatsaukseen sisällytettiin 17 säteenseurantaa reaaliaikaisessa renderöinnissä käsittelevää tutkimusartikkelia. Artikkeleissa esiintyvien menetelmien esittelemiseksi artikkelit jaettiin viitteellisesti niissä esiintyvien menetelmien perusteella kolmeen pääkategoriaan: näytteistämiseen, häiriönpoistoon ja muuhun. Jakoa näytteistämisen ja häiriönpoiston välillä voidaan pitää jokseenkin häilyvänä molempien pyrkiessä varianssin vähentämiseen ja osassa menetelmistä osittain samoin keinoin.

Kirjallisuuskatsauksessa näytteistäminen kategoriaan luokitelluissa artikkeleissa havaittuina merkittävinä haasteina reaaliaikaisessa renderöinnissä voidaan pitää monivalo-ongelmaa, kuin myös useamman kimmokkeen epäsuoran valaistuksen kuvaamista, kun taas näytteistämisen tavoitteena voidaan pitää harhattoman tai likimain harhattoman ja alhaisen varianssin estimaatin tuottamista. Siinä, missä osassa tutkielman aineiston videopelejä monivalo-ongelmaa vältteltiin rajaamalla tiettyjen valon ilmenemismuotojen kuvaaminen yksittäiseen valonlähteeseen, esitellään kirjallisuuskatsauksen artikkeleissa monivalo-ongelman ratkaisuksi useampia valohierarkioita, kuin myös ReSTIR-menetelmä. Valohierarkioiden ratkaistaessa monivalo-ongelma muuntamalla näytteistettävän valonlähteen valinta hierarkkiseksi painotetuksi näytteistämiseksi, voi valohierarkian ylläpitäminen osoittautua aikaa vieväksi virtuaalimaailman sisältäessä useita dynaamisia valonlähteitä.

Kenties mielenkiintoisempana menetelmänä monivalo-ongelman ratkaisemiseen voidaan pitää RIS-menetelmää tila-ajallisesti soveltavaa ReSTIR-menetelmää (Bitterli ym. 2020), joka ei myöskään poissulje valohierarkian käyttöä. ReSTIR-menetelmä on hyvä esimerkki siitä, miten kompleksisissakin virtuaalimaailmoissa, joissa painotettua näytteistämistä ei suo-

raan kyetä käyttämään, voidaan pienellä määrällä seurattuja säteitä saavuttaa vakuuttavia tuloksia. Monivalo-ongelman ohella ReSTIR-menetelmää voidaan hyödyntää useamman kimmokkeen epäsuoran valaistuksen kuvaamiseen (Ouyang ym. 2021) (Lin ym. 2022), kuin myös kirjallisuuskatsauksessa esiteltyä NRC-menetelmää, jossa epäsuoraa valaistusta approksimoidaan suorituksen aikana koulutettavalla neuroverkolla.

Siinä, missä ReSTIR-menetelmä hyödyntää tila-ajallista kandidaattinäytteiden uudelleenkäyttöä, on tila-ajallisesta näytteiden uudelleenkäytöstä muodostunut normi häiriönpoistossa kehyksellä tuotettujen uusien näytteiden alhaisen lukumäärän kompensoimiseksi. Vaikka tila-ajallinen uudelleenkäyttö mahdollistaa efektiivisen näytemäärän kasvattamisen, luo se tyypillisesti harhan, mikä tulee huomioida tila-ajallista näytteiden uudelleenkäyttöä hyödyntävissä menetelmissä pyrkimällä valitsemaan uudelleenkäytettävät näytteet siten, että niiden ratkaisuavaruudet ovat vähintäänkin likimain samat. Lisäksi ajallisessa näytteiden uudelleenkäytössä dynaamisessa virtuaalimaailmassa, kuin myös virtuaalikameran liikkuessa, riskinä on alhainen efektiivinen näytemäärä osassa pikseleitä, kuin myös haamukuvat virtuaalimaailman valaistuksen muuttuessa. Näin ollen ajallisesti ja tila-ajallisesti näytteitä uudelleen käyttävien menetelmien haasteena voidaan pitää ajallista vakautta, kuin myös oikeallisuutta.

Tila-ajallisen näytteiden uudelleenkäytön lisäksi kirjallisuuskatsauksessa esiteltyjä häiriönpoistomenetelmiä yhdistää rasteroinnilla tuotettujen häiriöttömien kuvapuskurien käyttö filtteröinnin ohjaamisessa, kuin myös säteenseurannalla kuvattujen valon eri ilmenemismuotojen käsittely erikseen. Rasteroinnilla tuotettujen häiriöttömien kuvapuskurien käyttö jo itsessään takaa sen, ettei säteenseuranta tule kokonaan rasterointia korvaamaan. Se, miten filtteröinnissä käytettävien filttereiden ytimien sisältämien painokertoimien muodostaminen tapahtuu häiriöttömien kuvapuskurien avulla, riippuu menetelmästä. Tavoitteena filtteröinnin ohjaamisessa on poistaa häiriö sumentamatta virtuaalimaailman kappaleiden välisiä rajoja. Kirjallisuuskatsauksessa esitellyissä häiriönpoistomenetelmissä filtterin ytimen sisältämien painokertoimien muodostamiseen hyödynnettiin muun muassa lineaarista regressioanalyysia kuin myös tekoälyä.

Tekoälyn kehittyessä nopeasti viime vuosina, näkyy se myös tietokonegrafiikassa säteenseurannan yhteydessä tekoälypohjaisten menetelmien yleistymisenä. Siinä, missä ei-reaaliaikaisessa renderöinnissä tekoälypohjaiset häiriönpoistomenetelmät ovat laajasti käytettyjä (Tho-

mas ym. 2022), mistä yhtenä esimerkkinä voidaan pitää Pixarin tuotannossa käyttämää tekoälypohjaista häiriönpoistomenetelmää (Dahlberg, Adler ja Newlin 2019), ovat reaaliaikaisessa renderöinnissä erityisesti videopeleissä käytettyjen NVIDIA:n DLSS ja Intelin X$^{e}$SS -teknologioiden myötä puolestaan tekoälypohjaiset ylösskaalausmenetelmät vakiinnuttaneet asemansa. Säteenseurantaa hyödyntävissä sovelluksissa, kuten videopeleissä, ylösskaalaus on merkittävässä roolissa korkean kehysnopeuden mahdollistamisessa säteenseurannan kompleksisuuden kasvaessa lineaarisesti pikseleiden lukumäärän myötä. Ylösskaalauksen lisäksi kirjallisuuskatsauksessa esitellyissä menetelmissä tekoälyä hyödynnettiin häiriönpoistoon sekä useamman kimmokkeen epäsuoran valaistuksen approksimointiin.

Yleisesti laitteistokiihdytteinen säteenseuranta voidaan katsoa tervetulleeksi uudistukseksi reaaliaikaisen renderöinnin rikastuttamiseksi, mutta samaan aikaan sen rajoittuvuudet reaaliaikaiseen renderöintiin tulee huomioida. Säteenseurannan ytimenä voidaan pitää mahdollisuutta suorittaa näkyvyystarkastelu mielivaltaisista pisteistä mielivaltaisiin suuntiin virtuaalimaailmassa, mikä tekee säteenseurannasta luonnollisen vaihtoehdon valon eri ilmenemismuotojen kuvaamiseen, mikä näkyy myös kirjallisuuskatsauksessa valon ilmenemismuoto -spesifisten menetelmien puutteena. Valon ilmenemismuoto -spesifisten menetelmien sijasta pääpainoksi säteenseurannan tutkimuksessa havaittiin Monte Carlo -estimaattorin varianssin vähentäminen niin näytteistämisen kuin myös häiriönpoisto menetelmin. Laitteistokiihdytyksestä huolimatta säteenseuranta on nykyisellääänkin laskennallisesti liian raskas, jotta esimerkiksi puhtaasti polunseurantamenetelmällä kyettäisiin useamman kimmokkeen valon kulkeutuminen kuvaamaan reaaliajassa kompleksisessa virtuaalimaailmassa. Näin ollen, vaikka rasteroinnin ja säteenseurannan välille voidaan luoda vastakkainasettelua tekniikkoina, ei säteenseurantaa voida eristää rasteroinnista nykyisissä reaaliaikaisen renderöinnin sovelluksissa. Kuten mainittiin, pelkästään jo häiriöttömien kuvapuskurien käyttö häiriönpoistossa luo tarpeen rasteroinnille ja samalla voidaan näkyvyys suorittaa rasteroinnilla, jolloin reaaliaikaisessa renderöinnissä hybriditekniikan uskotaan säilyttävän asemansa lähitulevaisuudessa.

Se, miten säteenseuranta tulee kehittymään ja syrjäyttääkö säteenseurantatekniikka rasterointitekniikan käytön valon ilmenemismuotojen kuvaamisessa reaaliaikaisessa renderöinnissä, tulee aika näyttämään. Katsottaessa miten rasterointipohjaiset menetelmät kuin myös

rasterointia kiihdyttävä laitteisto ovat kehittyneet vuosien saatossa, voidaan myös säteenseurannan olettaa kehittyvän kenties merkittävästikin vielä tulevaisuudessa. On toki hyvä muistaa, että säteenseurannalla on jokseenkin pitkä historia ei-reaaliaikaisessa renderöinnissä, jolloin teoria kuin myös menetelmät ovat osittain jo hyvinkin kehittyneitä. Kirjallisuuskatsauksessa tämä ilmeni useampien aikaisempien menetelmien, kuten välitön radiositeetti menetelmän ja radianssin varastointimenetelmän, soveltamisena reaaliaikaisessa renderöinnissä. Tulevaisuudessa säteenseurannan kehityksen kannalta merkittäviä tekijöitä uskotaan olevan laitteiston ja tekoälypohjaisten menetelmien kehittyminen. Siinä, missä laitteiston kehittyminen mahdollistaisi varianssin vähentämisen näytemäärää kasvattamalla ja sitä kautta toisi se osittaisen helpotuksen häiriönpoistolle, voitaisiin tekoälypohjaisia menetelmiä hyödyntää vaihtoehtoisesti kompensoimaan laitteiston säteenseurannan suorituskykyä. Toisaalta reaaliaikaisten tekoälypohjaisten menetelmien kehittymisenkin voidaan katsoa olevan ainakin osittain riippuvainen laitteiston kehittymisestä. Joka tapauksessa, säteenseurannan yleistyessä erityisesti ei korkeaan kehysnopeuteen tähtäävissä videopeleissä, uskotaan reaaliaikaisen säteenseurannan kehittyvän entisestään ja säilyvän tutkimuskohteena myös tulevaisuudessa.

# Lähteet

ACM. 2023. “ACM History”. Viitattu 2. huhtikuuta 2023. https://www.acm.org/about-acm/acm-history.

“Advanced API Performance: Async Compute and Overlap”. 2022. Viitattu 8. lokakuuta 2022. https://developer.nvidia.com/blog/advanced-api-performance-async-compute-and-overlap/.

Akenine-Möller, Tomas, Eric Haines ja Naty Hoffman. 2018. *Real-Time Rendering, Fourth Edition.* 4th. USA: A. K. Peters, Ltd. ISBN: 0134997832.

“AMD FidelityFX™ Supported Games”. 2022. Viitattu 18. syyskuuta 2022. https://www.amd.com/en/technologies/radeon-software-fidelityfx-supported-games.

Andersson, P., J. Nilsson, M. Salvi, J. Spjut ja T. Akenine-Möller. 2019. “Temporally Dense Ray Tracing”. Teoksessa *Proceedings of the Conference on High-Performance Graphics,* 33–38. HPG ’19. Strasbourg, France: Eurographics Association. https://doi.org/10.2312/hpg.20191193. https://doi.org/10.2312/hpg.20191193.

Appel, Arthur. 1968. “Some techniques for shading machine renderings of solids”.

Bavoil, Louis. 2014. *Deinterleaved Texturing for Cache-Efficient Interleaved Sampling.* Tekninen raportti.

Bishop, Gary, Henry Fuchs, Leonard McMillan ja Ellen J. Scher Zagier. 1994. “Frameless Rendering: Double Buffering Considered Harmful”. Teoksessa *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques,* 175–176. SIGGRAPH ’94. New York, NY, USA: Association for Computing Machinery. ISBN: 0897916670. https://doi.org/10.1145/192161.192195. https://doi.org/10.1145/192161.192195.

Bitterli, Benedikt, Chris Wyman, Matt Pharr, Peter Shirley, Aaron Lefohn ja Wojciech Jarosz. 2020. “Spatiotemporal Reservoir Resampling for Real-Time Ray Tracing with Dynamic Direct Lighting”. *ACM Trans. Graph.* (New York, NY, USA) 39, numero 4 (heinäkuu). ISSN: 0730-0301. https://doi.org/10.1145/3386569.3392481. https://doi-org.ezproxy.jyu.fi/10.1145/3386569.3392481.

Blinn, James F. 1977. “Models of Light Reflection for Computer Synthesized Pictures”. Teoksessa *Proceedings of the 4th Annual Conference on Computer Graphics and Interactive Techniques,* 192–198. SIGGRAPH ’77. San Jose, California: Association for Computing Machinery. ISBN: 9781450373555. https://doi.org/10.1145/563858.563893. https://doi.org/10.1145/563858.563893.

Boissé, Guillaume. 2021. “WORLD-SPACE SPATIOTEMPORAL RESERVOIR REUSE FOR RAY-TRACED GLOBAL ILLUMINATION”. Teoksessa *SIGGRAPH Asia 2021 Technical Communications.* SA ’21 Technical Communications. Tokyo, Japan: Association for Computing Machinery. ISBN: 9781450390736. https://doi.org/10.1145/3478512.3488613. https://doi-org.ezproxy.jyu.fi/10.1145/3478512.3488613.

Burgess, John. 2020. “RTX on—The NVIDIA Turing GPU”. *IEEE Micro* 40 (2): 36–44. https://doi.org/10.1109/MM.2020.2971677.

Burt, Peter J. 1981. “Fast filter transform for image processing”. *Computer Graphics and Image Processing* 16 (1): 20–51. ISSN: 0146-664X. https://doi.org/https://doi.org/10.1016/0146-664X(81)90092-7. https://www.sciencedirect.com/science/article/pii/0146664X81900927.

Christensen, Per, Julian Fong, Jonathan Shade, Wayne Wooten, Brenden Schubert, Andrew Kensler, Stephen Friedman ym. 2018. “RenderMan: An Advanced Path-Tracing Architecture for Movie Rendering”. *ACM Trans. Graph.* (New York, NY, USA) 37 (3). ISSN: 0730-0301. https://doi.org/10.1145/3182162. https://doi.org/10.1145/3182162.

Christensen, Per H., ja Wojciech Jarosz. 2016. “The Path to Path-Traced Movies”. *Foundations and Trends in Computer Graphics and Vision* 10, numero 2 (lokakuu): 103–175. ISSN: 1572-2740. https://doi.org/10/gfjwjc.

Conty Estevez, Alejandro, ja Christopher Kulla. 2018. “Importance Sampling of Many Lights with Adaptive Tree Splitting”. *Proc. ACM Comput. Graph. Interact. Tech.* (New York, NY, USA) 1, numero 2 (elokuu). https://doi.org/10.1145/3233305. https://doi.org/10.1145/3233305.

Cook, Robert L., Loren Carpenter ja Edwin Catmull. 1987. “The Reyes Image Rendering Architecture”. Teoksessa *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques,* 95–102. SIGGRAPH ’87. New York, NY, USA: Association for Computing Machinery. ISBN: 0897912276. https://doi.org/10.1145/37401.37414. https://doi.org/10.1145/37401.37414.

Cook, Robert L., Thomas Porter ja Loren Carpenter. 1984. “Distributed Ray Tracing”. *SIGGRAPH Comput. Graph.* (New York, NY, USA) 18 (3): 137–145. ISSN: 0097-8930. https://doi.org/10.1145/964965.808590. https://doi.org/10.1145/964965.808590.

Corporation, NVIDIA. 2017. *NVIDIA Tesla V100 GPU architecture, Tech. Rep. WP-08608-001 v1.1.* Tekninen raportti.

———. 2018. *NVIDIA Turing GPU Architecture, Tech. Rep. WP-09183-001 v01.* Tekninen raportti.

———. 2021. *NVIDIA Ampere GA102 GPU Architecture, V2.0.* Tekninen raportti.

———. 2022a. *NVIDIA Ada GPU Architecture, V1.01.* Tekninen raportti.

———. 2022b. *NVIDIA Ampere GPU Architecture Compatibility Guide for CUDA Applications, Tech. Rep. DA-09074-001_v11.7.* Tekninen raportti.

———. 2023. *NVIDIA Ada GPU Architecture, V2.01.* Tekninen raportti.

Dahlberg, Henrik, David Adler ja Jeremy Newlin. 2019. “Machine-Learning Denoising in Feature Film Production”. Teoksessa *ACM SIGGRAPH 2019 Talks.* SIGGRAPH ’19. Los Angeles, California: Association for Computing Machinery. ISBN: 9781450363174. https://doi.org/10.1145/3306307.3328150. https://doi.org/10.1145/3306307.3328150.

Dammertz, Holger, Daniel Sewtz, Johannes Hanika ja Hendrik P. A. Lensch. 2010. “Edge-Avoiding À-Trous Wavelet Transform for Fast Global Illumination Filtering”. Teoksessa *Proceedings of the Conference on High Performance Graphics,* 67–75. HPG ’10. Saarbrucken, Germany: Eurographics Association.

DICE, EA. 2019. “It Just Works: Ray-Traced Reflections in ’Battlefield V’”. GDC 2019. https://www.gdcvault.com/play/1026016/It-Just-Works-Ray-Traced.

Dutre, Philip, Kavita Bala, Philippe Bekaert ja Peter Shirley. 2006. *Advanced Global Illumination.* AK Peters Ltd. ISBN: 1568813074.

Engel, Wolfgang. 2017. *GPU Zen: Advanced Rendering Techniques.* Bowker Identifier Services. ISBN: 0998822892.

"Far Cry 6 Benchmarked". 2021. Viitattu 28. helmikuuta 2022. https://www.techspot.com/article/2339-far-cry-6-benchmarks/.

Fattal, Raanan, Maneesh Agrawala ja Szymon Rusinkiewicz. 2007. "Multiscale Shape and Detail Enhancement from Multi-Light Image Collections". *ACM Trans. Graph.* (New York, NY, USA) 26, numero 3 (heinäkuu): 51–es. ISSN: 0730-0301. https://doi.org/10.1145/1276377.1276441. https://doi.org/10.1145/1276377.1276441.

Fernando, Randima. 2004. *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics.* Pearson Higher Education. ISBN: 0321228324.

"FidelityFX CACAO - GitHub". 2022. Viitattu 22. syyskuuta 2022. https://github.com/GPUOpen-Effects/FidelityFX-CACAO.

Forsyth, David. 2018. *Probability and statistics for computer science.* Springer.

Games, 4A. 2019. "Exploring the Ray Traced Future in 'Metro Exodus'". GDC 2019. https://www.gdcvault.com/play/1026159/Exploring-the-Ray-Traced-Future.

Greger, Gene, Peter Shirley, P.M. Hubbard ja D.P. Greenberg. 1998. "The Irradiance Volume". *Computer Graphics and Applications, IEEE* 18 (huhtikuu): 32–43. https://doi.org/10.1109/38.656788.

"GTC Sept 2022 Keynote with NVIDIA CEO Jensen Huang". 2022. Viitattu 27. syyskuuta 2022. https://www.nvidia.com/gtc/keynote/.

Haines, Eric, ja Tomas Akenine-Möller, toimittaneet. 2019. "Ray Tracing Terminology". Teoksessa *Ray Tracing Gems: High-Quality and Real-Time Rendering with DXR and Other APIs,* 7–14. Berkeley, CA: Apress. ISBN: 978-1-4842-4427-2. https://doi.org/10.1007/978-1-4842-4427-2_1. https://doi.org/10.1007/978-1-4842-4427-2_1.

Hasselgren, J., J. Munkberg, M. Salvi, A. Patney ja A. Lefohn. 2020. “Neural Temporal Adaptive Sampling and Denoising”. *Computer Graphics Forum* 39 (2): 147–155. https://doi.org/https://doi.org/10.1111/cgf.13919. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13919. https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13919.

Heitz, Eric, Jonathan Dupuy, Stephen Hill ja David Neubelt. 2016. “Real-Time Polygonal-Light Shading with Linearly Transformed Cosines”. *ACM Trans. Graph.* (New York, NY, USA) 35, numero 4 (heinäkuu). ISSN: 0730-0301. https://doi.org/10.1145/2897824.2925895. https://doi.org/10.1145/2897824.2925895.

Holschneider, Matthias, Richard Kronland-Martinet, J. Morlet ja Ph Tchamitchian. 1989. “A Real-Time Algorithm for Signal Analysis with the Help of the Wavelet Transform”. *Wavelets, Time-Frequency Methods and Phase Space* -1 (tammikuu): 286. https://doi.org/10.1007/978-3-642-75988-8_28.

Interactive, IO. 2022. “Bringing 4K Ray-Traced Visuals to the World of HITMAN 3”. GDC 2022. https://www.youtube.com/watch?v=dG-hctI6ykw.

ITU-R. 2015. *Recommendation BT.709-6, R. Parameters value for the HDTV standards for production and international program exchange.*

Kajiya, James T. 1986. “The Rendering Equation”. Teoksessa *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques,* 143–150. SIGGRAPH ’86. New York, NY, USA: Association for Computing Machinery. ISBN: 0897911962. https://doi.org/10.1145/15922.15902. https://doi.org/10.1145/15922.15902.

Keating, Michael P. 2002. “Chapter 1 - Optics, Light, and Vision”. Teoksessa *Geometric, Physical, and Visual Optics (Second Edition),* Second Edition, toimittanut Michael P. Keating, 1–11. Burlington: Butterworth-Heinemann. ISBN: 978-0-7506-7262-7. https://doi.org/https://doi.org/10.1016/B978-0-7506-7262-7.50005-9. https://www.sciencedirect.com/science/article/pii/B9780750672627500059.

Keller, Alexander. 1997. “Instant Radiosity”. Teoksessa *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques,* 49–56. SIGGRAPH ’97. USA: ACM Press/Addison-Wesley Publishing Co. ISBN: 0897918967. https://doi.org/10.1145/258734.258769. https://doi.org/10.1145/258734.258769.

Keller, Alexander, ja Wolfgang Heidrich. 2002. “Interleaved Sampling”. *Proc. of the 12th Eurographics Workshop on Rendering* (maaliskuu). https://doi.org/10.2312/EGWR/EGWR01/269-276.

Kelly, Patrick, Yuriy O’Donnell, Kenzo Elst, Juan Cañada ja Evan Hart. 2021. “Ray Tracing in Fortnite”, 791–821. Elokuu. ISBN: 978-1-4842-7184-1. https://doi.org/10.1007/978-1-4842-7185-8_48.

Koskela, Matias, Kalle Immonen, Markku Mäkitalo, Alessandro Foi, Timo Viitanen, Pekka Jääskeläinen, Heikki Kultala ja Jarmo Takala. 2019. “Blockwise Multi-Order Feature Regression for Real-Time Path-Tracing Reconstruction”. *ACM Trans. Graph.* (New York, NY, USA) 38, numero 5 (kesäkuu). ISSN: 0730-0301. https://doi.org/10.1145/3269978. https://doi.org/10.1145/3269978.

Krivanek, J., P. Gautron, S. Pattanaik ja K. Bouatouch. 2005. “Radiance caching for efficient global illumination computation”. *IEEE Transactions on Visualization and Computer Graphics* 11 (5): 550–561. https://doi.org/10.1109/TVCG.2005.83.

Lafortune, Eric, ja Yves Willems. 1998. “Bi-Directional Path Tracing”. *Proceedings of Third International Conference on Computational Graphics and Visualization Techniques (Compugraphics’* 93 (tammikuu).

Lambru, Cristian, Anca Morar, Florica Moldoveanu, Victor Asavei ja Alin Moldoveanu. 2021. “Comparative Analysis of Real-Time Global Illumination Techniques in Current Game Engines”. *IEEE Access* 9:125158–125183. https://doi.org/10.1109/ACCESS.2021.3109663.

Lauterbach, Christian, Michael Garland, Shubhabrata Sengupta, David Luebke ja Dinesh Manocha. 2009. “Fast BVH construction on gpus”. *Computer Graphics Forum* 28 (huhtikuu): 375–384. https://doi.org/10.1111/j.1467-8659.2009.01377.x.

Lawson ja J. Zhu. 2002. “Image compression using wavelets and JPEG 2000: A Tutorial”. *Electronics & Communication Engineering Journal* 14 (heinäkuu). https://doi.org/10.1049/ecej:20020303.

Lee, Won-Jong, ja Gabor Liktor. 2020. “Lazy Build of Acceleration Structures with Traversal Shaders”. Teoksessa *SIGGRAPH Asia 2020 Technical Communications.* SA ’20. Virtual Event, Republic of Korea: Association for Computing Machinery. ISBN: 9781450380805. https://doi.org/10.1145/3410700.3425430. https://doi.org/10.1145/3410700.3425430.

Lee, Won-Jong, Gabor Liktor ja Karthik Vaidyanathan. 2019. “Flexible Ray Traversal with an Extended Programming Model”. Teoksessa *SIGGRAPH Asia 2019 Technical Briefs,* 17–20. SA ’19. Brisbane, QLD, Australia: Association for Computing Machinery. ISBN: 9781450369459. https://doi.org/10.1145/3355088.3365149. https://doi.org/10.1145/3355088.3365149.

Lin, Daqi, Markus Kettunen, Benedikt Bitterli, Jacopo Pantaleoni, Cem Yuksel ja Chris Wyman. 2022. “Generalized Resampled Importance Sampling: Foundations of ReSTIR”. *ACM Trans. Graph.* (New York, NY, USA) 41, numero 4 (heinäkuu). ISSN: 0730-0301. https://doi.org/10.1145/3528223.3530158. https://doi.org/10.1145/3528223.3530158.

Lin, Daqi, Chris Wyman ja Cem Yuksel. 2021. “Fast Volume Rendering with Spatiotemporal Reservoir Resampling”. *ACM Trans. Graph.* (New York, NY, USA) 40, numero 6 (joulukuu). ISSN: 0730-0301. https://doi.org/10.1145/3478513.3480499. https://doi.org/10.1145/3478513.3480499.

Lin, Daqi, ja Cem Yuksel. 2019. “Real-Time Rendering with Lighting Grid Hierarchy”. *Proc. ACM Comput. Graph. Interact. Tech.* (New York, NY, USA) 2, numero 1 (kesäkuu). https://doi.org/10.1145/3321361. https://doi.org/10.1145/3321361.

———. 2020. “Real-Time Stochastic Lightcuts”. *Proc. ACM Comput. Graph. Interact. Tech.* (New York, NY, USA) 3, numero 1 (huhtikuu). https://doi.org/10.1145/3384543. https://doi.org/10.1145/3384543.

Lindholm, Erik, John Nickolls, Stuart Oberman ja John Montrym. 2008. “NVIDIA Tesla: A Unified Graphics and Computing Architecture”. *IEEE Micro* 28 (2): 39–55. https://doi.org/10.1109/MM.2008.31.

Liu, Lufei, Wesley Chang, Francois Demoullin, Yuan Hsi Chou, Mohammadreza Saed, David Pankratz, Tyler Nowicki ja Tor M. Aamodt. 2021. “Intersection Prediction for Accelerated GPU Ray Tracing”. Teoksessa *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture,* 709–723. MICRO ’21. Virtual Event, Greece: Association for Computing Machinery. ISBN: 9781450385572. https://doi.org/10.1145/3466752.3480097. https://doi.org/10.1145/3466752.3480097.

Lyon, Arkane Studios. 2022. “A Guided Tour of Blackreef: Rendering Technologies in Deathloop”. GDC 2022. https://www.youtube.com/watch?v=Vn4Q4XeMyBk.

MachineGames. 2020. “Ray Traced Reflections in ’Wolfenstein: Youngblood’”. GDC 2020. Viitattu 28. helmikuuta 2022. https://www.gdcvault.com/play/1026723/Ray-Traced-Reflections-in-Wolfenstein.

Mallat, S.G. 1989. “A theory for multiresolution signal decomposition: the wavelet representation”. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 11 (7): 674–693. https://doi.org/10.1109/34.192463.

Marrs, Adam, Josef Spjut, Holger Gruen, Rahul Sathe ja Morgan McGuire. 2018. “Adaptive Temporal Antialiasing”. Teoksessa *Proceedings of the Conference on High-Performance Graphics.* HPG ’18. Vancouver, British Columbia, Canada: Association for Computing Machinery. ISBN: 9781450358965. https://doi.org/10.1145/3231578.3231579. https://doi.org/10.1145/3231578.3231579.

Meister, Daniel, Jakub Boksansky, Michael Guthe ja Jiri Bittner. 2020. “On Ray Reordering Techniques for Faster GPU Ray Tracing”. Teoksessa *Symposium on Interactive 3D Graphics and Games.* I3D ’20. San Francisco, CA, USA: Association for Computing Machinery. ISBN: 9781450375894. https://doi.org/10.1145/3384382.3384534. https://doi.org/10.1145/3384382.3384534.

“Microsoft DirectX Raytracing Functional Spec”. 2022. Viitattu 12. elokuuta 2022. https://microsoft.github.io/DirectX-Specs/d3d/Raytracing.html.

Mittring, Martin. 2007. “Finding Next Gen: Cryengine 2”. Elokuu. https://doi.org/10.1145/1281500.1281671.

Mobley, Michael. 2005. “The nature of light: Description of photon diffraction based upon virtual particle exchange”. *Proceedings of SPIE - The International Society for Optical Engineering* 5866 (elokuu). https://doi.org/10.1117/12.637439.

Moreau, P., M. Pharr ja P. Clarberg. 2019. “Dynamic Many-Light Sampling for Real-Time Ray Tracing”. Teoksessa *Proceedings of the Conference on High-Performance Graphics,* 21–26. HPG ’19. Strasbourg, France: Eurographics Association. https://doi.org/10.2312/hpg.20191191. https://doi.org/10.2312/hpg.20191191.

Morgan McGuire, NVIDIA. 2019. “Ray-Traced Irradiance Fields”. GDC 2019. https://www.gdcvault.com/play/1026182/.

Müller, Thomas, Fabrice Rousselle, Jan Novák ja Alexander Keller. 2021. “Real-Time Neural Radiance Caching for Path Tracing”. *ACM Trans. Graph.* (New York, NY, USA) 40, numero 4 (heinäkuu). ISSN: 0730-0301. https://doi.org/10.1145/3450626.3459812. https://doi.org/10.1145/3450626.3459812.

NVIDIA, Nixxes. 2019. “"Shadows"of the Tomb Raider: Ray Tracing Deep Dive”. GDC 2019. https://www.gdcvault.com/play/1026163/-Shadows-of-the-Tomb.

“NVIDIA GeForce RTX 2060 Specification”. 2022. Viitattu 30. kesäkuuta 2022. https://www.nvidia.com/en-gb/geforce/news/gfecnt/nvidia-geforce-rtx-2060/.

“NVIDIA Real-Time Denoisers (NRD)”. 2021. Viitattu 14. maaliskuuta 2022. https://developer.nvidia.com/rtx/ray-tracing/rt-denoisers.

“NVIDIA RTX: Best Practices”. 2019. Viitattu 12. elokuuta 2022. https://www.eurogamer.net/digitalfoundry-2019-wolfenstein-youngblood-tech-analysis.

“NVIDIA RTX: List Of All Games, Engines And Applications Featuring GeForce RTX-Powered Technology”. 2022. Viitattu 18. syyskuuta 2022. https://www.nvidia.com/en-us/geforce/news/nvidia-rtx-games-engines-apps/.

“NVIDIA Turing tuning guide”. 2022. Viitattu 1. lokakuuta 2022. https://docs.nvidia.com/cuda/turing-tuning-guide/index.html.

Oberberger, Max, Matthäus G. Chajdas ja Rüdiger Westermann. 2022. “Spatiotemporal Variance-Guided Filtering for Motion Blur”. *Proc. ACM Comput. Graph. Interact. Tech.* (New York, NY, USA) 5 (3). https://doi.org/10.1145/3543871. https://doi.org/10.1145/3543871.

Ouyang, Y., S. Liu, M. Kettunen, M. Pharr ja Jacopo Pantaleoni. 2021. “ReSTIR GI: Path Resampling for Real-Time Path Tracing”. *Computer Graphics Forum* 40 (joulukuu): 17–29. https://doi.org/10.1111/cgf.14378.

Pankratz, David, Tyler Nowicki, Ahmed Eltantawy ja José Nelson Amaral. 2021. “Vulkan Vision: Ray Tracing Workload Characterization using Automatic Graphics Instrumentation”. Teoksessa *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO),* 137–149. https://doi.org/10.1109/CGO51591.2021.9370320.

Paris, Sylvain, Pierre Kornprobst, Jack Tumblin ja Frédo Durand. 2009. “Bilateral Filtering: Theory and Applications”. *Foundations and Trends in Computer Graphics and Vision* 4 (tammikuu): 1–74. https://doi.org/10.1561/0600000020.

Patney, Anjul, Marco Salvi, Joohwan Kim, Anton Kaplanyan, Chris Wyman, Nir Benty, David Luebke ja Aaron Lefohn. 2016. “Towards Foveated Rendering for Gaze-Tracked Virtual Reality”. *ACM Trans. Graph.* (New York, NY, USA) 35 (6). ISSN: 0730-0301. https://doi.org/10.1145/2980179.2980246. https://doi.org/10.1145/2980179.2980246.

Peters, Christoph. 2021. “BRDF Importance Sampling for Polygonal Lights”. *ACM Trans. Graph.* (New York, NY, USA) 40, numero 4 (heinäkuu). ISSN: 0730-0301. https://doi.org/10.1145/3450626.3459672. https://doi.org/10.1145/3450626.3459672.

Peters, Christoph, ja Carsten Dachsbacher. 2019. “Sampling Projected Spherical Caps in Real Time”. *Proc. ACM Comput. Graph. Interact. Tech.* (New York, NY, USA) 2 (1). https://doi.org/10.1145/3320282. https://doi.org/10.1145/3320282.

Phong, Bui Tuong. 1975. “Illumination for Computer Generated Pictures”. *Commun. ACM* (New York, NY, USA) 18 (6): 311–317. ISSN: 0001-0782. https://doi.org/10.1145/360825.360839. https://doi.org/10.1145/360825.360839.

Productions, Luminous. 2022. “Breaking Down the World of Athia: The Technologies of Forspoken”. GDC 2022. https://www.youtube.com/watch?v=MlTohmB4Gh4.

Ramamoorthi, Ravi, ja Pat Hanrahan. 2001. “An efficient representation for irradiance environment maps”. Teoksessa *Proceedings of the 28th annual conference on Computer graphics and interactive techniques,* 497–500.

“Ray Tracing Gems Series”. 2021. Viitattu 14. maaliskuuta 2022. https://www.realtimerendering.com/raytracinggems/.

“RenderMan 21 Documentation”. 2016. Viitattu 19. marraskuuta 2022. https://rmanwiki.pixar.com/display/REN21/RenderMan+21+Documentation.

Ronneberger, Olaf, Philipp Fischer ja Thomas Brox. 2015. *U-Net: Convolutional Networks for Biomedical Image Segmentation.* arXiv: 1505.04597 [cs.CV].

Roychoudhuri, C., A.F. Kracklauer ja K. Creath. 2008. *The Nature of Light: What is a Photon?* Optical Science and Engineering. CRC Press. ISBN: 9780367387105.

Rubin, Donald B. 1987. “Comment : A noniterative sampling/importance resampling alternative to the data augmentation algorithm for creating a few imputations when fractions of missing information are modest : The SIR Algorithm”.

Rusch, Matthew, Neil Bickford ja Nuno Subtil. 2021. “Introduction to Vulkan Ray Tracing”. Teoksessa *Ray Tracing Gems II: Next Generation Real-Time Rendering with DXR, Vulkan, and OptiX,* toimittanut Adam Marrs, Peter Shirley ja Ingo Wald, 213–255. Berkeley, CA: Apress. ISBN: 978-1-4842-7185-8. https://doi.org/10.1007/978-1-4842-7185-8_16. https://doi.org/10.1007/978-1-4842-7185-8_16.

Saito, Takafumi, ja Tokiichiro Takahashi. 1990. “Comprehensible Rendering of 3-D Shapes”. Teoksessa *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques,* 197–206. SIGGRAPH ’90. Dallas, TX, USA: Association for Computing Machinery. ISBN: 0897913442. https://doi.org/10.1145/97879.97901. https://doi.org/10.1145/97879.97901.

Sanzharov, V. V., V. A. Frolov ja V. A. Galaktionov. 2020. “Survey of Nvidia RTX Technology”. *Program. Comput. Softw.* (USA) 46, numero 4 (heinäkuu): 297–304. ISSN: 0361-7688. https://doi.org/10.1134/S0361768820030068. https://doi.org/10.1134/S0361768820030068.

Schied, Christoph, Anton Kaplanyan, Chris Wyman, Anjul Patney, Chakravarty R. Alla Chaitanya, John Burgess, Shiqiu Liu, Carsten Dachsbacher, Aaron Lefohn ja Marco Salvi. 2017. "Spatiotemporal Variance-Guided Filtering: Real-Time Reconstruction for Path-Traced Global Illumination". Teoksessa *Proceedings of High Performance Graphics.* HPG '17. Los Angeles, California: Association for Computing Machinery. ISBN: 9781450351010. https://doi.org/10.1145/3105762.3105770. https://doi.org/10.1145/3105762.3105770.

Schied, Christoph, Christoph Peters ja Carsten Dachsbacher. 2018. "Gradient Estimation for Real-Time Adaptive Temporal Filtering". *Proc. ACM Comput. Graph. Interact. Tech.* (New York, NY, USA) 1, numero 2 (elokuu). https://doi.org/10.1145/3233301. https://doi.org/10.1145/3233301.

Schlick, Christopher M. 1994. "An Inexpensive BRDF Model for Physically-based Rendering". *Computer Graphics Forum* 13.

Silvennoinen, Ari, ja Ville Timonen. 2015. "Multi-Scale Global Illumination in Quantum Break". Teoksessa *ACM SIGGRAPH Advances in Real-Time Rendering Course.* Los Angeles, California: ACM. http://wili.cc/research/quantum_break/.

Sjöholm, Juha, Paula Jukarainen ja Tatu Aalto. 2021. "Ray Tracing in Control". Teoksessa *Ray Tracing Gems II: Next Generation Real-Time Rendering with DXR, Vulkan, and OptiX,* toimittanut Adam Marrs, Peter Shirley ja Ingo Wald, 739–764. Berkeley, CA: Apress. ISBN: 978-1-4842-7185-8. https://doi.org/10.1007/978-1-4842-7185-8_46. https://doi.org/10.1007/978-1-4842-7185-8_46.

Sloan, Peter-Pike. 2008. "Stupid Spherical Harmonics (SH) Tricks". *Game Developers Conference* (tammikuu).

Soldado, Rosana Montes, ja Carlos Ureña Almagro. 2012. "An Overview of BRDF Models".

Sousa, Tiago Lincka, N. Kasyan ja Nicolas Schulz. 2012. "CryENGINE 3: Three Years of Work in Review".

Talbot, Justin F. 2005. "Importance Resampling for Global Illumination". Tutkielma.

Talbot, Justin F., David Cline ja Parris Egbert. 2005. “Importance Resampling for Global Illumination”. Teoksessa *Proceedings of the Sixteenth Eurographics Conference on Rendering Techniques,* 139–146. EGSR ’05. Konstanz, Germany: Eurographics Association. ISBN: 3905673231.

Tatzgern, Wolfgang, Benedikt Mayr, Bernhard Kerbl ja Markus Steinberger. 2020. “Stochastic Substitute Trees for Real-Time Global Illumination”. Teoksessa *Symposium on Interactive 3D Graphics and Games.* I3D ’20. San Francisco, CA, USA: Association for Computing Machinery. ISBN: 9781450375894. https://doi.org/10.1145/3384382.3384521. https://doi.org/10.1145/3384382.3384521.

Thomas, Manu Mathew, Gabor Liktor, Christoph Peters, Sungye Kim, Karthik Vaidyanathan ja Angus G. Forbes. 2022. “Temporally Stable Real-Time Joint Neural Denoising and Supersampling”. *Proc. ACM Comput. Graph. Interact. Tech.* (New York, NY, USA) 5, numero 3 (heinäkuu). https://doi.org/10.1145/3543870. https://doi.org/10.1145/3543870.

“Tips: Acceleration Structure Compaction”. 2021. Viitattu 1. maaliskuuta 2022. https://developer.nvidia.com/blog/tips-acceleration-structure-compaction.

Tomasi, C., ja R. Manduchi. 1998. “Bilateral filtering for gray and color images”. Teoksessa *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271),* 839–846. https://doi.org/10.1109/ICCV.1998.710815.

Toronto, Ubisoft. 2022. “Performant Reflective Beauty: Hybrid Raytracing with Far Cry 6”. GDC 2022. https://www.youtube.com/watch?v=nTZpKD600eQ.

Walter, Bruce, Sebastian Fernandez, Adam Arbree, Kavita Bala, Michael Donikian ja Donald P. Greenberg. 2005. “Lightcuts: A Scalable Approach to Illumination”. *ACM Trans. Graph.* (New York, NY, USA) 24, numero 3 (heinäkuu): 1098–1107. ISSN: 0730-0301. https://doi.org/10.1145/1073204.1073318. https://doi.org/10.1145/1073204.1073318.

Walter, Bruce, Stephen Marschner, Hongsong Li ja Kenneth Torrance. 2007. “Microfacet Models for Refraction through Rough Surfaces”, 195–206. Tammikuu. https://doi.org/10.2312/EGWR/EGSR07/195-206.

Ward, Gregory J., Francis M. Rubinstein ja Robert D. Clear. 1988. “A Ray Tracing Solution for Diffuse Interreflection”. *SIGGRAPH Comput. Graph.* (New York, NY, USA) 22, numero 4 (kesäkuu): 85–92. ISSN: 0097-8930. https://doi.org/10.1145/378456.378490. https://doi.org/10.1145/378456.378490.

Veach, Eric. 1998. “Robust Monte Carlo Methods for Light Transport Simulation”. AAI9837162. Tohtorinväitöskirja.

Whitted, Turner. 1979. “An Improved Illumination Model for Shaded Display”. *SIGGRAPH Comput. Graph.* (New York, NY, USA) 13 (2): 14. ISSN: 0097-8930. https://doi.org/10.1145/965103.807419. https://doi.org/10.1145/965103.807419.

“How Wolfenstein: Youngblood scales from top-end PC to Nintendo Switch”. 2019. Viitattu 18. maaliskuuta 2022. https://developer.nvidia.com/blog/rtx-best-practices/.

“Vulkan 1.2.162.0 SDK release notes”. 2020. Viitattu 8. elokuuta 2022. https://vulkan.lunarg.com/doc/sdk/1.2.162.0/windows/release_notes.html.

“Vulkan® 1.3.224 - A Specification”. 2022. Viitattu 27. huhtikuuta 2023. https://registry.khronos.org/vulkan/specs/1.3-extensions/html/vkspec.html.

Xu, Tianchen, Xiaohua Ren ja Enhua Wu. 2019. “The Power of Box Filters: Real-Time Approximation to Large Convolution Kernel by Box-Filtered Image Pyramid”. Teoksessa *SIGGRAPH Asia 2019 Technical Briefs,* 1–4. SA ’19. Brisbane, QLD, Australia: Association for Computing Machinery. ISBN: 9781450369459. https://doi.org/10.1145/3355088.3365143. https://doi.org/10.1145/3355088.3365143.

Xu, Yang, Yuanfa Jiang, Junbo Zhang, Kang Li ja Guohua Geng. 2022. “Real-Time Ray-Traced Soft Shadows of Environmental Lighting by Conical Ray Culling”. *Proc. ACM Comput. Graph. Interact. Tech.* (New York, NY, USA) 5, numero 1 (toukokuu). https://doi.org/10.1145/3522617. https://doi.org/10.1145/3522617.

Yang, Lei, Shiqiu Liu ja Marco Salvi. 2020. “A Survey of Temporal Antialiasing Techniques”. *Computer Graphics Forum* 39 (toukokuu): 607–621. https://doi.org/10.1111/cgf.14018.

Ylianttila, Lasse, ja Kari Jokela. 2009. “Radiometria”. Teoksessa *Ultravioletti- ja lasersäteily,* 19–73. Säteilyturvakeskus. ISBN: 978-951-712-509-3.

Yuksel, Can, ja Cem Yuksel. 2017. “Lighting Grid Hierarchy for Self-Illuminating Explosions”. *ACM Trans. Graph.* (New York, NY, USA) 36, numero 4 (heinäkuu). ISSN: 0730-0301. https://doi.org/10.1145/3072959.3073604. https://doi.org/10.1145/3072959.3073604.

Yuksel, Cem. 2019. “Stochastic Lightcuts”. Teoksessa *High-Performance Graphics (HPG 2019),* 27–32. Strasbourg, France: The Eurographics Association. ISBN: 978-3-03868-092-5. https://doi.org/10.2312/hpg.20191192.