Ville Peltola

RAYTRACING SELAINYMPÄRISTÖSSÄ

Opinnäytetyö
CENTRIA-AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma
Huhtikuu 2014

Centria
AMMATTIKORKEAKOULU

TIIVISTELMÄ OPINNÄYTETYÖSTÄ

Yksikkö Kokkola-Pietarsaari	**Aika** Huhtikuu 2014	**Tekijä/tekijät** Ville Peltola
Koulutusohjelma Tietotekniikan koulutusohjelma		
Työn nimi RAYTRACING SELAINYMPÄRISTÖSSÄ		
Työn ohjaaja Kauko Kolehmainen		**Sivumäärä** 66
Työelämäohjaaja Jani Ylikangas		

Tämän opinnäytetyön tarkoituksena oli selvittää, miten raytracingtekniikka (sekä myös alaluokka raycasting) soveltuisi nykyhetkellä kolmiulotteisen grafiikan tuottamiseen. Kohdealustaksi valittiin internetselaimet ja demonstraatio toteutettiin JavaScript-ohjelmointikielellä.

Opinnäytetyössä perehdyttiin raytracingin toimintaan. Selvitettiin, tekniikan edut ja haitat, mitä mahdollisuuksia tekniikka tarjoaa sekä millainen sen suorituskyky on nykyaikaisilla internetselaimilla.

Tuloksena saatiin muodostettua opettavainen materiaali sekä pienimuotoinen demonstraatio, joista käyvät ilmi tekniikan vahvuudet. Opinnäytetyön avulla Livion Oy sai lisää tietoa erilaisista mahdollisuuksista, joita tekniikka tarjoaa, ja aloituspohjan tulevaa aikaa varten.

Asiasanat
HTML5, JavaScript, raycasting, raytracing

Centria
AMMATTIKORKEAKOULU

ABSTRACT

Unit	**Date**	**Author/s**
Kokkola-Pietarsaari	April 2014	Ville Peltola
Degree programme Information Technology		
Name of thesis RAYTRACING IN A BROWSER ENVIRONMENT		
Instructor Kauko Kolehmainen		**Pages** 66
Supervisor Jani Ylikangas		

The objective of this thesis was to investigate how the raytracing (and also raycasting subclass) technique could be applied to create three dimensional graphics at present time. Internet browsers were chosen as the target platform and the demonstration was implemented using JavaScript.

During the thesis work the operation of the technique was examined, to determine the possibilities the technique could offer and also its performance in modern day internet browsers.

As a result, an educational material was compiled and a small scale demonstration was produced to show the strengths of the technique. This thesis provided Livion Oy, the commissioner of the thesis work, more information about the possibilities that the technique can offer and also a starting point for future use.

Key words
HTML5, JavaScript, raycasting, raytracing

KÄSITTEIDEN MÄÄRITTELY

AJAX	Asynchronous JavaScript and XML asiakaspuolen webteknologia, joka mahdollistaa laajemman sivustojen interaktiivisuuden, päätoimintana auttaa kehittäjiä tekemään WWW-pohjaisia sovelluksia, jotka toimivat samalla tavalla kuin perinteiset sovellukset
CSS	Cascading Style Sheets standardi joka mahdollistaa sivuston ulkonäön muokkauksen ja kertoo selaimelle, minkälaiselta sivuston elementtien pitäisi näyttää
DHTML	Dynamic HyperText Markup Language nimitys teknologioille, joita käytetään yhdessä luomaan dynaamisia, interaktiivisia sekä animoituja internetsivustoja
DOM	Document Object Model ympäristöstä riippumaton tapa kuvata miten sivustolla olevat objektit ovat vuorovaikutuksessa toisensa kanssa
HDR	High Dynamic Range imaging ryhmä metodeja joita käytetään valokuvauksessa tai grafiikassa, jotta saavutettaisiin parempi ero vaaleimman ja tummimman kohdan välillä
HTML	Hypertext Markup Language staattinen kuvauskieli, jonka avulla Internetin sivustot näytetään, se organisoi ja järjestää sisällön ja on pohjana kaikille sivustoille
JavaScript	skriptauskieli, jota käytetään pääasiassa internetsivustoilla, tehostaa sivustoja ja lisää käyttäjän vuorovaikutusmahdollisuuksia niiden kanssa

OpenGL ES	Open Graphics Library for Embedded Systems selainten käyttämä versio OpenGL-grafiikkakiihdytyksen ohjelmointirajapinnasta (API, Application Programming Interface), käytetään kolmiulotteisen grafiikan luomiseen WebGL:n avulla
Polygon	kaksiulotteinen muoto, jolla on useampi sivu, kuitenkin vähintään kolmisivuinen, ja joka yhdistetään toisiin polygoneihin kärkipisteiden (vertices) avulla, suurempi määrä parantaa objektin tarkkuutta, nostaen samalla tehon tarvetta renderöinnissä.
Renderöinti	kolmiulotteinen grafiikka, jonka kamera (katsoja) näkee, kuten 3D-mallit, skene ja valaistus, käännetään kaksiulotteiseksi kuvaksi, joka mahdollistaa sen piirtämisen ruudulle
Skene	rakennus, maasto, kaupunki tai mikä tahansa paikka, joka esitetään kolmiulotteisena, ja jonka sisältöön kuuluu objektit (3D mallit), valaistus, kamerat ja kaikki ympäröivä rekvisiitta, siis kaikki tiedot, jotka renderöintiin tarvitaan
Sprite	kaksiulotteinen bittikarttakuva, staattinen tai animoitu, esimerkiksi kolmiulotteisen ympäristön litteät objektit tai useamman kuvan, sisältävä kuvatiedosto, kuten animaatioista, yleisesti käytetty grafiikassa ennen aitojen kolmeulotteisten objektien yleistymistä.
WWW	World Wide Web maailmanlaajuinen verkko, johon on liitetty kaikki julkiset sivustot, jotka on yhdistetty Internetiin, sisältää kaikki HTTP:tä (Hypertext Transfer Protocol) käyttävät resurssit ja käyttäjät

KUVIOT

TAULUKOT

1 JOHDANTO

JavaScriptin suoritusnopeus internetselaimissa sekä tietokoneiden (kuten myös mobiililaitteiden) nopeus on vuosi vuodelta kasvanut. Vielä tällä hetkellä keskeneräinen jatkuvasti elävä HTML5- standardi on tuonut eri käyttöjärjestelmät lähemmäksi toisiansa tarjoamalla yhteisen pohjan sisällön tuottamiselle, natiivisti toteutettujen sovellusten rinnalle. Standardin avulla on mahdollista käyttää samaa koodia ja sisältöä riippumatta siitä, mikä käyttöjärjestelmä on asennettuna, ilman sovelluksen tekemistä pahimmassa tapauksessa eri ohjelmointikielellä eri alustalle. Standardi ei kuitenkaan nykyhetkellä ole vielä täydellinen, ja eri alustat voivat vaatia omia pieniä virityksiä ja muutoksia joihinkin koodin osiin tarvittavan toimivuuden saavuttamiseksi. Kuitenkin aikaa myöten kyseisten muutosten tai lisäysten toteuttamisen tarve vähenee. Standardin kehittäjällä W3C:llä on toivomuksena on nykyisen haavoittuvaisen Flash-liitännäisen syrjäyttäminen.

Työn tarkoituksena oli selvittää, miten hyvin nykyiset internetselaimet toimivat kolmiulotteisen grafiikan luomisessa. Opinnäytetyön tein Livion Oy:lle, joka oli kiinnostunut erityisesti aiheesta ja niistä mahdollisuuksista, joita kolmeulotteisen grafiikan avulla olisi mahdollista saavuttaa. Työn aiheeksi rajattiin yrityksen suuren kiinnostuksen vuoksi raytracing ja sen alaluokaksi luokiteltu raycasting. Työn ohjaaja oli hyvin kiinnostunut raytracingistä, ja varsinkin nykyisten prosessorien nopeutuessa yleinen kiinnostus tekniikkaan on kasvanut vuosien aikana. Tekniikan avulla on mahdollista muodostaa realistisen näköisiä kolmiulotteisia ympäristöjä. Raytracingin lisäksi ympäristön luomiseen on mahdollista käyttää esimerkiksi WebGL-tekniikkaa, joka perustuu nykyään yleisessä käytössä olevaan OpenGL ES 2.0 -versioon.

Opinnäytetyön toisessa luvussa käydään läpi Internetin kehityshistoriaa eli sitä, miten alkuajoilta on päästy nykyhetkeen, ja eri tekniikat, joita on vuosien aikana kehitetty. Mitä tarkoittaa tietokonegrafiikka ja millä eri tavoilla sitä käytetään. Luvussa verrataan myös raytracingiä perinteisempään rasterointitekniikkaan. Kolmannessa luvussa esitetään raycasting kaksi- ja kolmiulotteisena sekä raytracingin toiminta. Tavoitteena on esittää ymmärrettävässä muodossa, miten tekniikka toimii ja miten sen voisi toteuttaa. Neljännessä luvussa esitellään raytracingillä tehty demonstraatio ja sen eteneminen vaiheittain sekä myös kaikki käytetyt muut tekniikat, joita käytin apuna toteutuksessa.

2 JOHDATUS TUTKIMUSAIHEESEEN

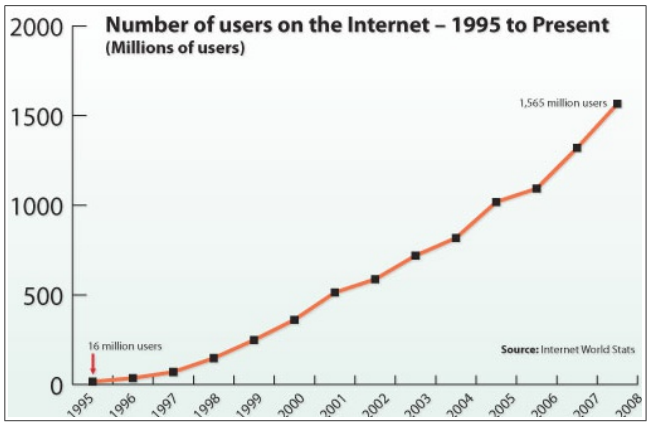
2.1 WWW:n historia

2.1.1 WWW sovellusalustana

Internet syntyi 1960-luvun lopulla nimellä Arpanet. Se oli täysin Yhdysvaltojen armeijan rahoittama projekti, jonka tarkoituksena oli yhdistää eri puolella maata sijaitsevat maan suurimmat yliopistot. Vuonna 1984 Arpanetiin kuului tuhat eri käyttäjää, ja sen nimi päätettiin vaihtaa nykyiseen kaikille tuttuun muotoon Internet. 1990-luvun alussa Internet aloitti laajentumisen ympäri maailmaa, ja tämän seurauksena syntyi WWW. (Marsan 2009.)

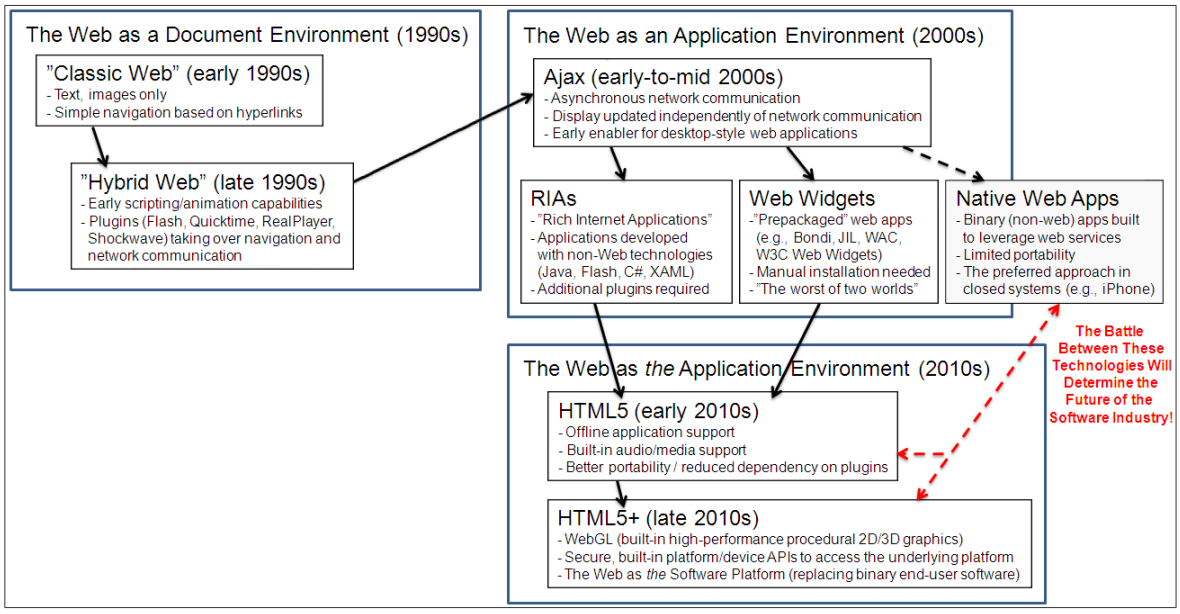
Internet ja WWW ovat niin integroituneet nykyiseen yhteiskuntaan, että on helppo unohtaa, miten kukaan ei 25 vuotta sitten tiennyt sanaa Internet tai ollut edes kuullutkaan siitä. Alkuperäiset suunnitteludokumentit sijoittuvat 1980-luvun loppuun, ja Tim Berners-Lee rakensi ensimmäisen prototyypin joulukuussa 1990. Ensimmäinen suosioon noussut internetselain Mosaic julkaistiin julkiseen käyttöön helmikuussa 1993 ja ensimmäinen kaupallinen selain Netscape Navigator vuoden 1994 lopussa. WWW:n käyttö alkoi yleistyä vasta vuosikymmenen lopulla. (Taivalsaari & Mikkonen 2011, 170.)

Internetin käyttö aloitti räjähdysmäisen laajenemisen vuonna 1995. Tämän seurauksena vuosittain uusien käyttäjien määrä jatkoi nopeaa kasvua. Vuonna 1995 käyttäjiä oli arviolta 16 miljoonaa ja vuoteen 2008 mennessä 1,5 miljardia. Nopean suosion kasvun seurauksena myös laittoman toiminnan määrä alkoi nopeasti nousta. Tämä näkyi erilaisten huijausten ja roskapostiksi luokiteltujen sähköpostiviestien nopeana kasvuna. Arvioiden mukaan 96 prosenttia vuonna 2008 lähetetyistä sähköpostiviesteistä oli roskapostia, päivittäin lähetettyjen roskapostiviestien määrä on arviolta 164 miljardia. (Marsan 2009.) Kuviossa 1 on esitetty Internetin käyttäjien määrän nopea kasvu vuodesta 1995 aina vuoteen 2008 asti.



KUVIO 1. Internetin käyttäjien kasvu (Marsan 2009.)

Vuosien kuluessa WWW on kehittynyt alkuaikojen yksinkertaisesta tiedostojen jaosta nykyiseksi massiiviseksi yleiskäyttöiseksi sisällönjakoympäristöksi. Kehitys ei kuitenkaan ole tapahtunut yhdellä hypyllä, sen sijaan WWW on käynyt läpi useita eri kehitysaskeleita vuosien aikana, minkä lopputuloksena on nykyinen muoto. Kehitysaskeleet voidaan jakaa kolmeen osaan: asiakirjaympäristönä (The Web as a Document Environment), WWW-sovellusten alustana (The Web as an Application Environment) ja WWW-sovellusalustana (The Web as the Application Environment). (Taivalsaari & Mikkonen 2011, 170.) WWW:n kehitys sekä sen askeleet on esitetty kuviossa 2.



KUVIO 2. Internetin kehitys (Taivalsaari & Mikkonen 2011, 171.)

2.1.2 WWW dokumenttiympäristönä

Julkiseen käyttöön Internet alkoi ilmaantua 1990-luvun alussa, ja sivustot olivat oikeasti kirjan sivun tyylisiä sisältäen pääasiassa tekstiä, jonka välissä saattoi olla kuvia. Kukaan ei ollut nähnyt tai kuullutkaan minkäänlaisista animaatioista tai edes vuorovaikutteisista käyttöliittymistä. Navigointi tapahtui yksinkertaisilla linkeillä joita sivustoilta löytyi, myöskään selattavaa sivua ei välimuistitettu mitenkään, joten avattaessa sivua se ladattiin joka kerta kokonaisuudessaan uudestaan palvelimelta. (Taivalsaari & Mikkonen 2011, 171.)

Vuosikymmenen puolessa välissä esiteltiin DHTML, joka sisälsi HTML:n, CSS:n, JavaScriptin sekä DOM-elementit. Tämä mahdollisti sen, että voitiin luoda interaktiivisia sivustoja, jotka saattoivat sisältää grafiikkaa ja animaatiota. Vuosikymmenen loppuun mennessä sivustoista tuli interaktiivisia, ja ne sisälsivät animoitua grafiikkaa. Lisäksi tulivat mukaan erilaiset liitännäiset joiden avulla toimintoja laajennettiin. Liitännäisiin kuuluivat mm. Flash, RealPlayer, QuickTime ja Shockwave. Kaupallinen käyttö lisääntyi yritysten huomattua miten Internetiä voi käyttää mainostamiseen, palveluihin ja myyntiin, ja näin sivustojen ja internetyritysten määrä kasvoi erittäin voimakkaasti. (Taivalsaari & Mikkonen 2011, 171.)

2.1.3 WWW sovellusten alustana

Vuosituhannen vaihteen jälkeen kehitettiin AJAX, joka mahdollisti asynkronisen kommunikoinnin palvelimen ja asiakkaan välille. Sivustoa ei tarvinnut päivittää alusta asti uudestaan, vaan jos yksi kohta sivulla muuttui, AJAX mahdollisti vain muuttuvan kohdan päivittämisen muun sivun pysyessä ennallaan. Se aloitti myös yleistasoisen kiinnostuksen selaimen kautta ajettaviin sovelluksiin. Sovellusten laajentuessa ajan myötä tulivat AJAXin ja selainten rajat nopeasti vastaan. Selainten graafisia ominaisuuksia ei ollut suunniteltu täysin interaktiiviselle graafisia objekteja käsittelevälle sovellukselle. Saatavilla ei ollut sovelluksen toimivuutta eri selaimissa parantavia kehitysalustoja. (Taivalsaari & Mikkonen 2011, 171–172.)

AJAXin rajoitusten ilmaantumisen jälkeen aloitettiin selainriippumattomien sovellusten toimivuuden parantamiseksi kehittämään kehitysalustoja. Näihin kuuluivat esimerkiksi

Adobe AIR, Microsoft Silverlight ja Sun (nykyään Oracle) JavaFX. Kehitysalustat mahdollistivat perinteisten työpöytäsovellusten ohjelmointityökalujen käytön selaimessa sitä tukevan liitännäisen avulla, ja näitä kutsuttiin nimellä Rich Internet Applications (RIAs). RIA-teknologioita ei kuitenkaan luokitella webteknologioiksi, koska ne piilottavat selaimen omat sisäänrakennetut mahdollisuudet korvaten ne omilla vastaavilla. (Taivalsaari & Mikkonen 2011, 172.)

2000-luvun puolivälissä WWW:n mobiilipuoli alkoi kasvaa, mutta pahimpina rajoituksina olivat pienemmät näyttökoot sekä pöytäkoneita huomattavasti hitaampi suorituskyky. Näiden rajoitusten pohjalta kehitettiin Web Widget, esipakattu websovellus, jonka voi asentaa ja ajaa mobiililaitteella. Ne eivät olleet kuitenkaan täysin aitoja websovelluksia latauksen ja asennuksen vuoksi. Web Widgetit onnistuivat kuitenkin nostamaan kehittäjien mielenkiinnon mobiilisovelluksia kohtaan. (Taivalsaari & Mikkonen 2011, 172.)

Web Widgeteiden seuraajaksi kehitettiin Native Web Client Application eli nykyiset natiivit mobiilisovellukset, tutummalta nimeltä Apps. Ne toimivat vain siinä laitteessa, jolle ne on suunniteltu. Ideana natiivisissa sovelluksissa oli vähentää verkon kaistan käyttöä. Sovellus asennetaan laitteelle, ja sen jälkeen sovelluksen mukaan se ei käytä kaistaa lainkaan. Nämä sovellukset ovat tällä hetkellä HTML5-standardisten sovellusten pääkilpailija. (Taivalsaari & Mikkonen 2011, 172.)

2.1.4 WWW sovellusalustana

2010-luvun vaihteessa alettiin kehittää HTML5-standardia, joka tarjoaa monia uusia ominaisuuksia vanhempaan HTML4-standardiin verrattuna. Uusina ominaisuuksina on esimerkiksi applikaatioiden ajaminen ilman Internet-yhteyttä Canvas, joka mahdollisti 2D- ja 3D-grafiikan piirtämisen, asynkroninen skriptien lataus, sivustojen välinen kommunikointi, ja sivuston editointi käyttäjän toimesta. (Taivalsaari & Mikkonen 2011, 173.)

Tämän vuosikymmenen aikana selviää, kumpi nykyisistä webteknologioista, natiivi vai HTML5, ylittää lopullisesti toisensa ja jatkaa tulevaisuuteen. Natiivien sovellusten vaikeutena on niiden kohdistaminen pelkästään tietylle käyttöjärjestelmälle. Sovelluksen toimivuus muissa vaatii sovelluksen uudelleen ohjelmoimisen eli kääntämisen, joka voi vaatia

eri ohjelmointikielen käyttämisen. Perusteellisia syitä siihen miksi puhtaat websovellukset eivät voisi tarjota samoja mahdollisuuksia kuin natiivit tarjoavat, ei ole. Puhtaita websovelluksia käytetään internetselaimen avulla, ja tämän ansiosta niitä voi käyttää kaikilla käyttöjärjestelmillä. Tämä välttäisi sovelluksen kääntämiseltä käyttöjärjestelmältä toiselle. (Taivalsaari & Mikkonen 2011, 172 – 173.)

2.2 Tietokonegrafiikka

2.2.1 Mikä on tietokonegrafiikka

Termi tietokonegrafiikka kuvaa kuvien luontia ja manipulointia tietokoneen avulla (Shirley & Marschner 2009, 1). Tietokonegrafiikkaa säilytetään ja esitetään täysin digitaalisessa muodossa. Grafiikan analogisena muotona voidaan pitää paperia, jonka jälkeenpäin muuttamisen vaikeus riippuu siitä minkälaisia materiaaleja on käytetty. Kuvan voi piirtää ruudulle kahdella eri tavalla: rasteri- tai vektorigrafiikkana. (Woodford 2013.)

Tietokonegrafiikka esitetään käyttäjälle yleensä rasterityyppiseltä näytöltä. Rasterinäyttö esittää kuvat kaksiulotteiseen taulukoon sijoitettujen pikseleiden avulla. Yleisimmät esimerkit ovat tietokoneen näytöt tai televisiot, joiden kuva muodostuu yksittäisistä värillisistä pisteistä eli pikseleistä. Suurin osa tulostimista on rasteritoimisia, ja myös kamerat ja skannerit muodostavat kuvansa rasteroinnilla. Koska kaikki laitteet osaavat rasteroinnin, ovat rasterikuvat tämän vuoksi yleisin tapa esittää grafiikkaa. Rasterikuva on yksinkertainen kaksiulotteinen taulukko jonka jokaisessa alkiossa on väriarvo, joka yleensä sisältää kolme väriarvoa, näistä jokaiseen sijoitetaan jokainen pääväri eli punainen, vihreä ja sininen. Koska kaikissa näyttötyypeissä on yleensä vähemmän tai enemmän pikseleitä kuin mitä esitettävä kuva sisältää, rasterikuvaa ei voida esittää suoraan näytössä. Tähän väliin tarvitaan tulkki, joka muuntaa ja skaalaa lähdekuvan näytölle sopivaksi. (Shirley & Marschner 2009, 53.)

Vektorigrafiikka on toinen tapa esittää grafiikkaa. Esitettävää kuvaa ei säilytetä rasterikuvan tyylisessä kaksiulotteisessa taulukossa, sen sijaan vektorikuvat sisältävät ohjeita ja kuvauksia esitettävistä muodoista ilman mitään viittausta siihen mille kuvan pikselille ne

kuuluvat. Etuina vektorikuvalla on niiden resoluutioriippumattomuus, jonka seurauksena niitä voi esittää helposti eri pikselimääriä sisältävillä näytöillä. Suurimpana haittana kaikkien näyttöjen ollessa rasteripohjaisia on, se että ne eivät osaa näyttää vektorikuvaa ja niiden välissä pitää olla tulkki, joka muuntaa vektorikuvan rasterikuvaksi. Vektorigrafiikkaa käytetään yleensä teksteissä, kaavioissa, mekaanisissa piirustuksissa ja muissa kohteissa, joissa tarkkuus ja terävyys ovat tärkeät ominaisuudet. (Shirley & Marschner 2009, 54) Vektorigrafiikkaan ei tämän syvemmin mennä tässä opinnäytetyössä, koska grafiikka kuitenkin esitetään rasteroituna.

Tietokonegrafiikka muodostuu pienistä värillisistä pisteistä, joita kutsutaan pikseleiksi. Rasteroidun kuvan voi esimerkiksi esittää tiiliseinänä, jonka jokainen tiili on oman värinen. Näytön suurinta mahdollista pikselimäärää kutsutaan resoluutioksi. Jay Forrester ja Robert Everett kehittivät ensimmäisen tietokonegrafiikkaan kykenevän tietokoneen vuonna 1951. Tietokone pystyi piirtämään yksinkertaisia kuvia monitorille, joka perustui samaan tekniikkaan kuin sen ajan televisiot. (Woodford 2013.) Kuviossa 3 on näkyvissä rasterikuva jonka yksi kohta on suurennettu pikseleiden näkyvyyden parantamiseksi.

Tietokonegrafiikka esitetään binäärisessä muodossa, ja jokaisella pikselillä on oma tietty määrä bittejä eli ykkösiä tai nollia. Bittien määrä riippuu siitä, kuinka paljon on värejä käytettävissä. Yksinkertaisimmillaan bittejä on yksi, jonka avulla kuvan jokainen pikseli voi olla joko mustana tai valkoisena. Nykyhetkellä tavanomainen bittimäärä on jokaista pikseliä kohden 32 bittiä, joista kahdeksan bittiä kuuluu alpha- eli läpinäkyvyydelle, joka mahdollistaa 16 777 216 erilaista väriä. Suurempi bittimäärä käyttää myös suuremman määrän muistia. Seuraavalla kaavalla on mahdollista laskea, miten paljon kuva käyttäisi muistia. (Woodford 2013.)

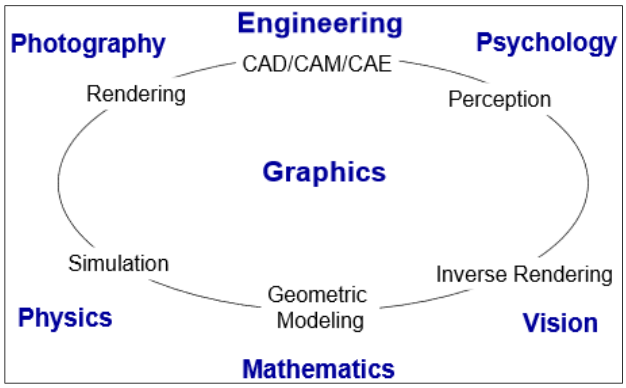
$$\text{Käytetty muisti} = \text{Resoluutio}_{\text{Leveys}} \times \text{Resoluutio}_{\text{Korkeus}} \times \text{Bitit per pikseli}$$

KUVIO 3. Rasterikuvan pikselit (Fleet & Hertzmann 2006, 1.)

Grafiikan ammattilaisista suurin osa asettaa tietokonegrafiikan pääalueiksi mallinnuksen, renderöinnin ja animaation. Muita alueita, joilla tietokonegrafiikkaa käytetään, on lukematon määrä, kuten esimerkiksi visuaalinen vuorovaikutus, kuten graafiset käyttöliittymät, virtuaalitodellisuus, visualisointi, kuvan manipulointi, 3D-skannaus ja valokuvaus tietokonetta käyttäen. (Shirley & Marschner 2009, 2.)

2.2.2 Mihin tietokonegrafiikkaa käytetään

Tietokonegrafiikalle on paljon erilaisia käyttökohteita. Selvimpiä ovat esimerkiksi taide, elokuvat (käytetään termiä CGI, Computer-generated Imagery) ja arkkitehtuuripiirustukset. Käyttökohteita on myös paljon sellaisia, jotka eivät ole niin selviä tai taiteellisia, ja näihin kuuluu esimerkiksi tieteellisen tai lääketieteellisen tiedon esittäminen visualisesti. Syy siihen miksi tietokonegrafiikkaa käytetään nykyään paljon ja sen käyttö laajenee jatkuvasti, on sen mahdollistama yksinkertainen tapa muuttaa monimutkainen ja vaikeasti ymmärrettävä tieto tai materiaali helposti visuaalisesti ymmärrettävään muotoon. Katsojan on vaikea ymmärtää pelkästä raakatiedosta tai numeroista, mitä tiedon tai materiaalin pitäisi tarkoittaa, jos ei ole alan asiantuntija. (Woodford 2013.) Kuviossa 4 on näkyvissä minkälaisiin osa-alueisiin tietokonegrafiikka olisi mahdollista jakaa.



KUVIO 4. Tietokonegrafiikan osa-alueet (Magnor 2006, 6.)

Melkein missä tahansa hankkeessa on mahdollista käyttää tietokonegrafiikkaa. Seuraavilla toimialoilla tietokonegrafiikan käyttö on suurta ja kasvaa jatkuvasti:

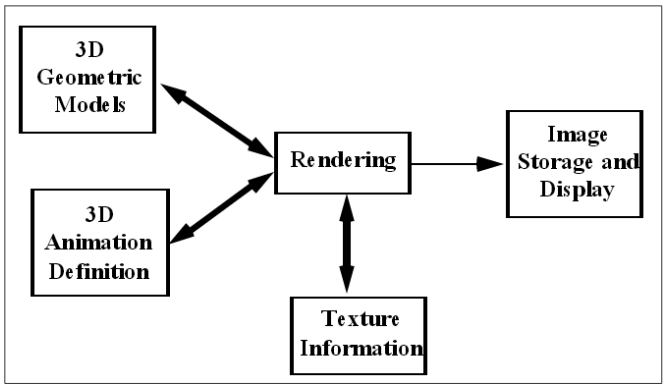
- Videopelit käyttävät kasvavassa määrin pitkälle kehitettyjä 3D-malleja ja renderöintialgoritmejä.
- Piirretyt renderöidään yleensä suoraan 3D-malleista. Monet kaksiulotteiset piirretyt käyttävät 3D-mallista renderöityä taustaa, joka sallii sulavasti liikkuvan kuvakulman säästäen graafikkojen aikaa.
- Visuaaliset efektit, digitaalinen sommittelu kuvatun materiaalin asettamisessa tietokoneella luodun materiaalin päälle ovat käytössä melkein kaikissa nykyisissä elokuvissa. Monesti elokuvassa käytetään myös 3D-mallinnusta ja animaatiota keinotekoisten ympäristöjen, objektien tai hahmojen luomiseen.
- Animoiduissa elokuvissa, käytetään kaikkia samoja tekniikoita kuin edellisissä visuaalisissa efekteissä käytetään mutta ilman samanlaista realistisuuden tavoittamista ympäristöissä ja muissa objekteissa.
- Computer-aided design (CAD)- ja Computer-aided manufacturing (CAM)- suunnittelussa käytetään tietokonegrafiikkaa tuotteiden ja erilaisten osien suunnittelussa, jonka jälkeen suunnitellusta objektista on mahdollista luoda virtuaalinen 3D-malli, jota voidaan käyttää valmistuksessa. Esimerkiksi mekaanisesta osasta voidaan suunnitella virtuaalinen 3D-malli, jonka jälkeen se voidaan valmistaa fyysisesti.
- Simulaatio voidaan kuvitella realistisempana videopelinä. Esimerkiksi lentosimulaattorit käyttävät pitkälle kehitettyjä 3D-malleja ja algoritmeja aidon lentokoneen lentämisen mallintamiseen. Simuloinnit voivat olla todella hyödyllisiä esimerkiksi erilaisten turvallisuutta vaarantavien tilanteiden, kuten ajamisen harjoittelussa.

Myös kokeneemmat henkilöt, kuten esimerkiksi palomiehet, voivat harjoitella tilanteita, jotka voivat olla liian vaarallisia tai kalliita toteuttaa fyysisesti.

- Lääketieteellisessä visualisoinnissa, voidaan potilastiedon pohjalta luoda kuvia, joiden tulkitseminen on helpompaa henkilölle, jolla ei ole lääketieteen tutkimusta. Esimerkiksi tietokonetomografian (computed tomograpdy, CT) materiaali esitetään isoon kolmiulotteiseen taulukkoon sijoitettuina tiheysarvoina. Tietokonegrafiikan avulla on mahdollista luoda 3D-malli tiedosta, joka helpottaa lääkäreiden työtä erottaa kaikkein keskeisin tieto materiaalista.
- Tietojen visualisoinnissa, voidaan luoda kuvia materiaalista, jota ei pysty visuaalisesti esittämään. Esimerkiksi tuotteen hinnan kehitystä pitemmällä aikavälillä ei voi suoraan esittää visuaalisesti, mutta on mahdollista esittää se erilaisten kaavioiden avulla, jotka helpottavat ihmisiä näkemään selkeän kuvion. (Shirley & Marschner 2009, 3.)

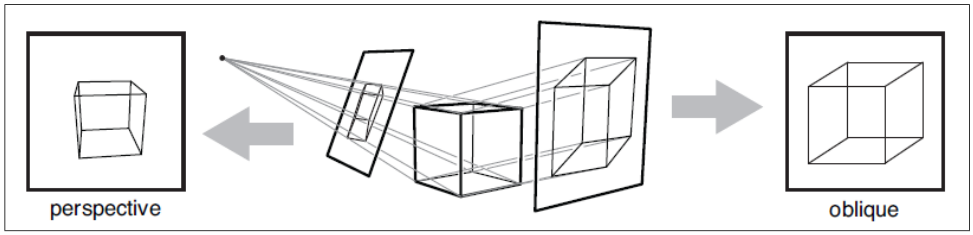
2.2.3 Kolmiulotteinen grafiikka

Yksi tietokonegrafiikan perustoiminnoista on renderöidä kolmiulotteiset objektit: ottamalla skenen tai mallin, ja tämän pohjalta luodaan kaksiulotteinen kuva. Renderöinti on prosessi, joka tuottaa kaksiulotteisen pikseleitä sisältävän taulukon sille annettujen objektien pohjalta. Kuviossa 5 on esitetty yhteenveto siitä, miten kolmiulotteisesta grafiikasta muodostetaan kaksiulotteinen kuva. Toimenpiteeseen vaaditaan vähintään kolmiulotteiset mallit, animaatiot ja tekstuurit eivät ole pakollisia, mutta parantavat realistisuutta huomattavasti. (Shirley & Marschner 2009, 69.)



KUVIO 5. Yhteenveto grafiikan tuottamisesta (Bailey, Glassner & Lathrop 1999, 9.)

Kolmiulotteisen objektin tai skenen esittäminen kaksiulotteisena kuvana on ongelmallista, ja taiteilijat ovatkin vuosisatojen aikoja tutkineet asiaa. Valokuvat esittävät myös kolmiulotteisen skenen kaksiulotteisen kuvan avulla. Perinteinen ja eniten käytetty lähestymistapa taiteen, valokuvan tai tietokonegrafiikan esittämisessä on lineaarinen perspektiivi (linear perspective). Tässä tavassa kolmeulotteiset objektit heijastetaan kuvatasolle (image plane), niin että skenen suorat viivat ovat myös kuvatasolla suoria. Yksinkertaisin heijastustapa on rinnakkaisheijastus (parallel projection), jossa kolmiulotteiset pisteet muunnetaan kaksiulotteisiksi liikuttamalla niitä heijastuksen suuntaan niin kauan, kunnes ne törmäävät kuvatasoon. Jos kuvataso on kohtisuorassa kameran suuntaan verrattuna, heijastusta kutsutaan ortografiseksi, muussa tapauksessa heijastus on epäsuora. (Shirley & Marschner 2009, 71–72.) Kuviossa 6 on esimerkki vasemmalla perspektiivisestä heijastuksesta ja oikealla puolella epäsuorasta heijastuksesta.



KUVIO 6. Perspektiivinen ja epäsuora heijastus (Shirley & Marschner 2009, 72.)

2.3 Raytracing

Opinnäytetyön aiheeksi valittiin raytracing sen viime vuosien aikana saavuttaman suosion kasvun vuoksi. Työelämänohjaaja Jani Ylikangas on itse myös suuresti kiinnostunut raytracingista ja hänellä on selvät syyt siihen, miksi tekniikka voi jossain vaiheessa ylittää perinteisemmän rasteroinnin. Raytracing on huomattavasti raskaampi tällä hetkellä verrattuna kilpaileviin perinteisempiin rasterointitekniikoihin kuten WebGL:ään. Prosessorivalmistaja Intel on yksi suuri raytracingin kannattaja, ja yhtiö onkin perustanut oman yksikön raytracingin kehittämiselle (Abi-Chahla 2009a). Kuviossa 7 on esitetty, miten Intel on saanut kehitettyä raytracingin ja prosessorien nopeutta vuosien aikana.

KUVIO 7. Intelin raytracingin-kehitys (Abi-Chahla 2009a.)

2.3.1 Raytracingin edut

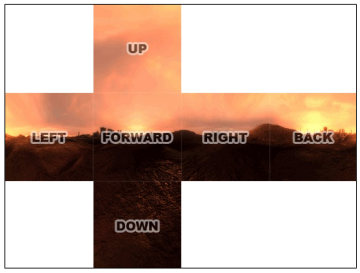
Raytracingillä on useita etuja verrattuna perinteisempiin rasterointitapoihin. Seuraavaksi esitellyt ominaisuudet ovat yksinkertaisempia ja helpompia toteuttaa raytracingillä paremman tarkkuuden saavuttamiseksi.

2.3.1.1 Heijastukset

Raytracingillä on useita etuja perinteisempiin rasterointitekniikoihin verrattuna. Algoritmi käyttää pelkästään yhtä alkukantaista objektia niiden efektien luomiseen, joita rasteroinnissa simuloidaan standardoitujen algoritmien avulla, jotka vaativat ohjelmoijalta kovaa työtä ja paljon nerokkuutta. Heijastukset ovat yksi asia, jossa raytracing loistaa. Nykyisillä kolmiulotteisilla grafiikkamoottoreilla, joita käytetään esimerkiksi videopeleissä, heijastukset lasketaan ympäristökartan (environment map) avulla. Tämä tekniikka antaa hyvän likiarvon siitä, minkälainen heijastus voisi olla, mutta lähellä olevissa objekteissa tekniikan rajoitukset tulevat esille. Kuviossa 8 on esitetty animaatiostudio Pixarin Autot-elokuvan kohtaus, jossa heijastukset on tehty rasteroinnin lisäksi raytracingillä. Raytracing-versiossa auton konepellillä on silmien heijastus, ja saman heijastuksen esittäminen rasteroinnilla ja ympäristökartan avulla ei ole mahdollista. (Abi-Chahla 2009a.)

KUVIO 8. Raytracingin etu heijastuksessa (Abi-Chahla 2009a.)

Ohjelmoijat ovat kehittäneet erilaisia kikkoja, ja mahdollistaneet realistisemmat heijastukset rasteroinnin avulla, käyttämällä dynaamisia kuutiokarttoja (cube map). Kamera voidaan asettaa esimerkiksi kilpa-autovideopelissä pelaajan ohjaaman ajoneuvon kanssa samalle tasolle ja renderöidä kohdasta kuva kuudesta eri suunnasta: vasemmalta, oikealta, edestä, takaa, yläpuolelta ja alapuolelta. Jokaisesta suunnasta renderöidyt kuvat tallennetaan kuutiokarttaan, ja niiden pohjalta lasketaan heijastukset. (Abi-Chahla 2009a.) Kuviossa 9 on esitetty esimerkki kuutiokartasta miten eri suunnista renderöidään kuvat. Kuviossa 10 on näkymä, jossa kuutiokartan avulla on luotu heijastukset. Kuviossa näkyvät rasteroinnin rajoitukset ja puutteet heijastuksissa, esimerkiksi teepannun nokka ei heijastu teepannun pinnalle ollenkaan.



KUVIO 9. Esimerkki kuutiokartasta (Habib's Water Shader 2009.)

KUVIO 10. Kuutiokartan avulla luodut heijastukset rasteroinnissa (Abi-Chahla 2009a.)

Dynaamisten kuutiokarttojen luominen on kuitenkin raskasta, ja tämän vuoksi kuutiokarttojen luomiselle on kolme rajoitusta. Kuutiokartat luodaan harvemmin kuin pääkuva, esimerkiksi tietyn aikavälein, sen sijaan että niitä luotaisiin reaaliaikaisesti. Tämä aiheuttaa viivettä heijastuksen liikkeeseen. Toisena rajoituksena kuutiokartan luomisen keventämiseksi ja ruudunpäivitysnopeuden kasvattamiseksi kuvat renderöidään pienemmällä resoluutiolla, mutta tämä aiheuttaa pikselöitymistä heijastuksiin. Kolmantena rajoituksena heijastukset rajoitetaan yleensä objekteihin, joihin katsoja kiinnittäisi eniten huomiota sillä hetkellä, ja tämän seurauksena joistakin objekteista voi mahdollisesti puuttua heijastukset kokonaan. (Abi-Chahla 2009a.)

Raytracingillä heijastukset ovat täysin hallittavissa ilman monimutkaisten algoritmien käyttöä. Renderöintialgoritmi hoitaa erillisten apualgoritmien sijaan kaikkien mitä heijastukseen tarvitsee. Suurena etuna rasterointiin verrattuna on erilaisten heijastusten toteutus, kuten esimerkiksi ajoneuvon tausta- tai sivupeilien heijastukset, jotka olisi äärimmäisen vaikeaa tai mahdotonta toteuttaa rasteroinnilla, ovat helposti toteutettavissa samalla tavalla kuin minkä tahansa muunkin heijastuksen. Heijastusten heijastukset, kuten peilistä toiseen peiliin, onnistuvat myös yhtä helposti. (Abi-Chahla 2009a.)

2.3.1.2 Läpinäkyvyys

Oikeanlaisen läpinäkyvyyden hallinta rasterointialgoritmien avulla on äärimmäisen vaikeaa läpinäkyvyyden laskennan ollessa täysin riippuvainen siitä, missä järjestyksessä objektien renderöinti suoritetaan. Ennen läpinäkyvyyden laskemista polygonit pitää järjestää kauimpana olevasta lähimpänä olevaan. Järjestäminen ei ole kuitenkaan helppoa, se on myös laskennallisesti raskasta, ja lopputuloksessa voi olla virheitä, koska järjestys pitää tehdään polygoneille eikä pikseleille. Raytracingillä läpinäkyvyyden laskeminen on helpompaa, koska polygonien sijasta työskennellään pikseleiden kanssa. (Abi-Chahla 2009a.)

2.3.1.3 Varjostus

Rasteroinnilla varjostuksen luomiseen käytetään standardoitua tekniikkaa nimeltä varjokartoitus (shadow mapping). Tekniikka kuitenkin kärsii monesta ongelmasta, muodostetut var-

jot ovat sahalaitaisia, ja varjokartan luominen vaatii paljon muistikapasiteettia. Raytracingllä varjojen laskemiseen ei tarvitse monimutkaisia algoritmeja, ja niiden laskemiseen käytetään samoja alkukantaisia objekteja, ilman että käytetään yhtään enempää muistia. (Abi-Chahla 2009a.)

2.3.1.4 Kaarevat pinnat

Raytracing osaa natiivisti hallita kaarevia pintoja. Rasteroinnin puolella vuosien aikana nykyisiin grafiikkakortteihin on lisätty kaareville pinnoille tuki, joka on hävinnyt ja tullut uudelleen esille ajureiden eri versioissa. Rasteroinnissa käytetään tekniikkaa nimeltä tesselaatio, jonka avulla voidaan luoda kolmioita, mikä on ainoa alkukantainen objekti, jota perinteinen rasterointi osaa hallita. Raytracingillä säteiden törmäys kaarevien pintojen kanssa on helppo tarkistaa. Tämä johtuu raytracingin kyvystä käyttää hyväkseen kaarevan pinnan matemaattista määrittelyä. (Abi-Chahla 2009a.)

2.3.2 Raytracingin rajoitukset

Pääsyy siihen minkä vuoksi raytracing ei käytetä vaikka sillä on useita suuria etuja rasterointiin verrattuna, on sen hitaus. Säteitä pitää laskea todella suuri määrä kerralla, mistä aiheutuu raskas kuorma tietokoneelle. Nopeus on kuitenkin vuosi vuodelta parantunut useammilla ytimillä varustettujen prosessorien vuoksi, jonka vuoksi kuormaa on mahdollista jakaa eri ytimien kesken. Algoritmien optimointi ja kehitys on myös mennyt eteenpäin jatkuvasti. Suurin nopeuteen vaikuttava tekijä tulee esille säteiden kohdalla. Jokainen säde käsittelee täysin erilaista tietoa kuin mitä edellinen säde käsitteli, ja tämän vuoksi nykyisiä välimuistitekniikoita, jotka ovat olennaisia osia hyvän suorituskyvyn saavuttamiseen, ei voi hyödyntää ollenkaan. Välimuistin ollessa poissa käytöstä suurin vaikuttaja toissijaisten säteiden laskemisnopeuteen onkin keskusmuistin viivenopeus. (Abi-Chahla 2009a.)

Terävien kulmien pehmentäminen (anti-aliasing) on myös ongelmallista raytracingillä. Laskettavat säteet ovat yksinkertaisia matemaattisia abstraktioita, eikä niillä ole kokoa ollenkaan. Jokainen törmäys objektin kanssa palauttaa pelkästään totuusarvon, ja yksityis-

kohtaisempien tiedon puuttuessa, kuten esimerkiksi ”40 % säteestä osuu objektiin”, ei ole saatavilla, tämä aiheuttaa terävät sahalaidat objektien reunoille. (Abi-Chahla 2009a.)

Rasteroinnilla reunojen pehmentäminen on helpompi toteuttaa. Erilaisia tekniikoita on tutkittu ja testattu ongelman ratkaisemiseksi toiveessa, kuten keilamaista tai kartiomaista säteen seurausta, joiden avulla säteelle voisi määritellä koon, mutta tekniikat ovat liian monimutkaisia ottaa käyttöön. Yksi tekniikka, joka on tarjonnut hyvän tuloksen sahalaitojen poistamiseen, on ylinäytteistys (supersampling), jossa lähetetään useampi säde yhden pikselin eri kohtiin. Ylinäytteistys raytracingillä on kuitenkin paljon raskaampaa. (Abi-Chahla 2009a.)

2.4 Videopeliteollisuus

2.4.1 Maailmanlaajuinen kehitys

Viime vuosien aikana videopeliteollisuus on kasvanut nopealla tahdilla. Michael Gallgherin, Enterntainment Software Associationin toimitusjohtajan mukaan mikään muu alue ei ole kokenut samanlaista räjähdysmäistä kasvua kuin videopeliteollisuus. Videopeliteollisuus on maailmanlaajuisesti ruvennut uhkaamaan elokuvateollisuutta. Vuonna 2012 pelaajat käyttivät videopeleihin yhteensä 20,77 biljoonaa dollaria. Suuresta kasvusta todistaa vähän aikaa sitten julkaistu Grand Theft Auto V, jonka kehitys- ja markkinointibudjetti oli yhteensä 256 miljoonaa dollaria, joka on samalla tasolla suurimpien elokuvien budjettien kanssa. Elokuvateollisuudessa esimerkiksi Man of Steel -elokuva tuotti 300 miljoonaa dollaria 14 viikon aikana, ja vastaavasti GTA V -videopeli tuotti biljoona dollaria kolmen päivän aikana. Samaan kokoluokkaan ylettyvää määrää ei ole elokuvateollisuuden puolella nähty. (Keswani 2013.)

Maailmanlaajuisesti videopeliteollisuuden tulot olivat 67 biljoonaa dollaria vuonna 2013. Tulojen on arvioitu kasvavan 82 biljoonaan dollariin vuoteen 2017 mennessä. Samaan aikaan elokuvateollisuuden tulot olivat vuonna 2010 94 biljoonaa dollaria, joka oli 17 prosenttia vähemmän kuin vuonna 2001. Nykyään erilaiset suoratoistopalvelut, kuten Netflix, ovat kasvattaneet suuresti suosiotaan, ja kuluttajien mielestä elokuvateatteriin meneminen

on kalliimpaa kuin saman elokuvan katsominen suoratoistopalvelun kautta. (Keswani 2013.)

2.4.2 Kehitys Suomessa

Videopeliteollisuus on kasvanut kovaa vauhtia maailmalla ja myös Suomessa. 1980-luvulla peliteollisuus sai alkunsa Suomessa, mutta silloin se oli kuitenkin todella pientä. 1990-luvulla suomalaiset peliohjelmoijat järjestäytyivät, jonka seurauksena oli suomalaisten peliyritysten, kuten Remedy Entertainmentin, ilmestyminen. Vuosikymmenen lopussa aloitettiin mobiilipelikokeilut ja tästä syntyi Suomen vahva suuntaus mobiilipuolelle. 2000-luvulla mobiilisuuntaus vahvistui Nokian julkaiseman N-Gage-laitteen myötä, ja vaikka laite ei ollut menestys, se oli suuri vaikuttaja suomalaisessa mobiilipeliteollisuudessa. 2010-luvulla digitaaliset jakelukanavat nostivat suosiotaan, tämä erilaisten mobiililaitteiden, kuten tablettien, avulla vahvisti huomattavasti mobiilipuolta ja synnytti monet suomalaiset tunnetut mobiilipelit, kuten Angry Birds, Hay Day ja Clash of Clanssin. (Hiltunen, Latva & Kaleva 2013, 8–9.)

Mobiilipuolen nousua auttoi huomattavasti digitaalisten jakelukanavien ilmestyminen. Videopelin tekemisestä tuli kehittäjille kannattavampaa: perinteisellä jakelukanavalla vain 10 % tuloista jäi kehittäjälle, kun taas digitaalisella jakelukanavalla kehittäjälle voi jäädä kanavan mukaan jopa 70 % tuloista. Tämä ero johtuu välikäsien, kuten jakelijan ja jälleenmyyjän, pois jäämisestä. Ennen nykyistä aikaa mobiilipuolen pelien pääasiallisena ohjelmointikielenä oli Java, mutta kielen suurin ongelma oli skaalautumattomuus ja mobiililaitteiden standardien puute. Tämän vuoksi pelistä piti aina tehdä niin monta eri versiota kuin oli kohdelaitteita. Ainoana jakelukanavana oli myös pelkästään operaattori, joka yleensä otti suuren osan myyntihinnasta. (Hiltunen ym. 2013, 19–21)

Videopeliteollisuuteen kuului arviolta 40 alan yritystä, jotka työllistivät suomessa keskimärin 600 henkilöä vuonna 2004. Vuoden 2013 loppuun mennessä yritysten määrä oli kasvanut 180 kappaleeseen, ja työntekijöiden määrä arviolta 2200 henkilöön. Alustavien arvioiden mukaan toimiala tulee työllistämään arviolta 3500 henkilöä vuonna 2020. Toimialan kasvu on synnyttänyt kursseja eri oppilaitoksissa. (Hiltunen ym. 2013, 46.)

Toimialan nousua rajoittaa eniten työvoiman puute. Koulutuksen aloituspaikkoja on kuitenkin lisätty, mikä helpottaa tilannetta. Yritysten mielestä on tärkeää, että vastavalmistuneiden ja kokeneiden työntekijöiden määrät olisivat sopivassa suhteessa. Suurena haasteena on myös yritysten alkuvaiheen rahoituksen vaikea saatavuus, joka johtuu siitä, että mahdollisilla sijoittajilla on vähän toimiala tuntemusta ja suomalaisten peliyritysten liiketoimintasuunnitelma ei yleensä houkuttele ulkomaalaisia sijoittajia. (Hiltunen ym. 2013, 47.)

3 RAYCASTING & RAYTRACING

Raycasting- ja raytracing-tekniikoiden inspiraationa ovat toimineet ihmisen silmä sekä erilaiset kamerat. Silmä ja kamera vastaanottavat valoa kolmiulotteisesta ympäristöstä ja keskittävät sen kaksiulotteiselle pinnalle. Kyseinen pinta on kameralla litteä tai silmällä kaareva. Pinnan muodolla ei kuitenkaan ole mitään väliä, koska siihen keskitetty kuva on menettänyt yhden ulottuvuuden verrattuna lähteeseen josta se tuli. Tämän vuoksi esimerkiksi kolmiulotteista palloa ei voi esittää kaksiulotteisena paperilla ilman vääristymiä, ja kameran tai silmän keskittämässä kuvassa on samanlaisia vääristymiä. Vääristymät ovat kuitenkin ihmisille niin jokapäiväisiä, minkä vuoksi aivot osaavat käsitellä vastaanotetun tiedon irrottamalla siitä kaiken hyödyllisen tiedon. (Cross 2013, 9.)

3.1 Historia

Raytracingia on käytetty optisten linssien simulointiin, sekä radioaaltojen ja muiden säteilyn lähteiden etenemisen simulointiin. Arthur Appel esitti vuonna 1968 raytracingin käyttämistä tietokonegrafiikassa. Raycasting ja raytracing ovat teknisesti erilaisia. Tekniikoiden toimintaideat ovat kuitenkin samankaltaisia, minkä vuoksi termejä onkin käytetty ristiin kirjallisuudessa. Nykyään termit erotetaan paremmin toisistaan algoritmeilla, jotka perustuvat raycastingiin tai raytracingiin. (Furth 2010, 530.)

Appelin kehittämät algoritmit kuvasivat, miten säde lähetetään katsojasta skeneen ja miten sen törmäykset geometrian kanssa käsitellään. Jokaista ruudun pikseliä kohtaan lähetetään skeneen oma säde, jonka törmäykset jokaista geometrista elementtiä kohtaan tarkistetaan, ja tunnistetut törmäykset matkan varrella tallennetaan muistiin. Sen jälkeen kun pikselin säteen törmäys on tunnistettu, voidaan se piirtää ruudulle sillä värillä, mikä oli törmätyllä objektilla. Yksinkertaisimmassa implementaatiossa jokaisella objektilla on yksi väri, joka piirrettään ruudun pikselin kohdalle. Tämä riittää kuvan tuottamiseen ruudulle. Syvyyspuskuri on myös tarpeellinen lähimmän törmätyn objektin määrittämiseen. (Furth 2010, 530.)

Appel kehitti algoritmia ottamalla huomioon törmätyn objektin materiaalin. Ympäristön valonlähteiden huomioon ottamisella voidaan objektille luoda varjostus. Appelin raycasting-metodin etuna vanhempiin juovariveihin perustuviin metodeihin on monimutkaisempien ja epäsuorien pintojen paljon tarkempi mallinnus ja renderöinti. Edellä mainituilla vanhemmilla metodeilla epäsuorat pinnat pitää ensin jakaa pienempiin ja yksinkertaisempiin geometrisiin pintoihin, kuten esimerkiksi yleisesti käytettyihin kolmioihin ja renderöidä paloittain likiarvolla. Raycasting-lähestymistavan ansiosta on mahdollista renderöidä, mikä tahansa pinta, jolle on mahdollista suorittaa törmäystarkistus, hyvällä tarkkuudella. (Furth 2010, 530.)

Ensimmäinen elokuva, joka käytti raycastingia yhdessä näyttelijöiden kanssa, oli vuonna 1982 julkaistu Tron. Tron oli historiallinen ja suuri julkaisu. Ensimmäisenä maailmassa elokuva käytti kehittyneitä raycasting-algoritmeja asettaakseen tietokonegrafiikkaa oikeiden näyttelijöiden päälle, ja tästä tuli myöhemmin tietokoneella luodun grafiikan kulmakivi. Samana vuonna ilmestyi Blade Runner, elokuva, joka on nykyäänkin suuri vaikuttaja tulevaisuuteen sijoittuvissa elokuvissa. Elokuvassa ei tosin käytetty tietokonegrafiikkaa ollenkaan vaan sen sijasta pienoismalleja sekä valokuvamaisemia. Nykyään tulevaisuuteen liittyvissä elokuvissa hyödynnetään kehittyneitä raytracing-renderöintialgoritmeja näyttelijöiden, pienoismallien ja perinteisten efektien lisäksi. (Boaz 2008, 43–44.)

Turner Whitted laajensi Appelin ideaa vuonna 1980 lisäämällä algoritmit, joiden avulla voi hallita heijastuksia ja valon taittumista. Näiden laajennuksien kanssa tekniikkaa kutsutaan raytracingiksi. Sen sijaan että laskettaisiin yksinkertaisesti törmäyspisteen varjostus, Whittedin uusi algoritmi generoi enintään kolme uutta sädettä, ja näitä kutsutaan toissijaisiksi säteiksi. Katsojasta lähetetty säde on nimeltään ensisijainen säde. Toissijaisten säteiden tarkemmat nimet, joilla ne erotetaan toisistaan, ovat varjostus-, heijastus- ja taittumissäde. Nämä sallivat tarkan varjojen, heijastusten ja taittumisen simuloinnin objekteissa, ja niitä käsitellään samalla tavalla kuin ensisijaistakin sädettä. Toissijaisten säteiden törmätessä objekteihin skenessä, näidenkin on mahdollista luoda omat toissijaiset säteensä. Suorituskyky kuitenkin kärsii huomattavasti säteiden määrän kasvaessa. (Furth 2010, 531.)

Whittedin laajennuksen jälkeen algoritmeja on laajennettu parantamalla mallinnuksen tarkkuutta lisäämällä erilaisia efektejä: syvyysvaikutelma (depth of field), hajasäteiden väliset

heijastukset, pehmeämmät varjot (soft shadows), liikkeen sumennus (motion blur) ja muita optisia efektejä. Useat kehittäjät ovat parantaneet raytracingin suorituskykyä käyttämällä uusia algoritmeja, joilla indeksoidaan skenen objekteja. (NVIDIA 2012, 3.)

Koska algoritmeja voidaan käsitellä rekursiivisesti, niiden laskennallinen monimutkaisuus kasvaa eksponentiaalisesti silloin, kun skenessä on paljon heijastavia objekteja ja valonlähteitä. Kehittyneempiä tekniikoita on kehitetty useampia erilaisia mahdollistaen esimerkiksi hajaantuvien säteiden simuloinnin. Algoritmien nopeutta on myös parannettu vuosien aikana käyttämällä pienikokoisia tietorakenteita tai tehokkaampia törmäystunnistusfunktioita. Nykyään valtaosa raytracingiä käyttävistä ohjelmista käyttää Whittedin heijastus-, taittumis- ja rekursiometodeja. (Furth 2010, 531.)

3.2 2D-raycasting

Kaksiulotteinen raycasting on renderöintitekniikka, joka mahdollistaa kolmiulotteisen näkymän luomisen kaksiulotteisesta kartasta. 1990-luvulla tietokoneiden ollessa hitaampia kolmiulotteisia grafiikkamoottoreita ei ollut mahdollista ajaa reaaliajassa. Tekniikassa lasketaan jokainen pystyrivi erikseen, esimerkiksi kuvan tarkkuuden ollessa 640 x 480 laskutoimitus tehdään yhtä monta kertaa kuin on vaakarivejä eli 640 kertaa. Kuviossa 11 on esitetty vuonna 1992 julkaistu Wolfenstein 3D, joka on tunnetuin kaksiulotteista raycastingia käyttävä videopeli. Käytetty pelimoottori on kuitenkin rajoittunut: seinät ovat kaikki samankorkuisia ja 90 asteen kulmilla varustettuja, ja lattiassa ja katossa ei ole tekstuuripinnoitusta ollenkaan. Rajoitukset kuitenkin mahdollistivat pelin ajamisen aikanaan myös hitaammilla 286-prosessoreilla, vaikka se olikin suunnattu enemmän 386-prosessoreille. Yhtenä tapana on tehdä kaksi renderöintikierrosta, ensimmäisellä piirretään seinät, katto ja lattia ja toisella asetetaan ruudulle näkyvät bittikarttakuvat eli spritet, joiden avulla esitetään esimerkiksi esineet ja hahmot skenessä. (Vandevenne 2007.)

KUVIO 11. Tunnetuin 2D-raycastingia käyttävä Wolfenstein 3D (3D Realms 2013.)

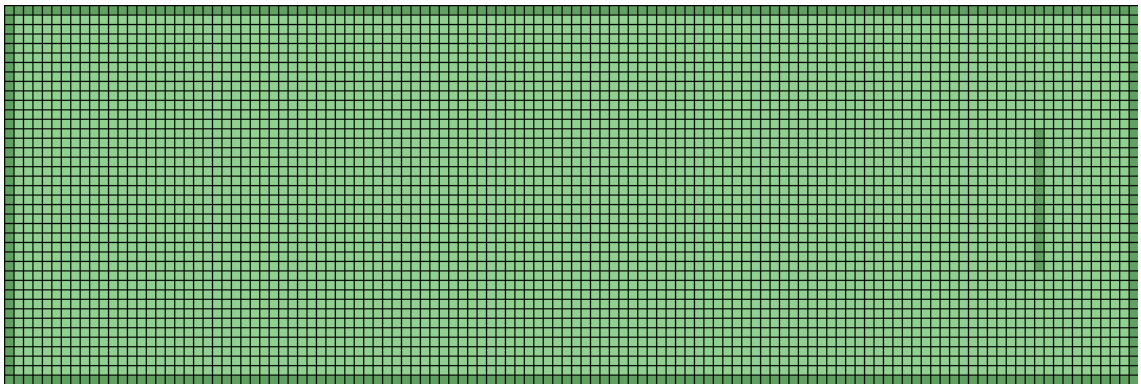
3.2.1 Rajoitukset

Tekniikan suurimpana rajoituksena geometristen rajoitusten lisäksi on kykenemättömyys Z-akselin ympäri kiertämiseen, eli katsetta ei voi kääntää ylös- tai alaspäin. Jos tämä sallittaisiin, seinät menisivät vinoon ja pystyrivien piirtämisen kaikki edut menetettäisiin. Näiden rajoitusten vuoksi raycastingin tekemää näkymää ei luokitella aidoksi kolmiulotteiseksi. (Permadi 2010b.)

Jotkin myöhemmät raycasting pelimoottorit sallivat ylös- ja alaspäin katsomisen, mutta tämä ei ole aitoa katseen siirtämistä. Efekti toteutetaan silmänkääntönä siirtämällä renderöitävän kuvan aluetta ylös- tai alaspäin nykyisestä näkymästä. Tämä aiheuttaa kuvan vääristymän, joka näyttää samanlaiselta kuin katseen siirtäminen ylös- tai alaspäin. (The Stack 2008.)

3.2.2 Toimintaidea

Raycastingissa kartta on kaksiulotteinen ruudukko, ja jokaisessa ruudussa on kokonaisluku, joka esimerkiksi nollana kuvaa tyhjää kohtaa, tai jokin positiivinen arvo, joka kuvaa seinää ja sitä, minkä tyyppinen seinä on. Kuviossa 12 on havainnollistettu karttaa ruudukon avulla, jossa tumman vihreät ruudut ovat esteitä ja vaaleat tyhjiä.



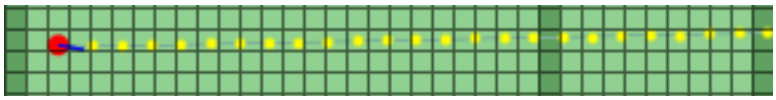
KUVIO 12. Raycastingin kaksiulotteinen ruudukkokartta

Jokaista ruudun pystyriviä kohti luodaan säde, jonka alkupiste on katsojan sijainti ja jonka suunta määräytyy katsojan suunnan ja ruudun x-koordinaatin perusteella. Kuviossa 13 on esitetty liikkeelle lähetetyt säteet sinisillä viivoilla. Säteet pysähtyvät, jos ne törmäävät ruutuun, joka on merkitty esteeksi tai säde pääsee kartan ulkopuolelle. Osuman jälkeen lasketaan etäisyys osumakohdasta katsojaan. Tuloksen perusteella lasketaan ruudulle piirrettävän esteen korkeus; suuremmalla etäisyydellä este on näkyy pienempänä ruudulla, ja vastaavasti pienellä etäisyydellä este esitetään isompana ruudulla. (Vandevenne 2007.)

KUVIO 13. Raycasting-säteet

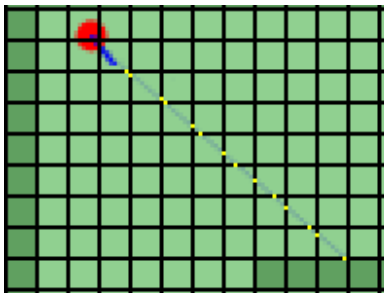
Esteen löytämiseksi pitää tarkistaa silmukan avulla sopivin askelein säteen matkalla, onko se mennyt ruutuun, joka on merkitty esteeksi. Säteen ollessa esteen sisällä silmukka voidaan lopettaa ja tehdä tarvittavat jatkotoimenpiteet. Jos yhden askeleen siirtymisen jälkeen säde on ruudussa, joissa ei ole estettä, silmukkaa jatketaan: lisätään tietty arvo säteen sijaintiin ja tarkistetaan uuden sijainnin kohta. Tätä tehdään niin kauan, kunnes este tulee vastaan. Ihminen näkee heti, missä kohtaa säde kohtaa esteen, mutta osumakohdan löytäminen tietokoneelle on vaikeampaa. Laskutoimituksia ei voi tehdä liikaa, tai muuten seurauksena olisi ruudunpäivitysnopeuden aleneminen. Yksinkertaisin tapa on asettaa aske-

leeksi vakioarvo, mutta tämä voi kuitenkin johtaa liian suurella askeleen pituudella mahdollisuuteen hypätä seinän yli. Kuviossa 14 on esitetty se tilanne, että jos askeleen pituus on liian suuri, keltaisten kohtien kohdalla tarkistetaan törmäys. Este jää tässä tilanteessa huomaamatta tarkistuskohtien ollessa vähän ennen estettä ja vähän sen jälkeen. (Vandevenne 2007.)



KUVIO 14. Liian harva säteen askeleen pituus

Edellä esitetyn ongelman voisi korjata lyhentämällä askeleen pituutta. Tämän seurauksena olisi kuitenkin se, että ruudunpäivitysnopeus laskisi törmäystarkistusten suuremman määrän vuoksi. Parempi ratkaisu tilanteeseen on käyttää askeleen pituutta, jonka suuruus riippuu siitä, kuinka kaukana seuraava tarkistettava ruutu on. Esimerkiksi jos asetetaan ruudun leveydeksi ja korkeudeksi sopivat kokonaisluvut, tarkistukseen riittää kokonaisluvullinen säteen x- tai y-koordinaatti, ja sen saavuttamisen kohdalla tarkistetaan nykyisen ja seuraavan leikkauspisteen välissä oleva ruutu. Kuviossa 15 on keltaisella värillä esitetty muuttuvapituinen askel, joka tarkistaa aina leikkauspisteen kohdalla, onko seuraavassa ruudussa este vai ei. (Vandevenne 2007.)

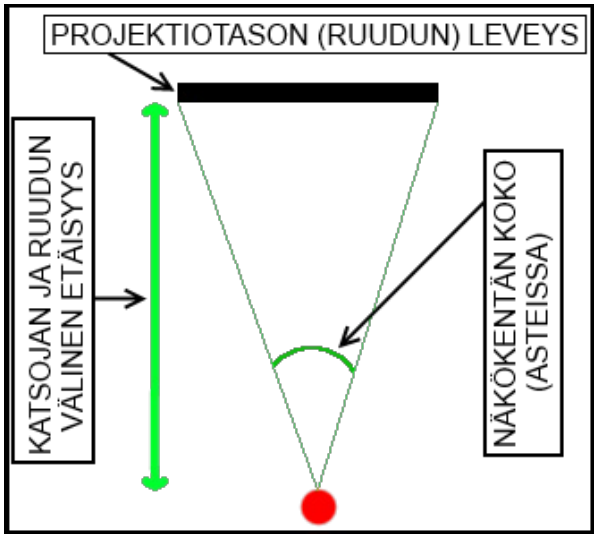


KUVIO 15. Muuttuva askeleen pituus

3.2.3 Näkymän luominen

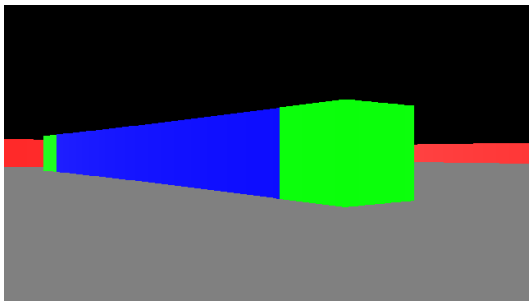
Kolmiulotteisen näkymän luomiseksi kaksiulotteisen raycastingin avulla tarvitaan katsojan korkeuden, sijainnin ja näkökentän (field of view) leveys. Laskemiseen tarvitaan myös projektiotason koko eli resoluutio, säteen lähtökulma sekä katsojan etäisyys projektiotasos-

ta. (Permadi 2010c.) Kuviossa 16 on esitetty kyseisiä tarvittavia arvoja näkymän luomiseksi. Seinää varten tarvitaan sen etäisyys katsojasta, ja tämän perusteella voidaan laskea kuinka korkea kyseinen seinä on ja piirtää se ruudulle.



KUVIO 16. Tarvittavat tiedot näkymän luomiseksi (Permadi 2010c.)

Näiden arvojen avulla saadaan muodostettua kolmiulotteinen näkymä. Kuviossa 17 on esitetty yksinkertainen näkymä. Seinän väri riippuu kokonaisluvusta, joka on kartassa kyseisessä kohdassa. Katon ja lattian värit ovat molemmat värejä jotka ulottuvat ylä- tai alalaidasta ruudun puoleen väliin.

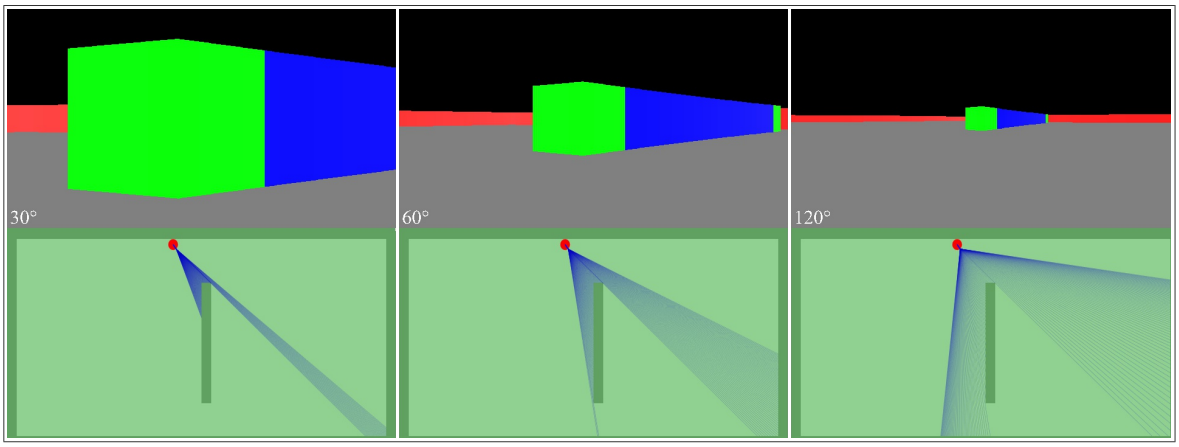


KUVIO 17. Yksinkertainen kolmiulotteinen näkymä

3.2.4 Näkökentän leveys

Näkökentän leveydellä (field of view) ilmaistaan kuinka paljon näkymästä oikeasti nähdään, ja arvo esitetään yleensä asteina (Optics HQ, 2011.). Pienemmällä arvolla kaukana

olevat kohteet näkyvät isompina, ja pienentämällä näkökenttää saadaan esimerkiksi zoomattua samalla tavalla kuin katsottaisiin kohdetta kiikareilla. Suurentamalla arvoa näkyvät kohteet pienenevät ja näyttävät olevan kauempana. Kuviossa 18 on esitetty sama näkymä eri näkökentän asteilla. Pienemmällä luvulla näkymä on pienempi ja lähempänä oleva, suuremmilla taas laajempi sekä objektit näyttävät olevan kauempana. Näkymän vieressä on näkymä ylhäältäpäin. 120 asteen kuvassa näkyy, miten säteet ovat tiheämpänä reunoissa, tämä aiheuttaa vääristymää kuvassa, ilmiö ei kuitenkaan esiinny kuviossa, mutta on kuitenkin olemassa.



KUVIO 18. Eri näkökentän leveyksiä

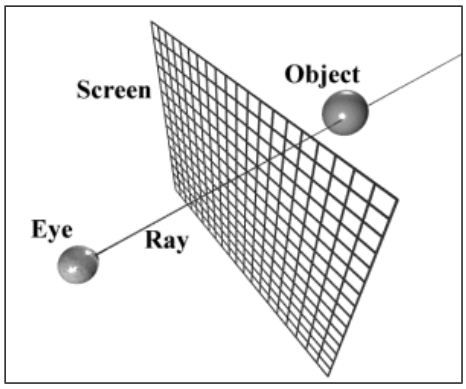
Oikean arvon asettaminen on erittäin tärkeää, sillä eri ihmiset reagoivat eri arvoihin eri tavalla. Hyvä tyyli onkin asettaa, jos mahdollista, arvo käyttäjän säädettäväksi. Liian pieni arvo liioittelee liikettä, eli pieni liike näkyy suurena liikkeenä ruudulla ja voi mahdollisesti aiheuttaa pahoinvointia katsojalle. Vastaavasti liian suuri arvo taas vähättelee liikettä, eli suuri liike näkyy pienenä liikkeenä ruudulla. Oikean arvon kummallakin puolella oleva arvo vääristää perspektiiviä, mikä johtaa pallon muotoisten geometristen muotojen projektoinnista ruudulle. Suuri arvo suurentaa esiintyvää vääristymää, ja pieni arvo litistää perspektiiviä. Optimaalisimman arvon voi teoreettisesti laskea sovittamalla kameran näkökentän leveyden katsojan silmän ja näytön reunan väliseen kulmaan ja siihen, kuinka kaukana katsoja on näytöstä. Tämä etäisyys vaihtelee sen mukaan, kuinka kaukana katsoja on ja minkä kokoinen käytetty näyttö on. (Valve Developer Community 2012.)

3.3 3D-raycasting

Kolmiulotteinen raycasting eroaa kaksiulotteisesta lähettämällä säteen jokaista ruudun pikseliä kohti, ei vain jokaista pystyriviä kohti, muuten yhden ulottuvuuden lisäämisen lisäksi toiminta on lähellä kaksiulotteisen toimintaa.

3.3.1 Toimintaidea

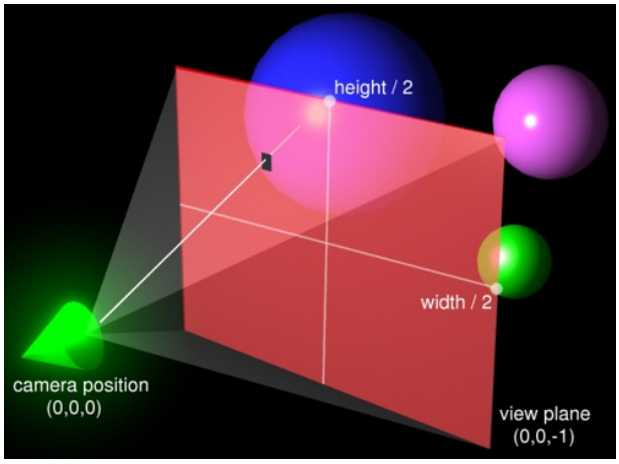
Ideana on lähettää säteitä katsojasta ruudun jokaista pikseliä kohden verrattuna kaksiulotteiseen raycastingiin, joka lähettää säteitä jokaista pystyriviä kohden, ja etsiä lähin objekti, joka estää säteen kulkemisen eteenpäin. Varjostus määritellään objektin materiaalin ja skenen valaistustason avulla. Laskemisessa tehdään se oletus, että objektin pinta on aina valoa kohti ja valon pääsyä pinnalle ei estä mikään. Säteen leikatessa matemaattisesti määritellyn pinnan tämän voi renderöidä raycastingin avulla. (Macey 2013, 4.) Kuviossa 19 on esitetty raycastingin perusidea.



KUVIO 19. 3D Raycastingin- ja raytracingin perusidea (Buck 2000.)

3.3.2 Säteen muodostaminen ja projektointi

Raycastingin ja raytracingin ensimmäinen askel on lähettää säde silmästä kohti skeneä. Tähän tarvitaan kamera, jonka sijainnista lähetetään säde projektiotason läpi. Projektiotasosta muodostetaan kuva, joka näytetään ruudulla. (UnknownRoad 2014.) Kuviossa 20 on esitetty visuaalisesti, miten säde muodostetaan.



KUVIO 20. Säteen muodostaminen (UnknownRoad 2014.)

Kuvan muodostaminen projektiotasosta (view plane) edellyttää muodostettavan kuvan leveyden ja korkeuden määrittämistä pikseleissä. Tähän tarvitaan myös näkökentän (Field of View, esiintyy yleensä lyhenteenä FOV) koko asteissa, ja tämän leveysarvo määrittää sen kuinka laaja kuva on sivusuunnassa, ja korkeusarvo sen, kuinka laaja kuva on korkeussuunnassa. Näkökentän arvoilla on suuri vaikutus minkälaiselta muodostettu kuva näyttää. Korkeusarvo lasketaan yleensä leveysarvon ja käytetyn kuvasuhteen avulla. Arvo voi olla periaatteessa mikä tahansa, mutta yleensä liikaa poiketessa oikeasta arvosta se voi aiheuttaa vääristymiä muodostettuun kuvaan. Seuraavien kaavojen avulla voidaan laskea näkökentän arvot. (UnknownRoad 2014.)

$$FOV_X = \frac{\pi}{4}$$

$$FOV_Y = \frac{KORKEUS}{LEVEYS} \times FOV_X$$

Säde lähetetään jokaisen pikselin keskiosaan ja tähän tarvitaan pikselin koordinaatit. Näitä kutsutaan u- ja v-koordinaateiksi ja niillä esitetään lopputuloksena ruudulle piirrettävän rasteroidun kuvan koordinaatteja. Koordinaattien avulla voidaan laskea projektiotason vastaavat koordinaatit, joita kutsutaan x- ja y-koordinaateiksi. Tämän jälkeen säde on mahdollista muodostaa. Säde alkaa kameran koordinaateista, ja sen suunta lasketaan niin, että se

on kohtisuorasti seuraavien kaavojen avulla laskettuja projektiotason x- ja y-koordinaatteja kohti. (UnknownRoad 2014.)

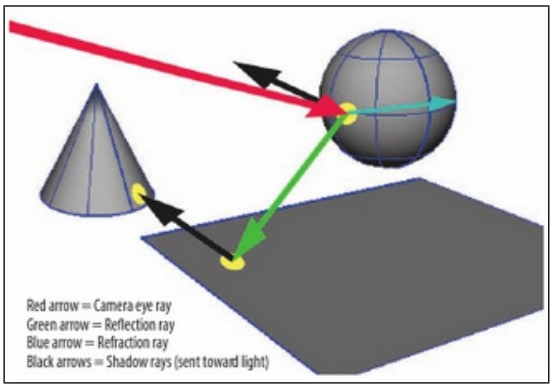
$$x = \left(\frac{2 \times u - LEVEYS}{LEVEYS}\right) \times \tan\left(FOV_X\right)$$

$$y = \left(\frac{2 \times v - KORKEUS}{KORKEUS}\right) \times \tan\left(FOV_Y\right)$$

3.4 Raytracing

Raytracing on ylikuormitettu termi, jonka merkitys riippuu käyttökohteesta. Se voi viitata kolmiulotteisten viivojen, kuten säteiden ja objektien, välisten leikkauspisteiden laskemiseen, tiettyihin algoritmeihin, kuten Whittedin kuvan generointimetodeihin, öljy-yhtiöiden käyttämiin algoritmeihin, joiden avulla simuloidaan aaltojen etenemistä, tai sitten algoritmi-ryhmään, johon Whittedin algoritmi myös kuuluu. (NVIDIA 2012, 2.)

Raytracing tarjoaa kehittyneempiä ominaisuuksia, joiden avulla voidaan ottaa huomioon heijastetut, taittuvat ja epäsuorat valot, joihin ympäristö vaikuttaa. Nämä sallivat HDR-kuvien käyttämisen oikean maailman valon voimakkuuksien simulointiin. Raytracingia on laajennettu ottamalla huomioon valon energia, joka heijastuu tiettyyn pisteeseen muista maailman objekteista. Raytracingillä on pääsy kaikkiin maailman objekteihin, toisin kuin scanline- eli rivikohtaisella. Tällä on pääsy vain objekteihin, jotka kamera näkee sillä hetkellä. Näin saadaan kuvan luonnille paras mahdollinen tarkkuus jokaisesta näkökulmasta. Kuviossa 21 on kuvattu yksi ensisijainen säde, joka osuu kohteeseen, ja sen jälkeen osumakohdasta on mahdollista lähettää toissijaiset säteet. Toissijaisen säteen osumakohdasta on mahdollista taas lähettää uudet toissijaiset säteet. Kuviossa 22 on esitetty raytracingin toimintaidea pseudokoodin muodossa. Raycastingin pseudokoodi on muuten sama, se rajoittuu vain ensimmäiseen kohtaan. (Boaz 2008, 44.)



KUVIO 21. Raytracingin säteet (Lanier 2008, 340.)

```
FOR all pixels P0 DO
 1. calculate viewing ray from the eye through P0
     intersect ray with all objects and choose closest intersection P
 2. FOR all light sources L DO
     intersect shadow ray P -> L with all objects
     IF no intersection between P and L THEN shading += influence of L
 3.IF P is on a reflective surface
   THEN trace secondary ray; shading += influence of reflection
 4.IF P is on a transparent surface
   THEN trace secondary ray; shading += influence of transparency
```

KUVIO 22. Raytracingin pseudokoodi (Purgathofer 2011, 36.)

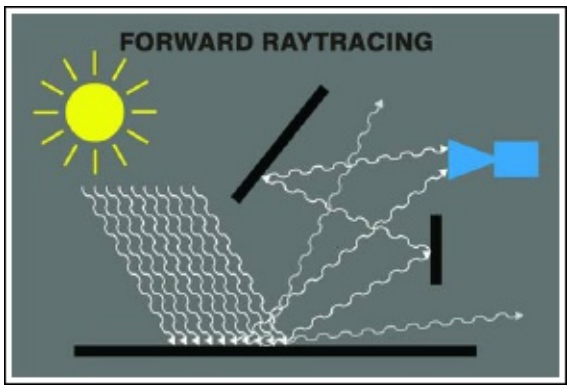
Raytracingiä on käytetty paljon ei-graafisissa ohjelmistoissa: suunnittelussa sitä käytetään arvioimaan jonkin objektin tilavuus ja määrittämään monimutkaisten objektien läheisyys ja törmäys. Objekti valitaan hiiren painalluksella ja sen jälkeen määritetään, mikä objekti on painetussa kohdassa, esimerkiksi hiirellä vedetty valintaruutu tai painallus ruudulla. (NVIDIA 2012, 3.)

Yhtenä ominaisuutena on raytracingin suorituskyky, joka ei ole suoraan verrannollinen objektien määrään. Kaksinkertaistamalla objektien määrä ei välttämättä kaksinkertaista renderöintiaikaa. Tämä saavutetaan järjestämällä objektit helposti hallittaviin rakenteisiin, joiden avulla ne pystytään ohittamaan nopeasti, jos säde ei tule ikinä osumaan niihin. (NVIDIA 2012, 3.)

Raytracingin graafinen määritelmä

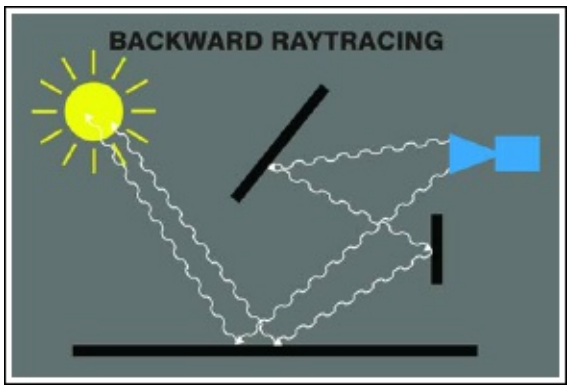
Graafisella puolella raytracing määritellään valonlähteen säteiden seuraajaksi näiden hyppiessä pinnalta pinnalle. Osa näistä säteistä voi päätyä lopuksi kameraan eli katsojan silmään, ja tätä kutsutaan eteenpäin kulkevaksi raytracingiksi. Valtaosa seuratuista säteistä ei

kuitenkaan ikinä päädy kameran näkökenttään, ja tästä aiheutuu on paljon turhaa säteiden seuraamista, jonka seurauksena on suuri tehokkuuden ja nopeuden lasku, kuten kuviossa 23 on esitetty. (Summers 2004, 230.)



KUVIO 23. Eteenpäin kulkeva raytracing (Summers 2004, 230.)

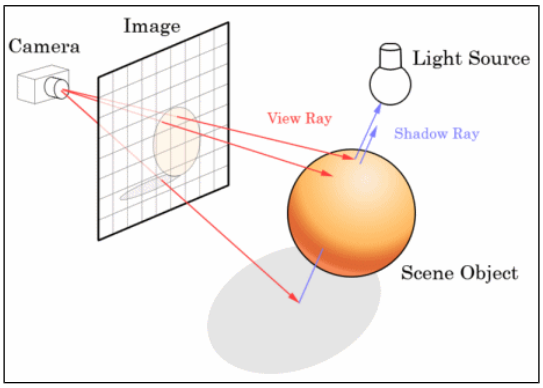
Edellä esitetty ongelma on ratkaistu kehittämällä algoritmeja seuraamaan säteitä takaperin eli kamerasta samanlaisella pintojen välisillä hyppimisillä, kunnes lopulta päädytään valonlähteeseen. Tätä taas kutsutaan takaperin kulkevaksi raytracingiksi, joka on esitetty kuviossa 24. Käänteisen suunnan avulla seurataan pelkästään oikeasti näkökenttään osuvia valonsäteitä. Seurattavien säteiden määrän vähenemisen vuoksi tehokkuus ja nopeus paranevat, mutta siitä huolimatta raytracing on todella raskasta ja vaatii tehokkaan prosessorin. (Summers 2004, 230 – 231.)



KUVIO 24. Takaperin kulkeva raytracing (Summers 2004, 230.)

Yleisin ominaisuus käytetyillä algoritmeilla on säteiden ja objektien välisen leikkauspisteen laskeminen. Renderöinnissä leikkauspisteen optiset ominaisuudet määrittävät, sen mitä säteelle tapahtuu kyseisessä kohdassa (heijastuuko se, katoaako kokonaan vai hajaan-

tuuko se eri suuntiin). Kaikissa käyttötarkoituksissa kuitenkaan millään muulla tiedolla ei ole mitään väliä kuin sillä, missä leikkauspiste on vai onko sitä ollenkaan. Yksi tällainen käyttötarkoitus on törmäystunnistus eli tunnistetaan, törmääkö säde johonkin vai ei, ja muilla tiedoilla ei ole mitään väliä. (NVIDIA 2012, 3.) Kuviossa 25 on esitetty tilanne, jossa oleva varjosta tarvitaan vain tunnistamaan, törmäsikö säde johonkin.



KUVIO 25. Raytracingin varjostus (Camargo 2009.)

Raytracingillä on useita etuja perinteisempiin rasterointitekniikoihin verrattuna. Optiset efektit, kuten valon heijastukset ja taittumiset, voi toteuttaa muutamalla koodirivillä, kiinteät varjot voidaan luoda helposti, ilman että tarvitsee muodostaa ylimääräistä varjokarttaa, josta nähdään, mitkä kohdat näkymästä ovat varjossa, ja myöskään pehmeiden varjojen toteutus ei ole vaikeampaa. (NVIDIA 2012, 3 – 4.)

3.5 Tekniikoiden erot

Raycastingilla ja raytracingillä on molemmilla sama määritys: määrittää pintojen näkyvyyden seuraamalla mielikuvituksellisia valonsäteitä katsojan silmästä kohteisiin: Tämä antaa sellaisen kuvan, että tekniikat olisivat samoja. Kirjallisuudessa termejä käytetään joskus ristiin, kuitenkin ohjelmoijan näkökulmasta raycasting on raytracingin aliluokka. Aliluokkanäkemys on tehty sen vuoksi, koska kaksi- ja kolmiulotteinen raycasting ovat nopeudeltaan parempia kuin raytracing. Nopeusero johtuu siitä, että kaksiulotteinen raycasting käyttää yksinkertaisia pystyrivikohtaisia säteitä sekä geometrisia rajoitteita renderöinnin nopeuttamiseksi ja kolmiulotteinen raycasting rajoittaa säteiden määrää yhteen jokaista pikseliä kohden. Edellä mainituista rajoitteista esimerkiksi seinät ovat aina kohtisuorassa lat-

tiaan tai kattoon nähden. Ilman geometrisiä rajoituksia raycasting ei olisi käyttökelpoinen. Tämä rajoittaa objektien muotoja; mielivaltaisista kuvioista on mahdotonta löytää rajoitteita joita voisi käyttää nopeutuksessa. (Permadi 2010a.) Taulukossa 1 on esitetty tekniikoiden pääeroja.

TAULUKKO 1. Raycastingin ja raytracingin eroja (mukaillen Permadi 2010a.)

	Kaksiulotteinen raycasting	
Periaate	Säteitä lähetetään ja seurataan ryhmissä perustuen geometrisiin rajoituksiin. Esimerkiksi resoluutiolla 640 x 480 raycaster seuraa 640 sädettä (luku tulee resoluution vaakatarkkuudesta, joten kuvassa on sen verran pystyrivejä).	
Laskukaava	Yleensä epätarkka.	
Nopeus	Nopea, sopii hyvin reaaliaikaiseen renderöintiin.	
Laatu	Muodostettu kuva ei ole kovin realistinen. Yleensä se on aika kulmikas. (KUVIO 26)	
Maailma	Rajoitettu yhdellä tai useammalla geometrisellä rajoituksella.	
	Kolmiulotteinen raycasting	**Raytracing**
Periaate	Jokaista sädettä seurataan erikseen eli jokaista pikseliä (pikselikoon ollessa 1 x 1) kohden lähetetään oma säde. Esimerkiksi resoluutiolla 640 x 480, kolmiulotteisen raycasterin ja raytracerin seuraamien säteiden määrä on 307200 (verrattuna 2D-raycasterin 640 säteeseen).	
Laskukaava	Yleensä tarkka.	
Nopeus	Hitaampi kuin kaksiulotteinen raycasting, nopeampi kuin raytracing; vaatii silti paljon tehoa sulavaan reaaliaikaiseen renderöintiin.	Todella hidasta; vaati enemmän tehoa kuin kolmiulotteinen raycasting reaaliaikaiseen renderöintiin.
Laatu	Muodostettu kuva on huomattavasti realistisempi kuin kaksiulotteisella raycastingilla, mutta yksinkertaisempi kuin raytracingillä, varjostusta ei ole ollenkaan. (KUVIO 27)	Muodostettu kuva on realistinen, joskus jopa liiankin realistinen. (KUVIO 28)
Maailma	Melkein minkä tahansa muodon voi renderöidä.	

KUVIO 26. Esimerkki 2D-raycastingista (The Economist 2013.)

KUVIO 27. Esimerkki 3D-raycastingista (Oatley 2009.)

KUVIO 28. Esimerkki raytracingista (NVIDIA 2010.)

3.6 Intelin raytracing-projekti

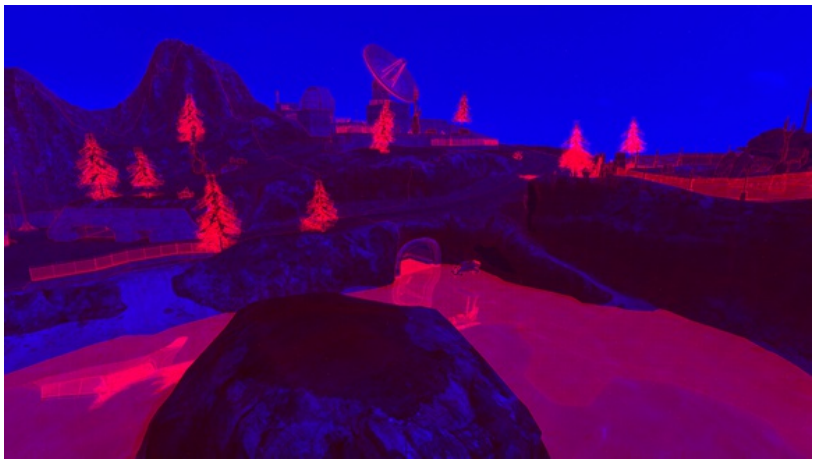
Intel on yksi suuri raytracingin kannattaja ja se on perustanut oman tekniikkaan keskittyvän osastonsa. Vuonna 2008 yritys muutti Quake Wars: Enemy Territory -videopelin käyt-

tämän perinteisesti tehdyn pelimoottorin raytracingillä toimivaksi. Kaikki nykyajan videopelit käyttävät renderöintiin rasterointia. Rasteroinnin ohjelmointi on vaativaa, ja monet erikoisefektit, kuten varjot ja heijastukset, vaativat niiden likiarvoisen tuloksen laskemisen useilla eri renderöintikierroksilla. Laskutoimitusten tulos tallennetaan yleensä omalle tekstuurille, joka asetetaan alkuperäisen tekstuurin päälle. Laskettu likiarvoinen tulos voi kuitenkin mahdollisesti epäonnistua. Raytracingillä varjon laskemisessahan ei tarvita muuta kuin tieto siitä, onko kyseisen kohdan ja valonlähteen välissä este vai ei. (Pohl 2012.)

Kehittäjät eivät yleensä käytä oikeaa kolmiulotteista geometriaa, sen sijaan kolmiulotteisista ominaisuuksista luodaan arvio kaksiulotteisen pinnan avulla, yleensä kahdesta kolmiosta eli polygonien avulla luodulle pinnalle, kuten kuviossa 29 on esitetty. Oikeanlaisen varjon luomisen rasteroinnilla edellä mainitulla tekniikalla ei ole helppoa. Yleisimmin käytetty algoritmi varjon laskemiseen on nimeltään ”shadow mapping” eli varjon kartoitus, joka ei anna hyödyllistä tietoa varjon muodostamiseen läpinäkyvässä objektissa. Tämän vuoksi varjot on joskus sisällytetty tekstuureihin, minkä seurauksena on staattinen varjon, joka ei muutu, jos esimerkiksi valon lähteen, kuten auringon sijainti muuttuu. Raytracingillä algoritmit ovat yksinkertaisempia. Varjostussäteen osuessa objektiin sen läpinäkyvyysarvo voidaan lukea tekstuurista ja sitten vain jatketaan eteenpäin, jos kyseinen kohta on läpinäkyvä. Etuna raytracingin käytettäessä läpinäkyviä objekteja ei tarvitse järjestää syvyyden mukaan. (Pohl 2012.)

KUVIO 29. Kahden kolmion pinnalle osittain läpinäkyvän tekstuurin asetus (Pohl 2012.)

Puun tai muun paljon läpinäkyvyyttä sisältävän objektin renderöinti raytracingillä on raskasta. Kuviossa 30 esitetään visuaalisesti, kuinka raskasta erilaisten objektien renderöinti on. Sinisemmällä olevat pikselit ovat kevyempiä renderöidä ja kirkkaalla punaisella olevat pikselit ovat raskaimpia. Kuviosta näkyy selvästi veden pinnan ja varsinkin puiden olevan ylivoimaisesti raskaimpia kohteita renderöidä. (Pohl 2012.)



KUVIO 30. Raytracingin värikoodattu suorituskyky (Pohl 2012.)

Suorituskyky on pääsyy siihen, miksi raytracing ei ole tällä hetkellä käytössä videopeleissä. Verrattuna rasterointiin erikoistuvaan laitteisto, kuten esimerkiksi nykyisiin näytönohjaimiin on raytracing hidasta. Myös prosessorin teksturointiyksiköiden puuttumisen vuoksi raytracing hidastuu entistä enemmän, jos kaikissa tekstuuripinnoissa käytetään trilineaarista suodatusta. Projektin tietokoneissa käytettiin useita Intelin uusimpia ja nopeimpia Intel Xeon X7460-prosessoreita, jotka toimivat 2,66 gigahertsin kellotaajuudella, ja joita oli neljä kappaletta. Tällä kokoonpanolla ruudunpäivitysnopeus vaihteli 1280 x 720 resoluutiolla 20–35 kuvan välillä. Vaikka tulos ei ole nopein mahdollinen, se on kuitenkin paljon nopeampi verrattuna aikaisempaan vuonna 2004 tehtyyn raytracing-projektiin, joka vaati 20 tietokonetta renderöimään nykyisiä yksinkertaisempaa videopeliä, ja suuresta määrästä huolimatta renderöinti oli hitaampaa ja vaati pienemmän resoluution. Maailmanlaajuisesti on tehty erilaisia tutkimuksia, joiden tulosten pohjalta on saatu parannettua suorituskykyä käyttämällä useampiytimisiä prosessoreita, jotka käyttävät rinnakkaislaskentaa. (Pohl 2012.)

4 DEMONSTRAATIO

Aloittaessani opinnäytetyötä itselläni ei ollut aikaisempaa kokemusta kolmiulotteisen grafiikan luomisesta, ja näin aihe tarjosi paljon uusia asioita sekä haasteita. Demonstraationa aiheen tutkimisen lisäksi oli ajatuksena luoda kohtuullisen yksinkertainen skene käyttämällä raytracing-tekniikkaa ja niiden pohjalta tutkia miten nopeasti raytracingiä voisi ajaa ny-

kyisillä selaimilla ja mobiililaitteilla, sekä pohtia tekniikan erilaiset mahdolliset käyttökohteet. Säteet on demonstraatiossa rajattu ensisijaiseen ja yhteen toissijaiseen varjostussäteeseen, ja heijastus- ja hajaantumissäteet on jätetty kokonaan pois, koska nämä toimivat pääpiirteittäin samalla tavalla mukaan otettujen säteiden kanssa.

4.1 Käytetyt tekniikat

Tässä luvussa käydään läpi erilaisia tekniikoita ja termejä, joita demonstraation muodostamisen apuna on käytetty.

4.1.1 Vokseli

Voxel (vokseli) on lyhenne sanoista ”volumetric pixel” eli tilavuuspikseli, joka kuvaa kolmiulotteisen kuvan pienintä mahdollista näkyvää osaa. Se esittää tiettyä ruudukon arvoa kolmiulotteisessa avaruudessa. (technopedia 2014.)

Vokseleilla voidaan mallintaa kolmiulotteisia objekteja käyttämällä näytepisteitä (vokseleita) pinnan sijasta. Vokseleiden etuna on kyky käsitellä korkeatasoisia objekteja ja objekteja, joilla on vaikeasti tai huonosti määriteltäviä reunoja, kuten esimerkiksi pilviä. Suurin este vokseleiden käyttämiseen grafiikassa on ollut laskentatehon ja standardien puute. Nykyään kyseiset esteet eivät ole kasvaneen laskentatehon, sekä jatkuvasti paremmiksi kehittyvien pakkausalgoritmien ansiosta niin suuria. Vokselipohjaista grafiikkaa kutsutaan myös tilavuusgrafiikaksi. (Accomazzi 2011.)

Tilavuusgrafiikalla voi kuvata objekteja, ilman että niille tarvitsee määritellä rajoja ollenkaan, kuten yleisesti käytettyjen polygoneiden kanssa pitää tehdä. Vokselit toimivat hyvin korkealaatuisten objektien, kuten ihmisen ruumis, esittämisessä ja on tämän vuoksi todella laajassa käytössä lääketieteen puolella. Muistin käyttö ja renderöintinopeus riippuvat tietoaineiston koosta, eivät siitä, miten monimutkainen objekti on. Vastaavasti mitä monimutkaisempi polygoneilla tehty objekti on kyseessä, sitä suurempi on resurssien käyttö. Monimutkaisemman objektin renderöintiin tarvitaan enemmän polygoneja. (Accomazzi 2011.)

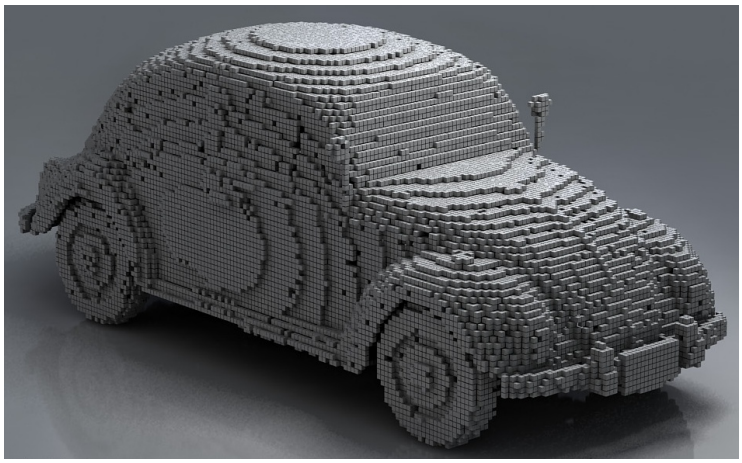
Vokseleiden käyttöä ja yleistymistä rajoittavat neljä suurinta syytä tällä hetkellä ovat seuraavat:

- riittämätön laskentateho ja muistiväylän kaista
  suuren tietoaineiston renderöimiseen tarvitaan yleensä kallis työasema, jossa on yleensä useampi prosessori
- eri ohjelmistojen kommunikoinnin mahdollistavien standardien puute
  lääketieteessä rajoitus ohitettu kehittämällä ”Digital Imaging and Communications in Medicine” (DICOM) -standardi
- tietoisuuden puute teknologiasta
  polygoneilla on tällä hetkellä suurempi suosio.
- halpojen laskentateholtansa riittävien laitteistojen (mm. tietokoneiden) saatavuus. (Accomazzi 2011.)

Kehitystä on kuitenkin tapahtunut rajoitusten vaikutusten vähentämiseksi tai poistamiseksi:

- Nopeammat prosessorit, kuten Pentium 4, ja nopeammat prosessorit mahdollistavat nopeamman ruudun päivitysnopeuden verrattuna vanhempiin prosessoreihin
- Pakkausformaattien kehitys, kuten esimerkiksi JPEG2000:n, tukee kolmiulotteisen tilavuuden pakkausta käyttämällä aallokkeenmuunnosta.
- Vokseleiden käyttö erilaisissa ohjelmistoissa kasvaa, grafiikassa paremman realistisuuden tavoittamisen ansiosta vokselit ovat alkaneet kasvattamaan suosiota. (Accomazzi 2011.)

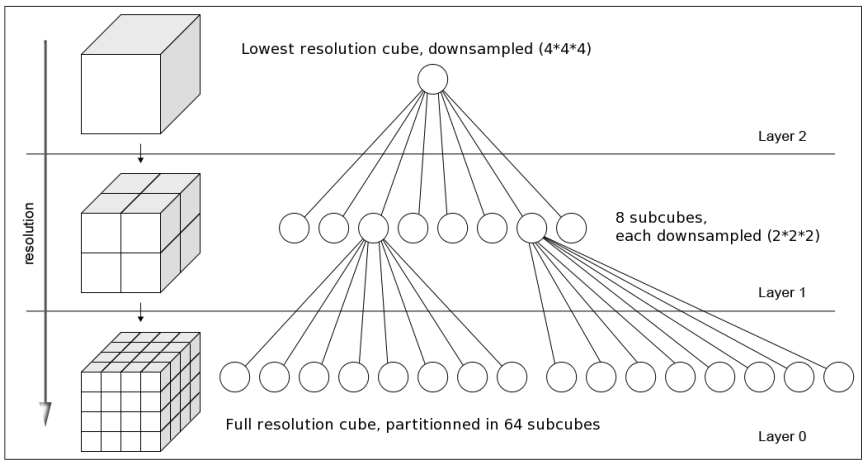
Kuviossa 31 on esimerkkinä vokseleilla rakennettu auto. Se ei näytä kovinkaan hienolta mutta vokseleiden kokoa voidaan pienentää ja lisätä, jolloin seurauksena olisi realistisemman näköinen auto. Sen renderöinti olisi vastaavasti hitaampaa vokselimäärän kasvaessa ja tarkkuuden parantuessa.



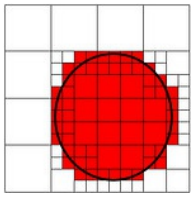
KUVIO 31. Esimerkki vokseliobjektista (Bilderzucht-blog 2010.)

4.1.2 Sparse Voxel Octree

Jos vokseleilla haluaisi luoda tarkan mallin eikä se näyttäisi niin kuutiomaiselta, tarvittaisiin vokseleita todella suuri määrä, joka taas vaatisi suuren määrän muistia. Tarvittavan muistin määrän vähentämiseksi voidaan käyttää esimerkiksi jonkinlaista puurakennetta helpottamaan ja vähentämään tarvittavien resurssien määrää. Ideana on aloittaa ”juuresta” ja jakaa malli kahdeksaan yhtä suureen osaan ”oksaan”, ja tätä jatketaan eteenpäin jakamalla ”oksia” kahdeksaan osaan, jos kyseinen alue ei ole tyhjä. (Anteru 2008.) Kuviossa 32 on esitetty kuutio jolle tehdään puurakennejako kolmelle tasolle. Kuviossa 33 on esitetty kaksiulotteisesti miten SVO:lla jaetaan osiin kuvio, kaksiulotteista octreetä kutsutaan quadtreeksi. Puurakenteeksi valitsin ”Sparse Voxel Octreen” (SVO) eli ”väljän vokseli puurakenteen”, joka on nimensä mukaisesti aika yksinkertainen hierarkkinen puurakenne.

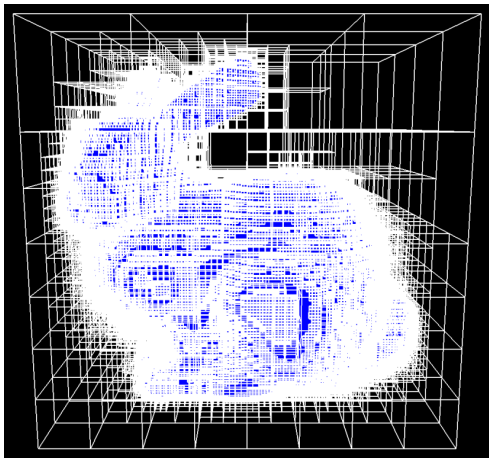


KUVIO 32. Esimerkki Sparse Voxel Octreestä (Gonzalez, Lucchi & Fua 2013.)



KUVIO 33. Sparse Voxel Octree kaksiulotteisesti (Abi-Chahla 2009b.)

Etuna puurakenteessa on se, että on helppo ohittaa tyhjät oksat, objektin tarkkuutta pystyy hallitsemaan rajoittamalla puurakenteen syvyyttä siinä, kuinka syvälle algoritmi voi edetä, ja rakenteen avulla on mahdollista saavuttaa korkealaatuinen geometrinen tarkkuus. (Anteru 2008.) Kuviossa 34 on esitetty puurakenteeseen jaettu kolmiulotteinen objekti. Näkyvissä on miten objekti on jaettu erisuuruisiin neliöihin. Esimerkiksi isommat oikeassa yläkulmassa olevat neliöt ovat tyhjiä, ja tämän vuoksi niitä ei ole tarve enää jakaa pienemmiksi.



KUVIO 34. 3D-malli SVO-puurakenteessa (Darnell 2011.)

4.1.3 Perlin Noise

1980-luvulla aloitettiin kohinafunktioiden kehittäminen, jonka tavoitteena oli tarjota vaihtoehtoinen ja yksinkertainen tapa objektien teksturointiin. Tietokoneilla oli tuolloin todella rajallinen muistikapasiteetti ja tekstuureina käytetyt kuvat eivät näin mahtuneet kovinkaan hyvin keskusmuistiin. Tietokonegrafiikkaan erikoistuneet yritykset alkoivat etsiä vaihtoehtoista tapaa, joka tarvitsisi vähemmän keskusmuistia. Pelkästään kiinteällä värillä renderöidyt objektit ovat liian puhtaan näköisiä. Tämän puhtaan pinnan rikkomiseksi voidaan moduloida objektin visuaalisia ominaisuuksia, esimerkiksi väriä ja kiiltävyyttä. Yleensä tähän

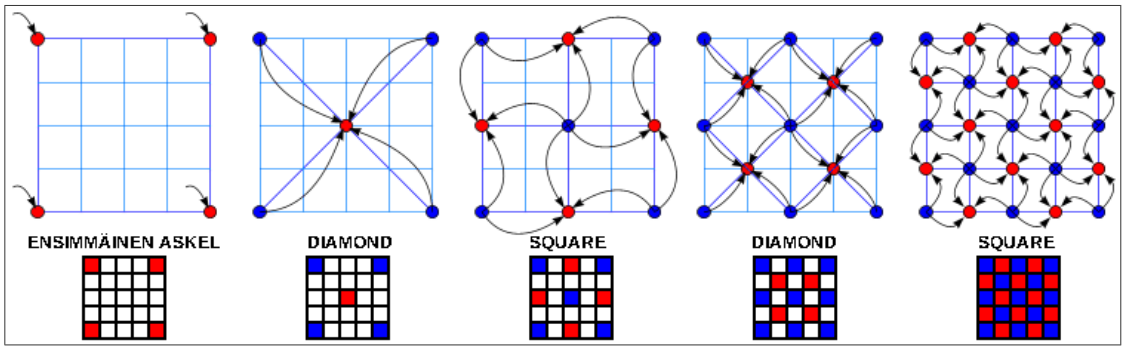
käytettään satunnaisuuden luomiseksi satunnaislukugeneraattoria, joka ei kuitenkaan ole riittävä. Satunnaiset kuviot luonnossa ovat yleensä todella pehmeitä, kaksi objektin pinnalla sijaitsevaa pistettä näyttävät melkein samoilta ollessaan lähellä toisiansa, ja sijaitessaan kaukana toisistaan ne eivät näytä samanlaisilta ollenkaan. Ohjelmointikielien satunnaislukugeneraattoreilla ei ole kyseistä ominaisuutta lainkaan. Jokaisella satunnaisluvun luontikerralla luvun arvo ei ole mitenkään suhteellinen edelliseen satunnaislukuun verrattuna. Ken Perlin kehitti Tron -elokuvaa varten kohinafunktion, joka lisää satunnaislukuun ominaisuuden, jonka avulla sen on mahdollista olla suhteellinen vieressä olevan luvun kanssa, ja muodostettu pinta on luonnollisen ja realistisen näköinen. (Scratchapixel 2012.)

4.1.4 Diamond-Square

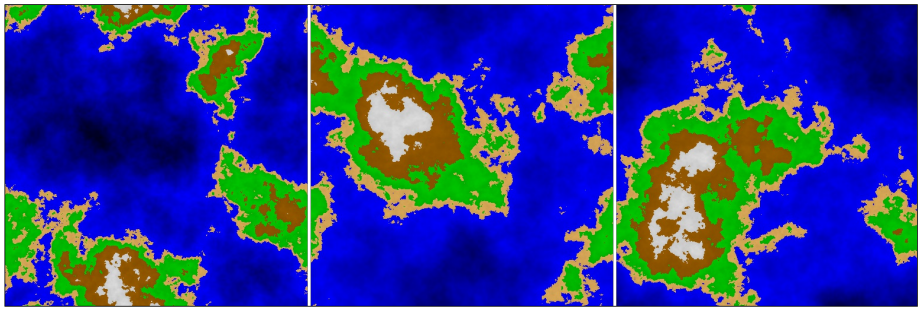
Diamond-Square (timantti-neliö) on algoritmi satunnaisen maaston luontiin. Luotu maasto on uskottavan näköinen, ja sen reunat voi tehdä jatkuvaksi, mikä mahdollistaa maaston toiston. Saman maastokuvan voi laittaa vierekkäin monta kertaa, ja se sulavasti jatkuu eteenpäin, tietenkin maasto on aina sama. Algoritmissa arvot luodaan kaksiulotteiseen taulukkoon, jonka koko on kahden potenssissa, ja sen jälkeen siihen lisätään yksi. (Martz 1997.)

Ensimmäisessä askeleessa taulukkoon asetetaan jokaiseen kulmakohtaan aloitusarvo. Aloitusarvojen sijoittamisen jälkeen tehdään timanttiaskel, ja siinä otetaan jokaisen äsken asetettujen kulmien arvot ja lasketaan niistä keskiarvo, johon lisätään satunnaisluku. Laskettu arvo sijoitetaan otettujen arvojen keskipisteeseen, joka on kulma-arvojen keskipisteessä. Askelta kutsutaan timanttiaskeleeksi, koska tämä luo timanttimuotoisen kuvion pisteiden yhdistämisen jälkeen. Tämän jälkeen tehdään neliöaskel, jossa otetaan laskettavan kohdan jokaisesta pääilmansuunnasta yhteenlasketut keskiarvot. Tätä kutsutaan neliöaskeleeksi, koska timanttiaskeleen tavoin viivojen yhdistämisen jälkeen saadaan neliön muotoinen kuvio. Tämän jälkeen puolitetaan askeleen pituus, lyhennetään satunnaislukuarvojen väliä ja tehdään timanttiaskel. Tätä jatketaan, kunnes jokainen taulukon kohta sisältää arvon. (Hughes 1998.) Kuviossa 35 on esitetty algoritmin kulku, ja siinä alempana on ylemmän kuvan kulkua vastaava kaksiulotteinen taulukko. Kuviosta ilmenee, miksi taulukon kahden potenssin kokoon pitää lisätä yksi: taulukosta ei löydy keskikohtaa, jos sen kokoa ei suuren-

neta yhdellä leveys- ja pystysuunnassa. Kuviossa 36 on esitetty muutama algoritmin avulla luotu maasto, ja sen ominaisuuksia ja muotoa voi muuttaa vaihtelemalla parametreja.



KUVIO 35. Diamond-Square-algoritmin kulku (Hughes 1998.)

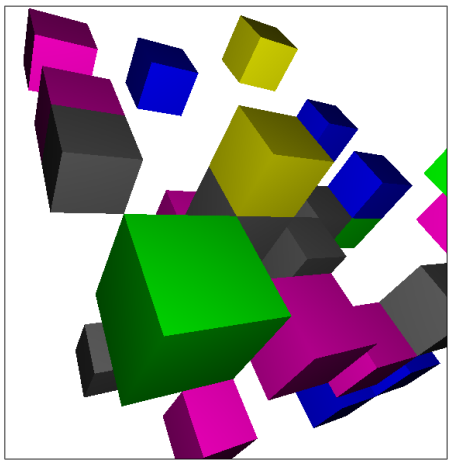


KUVIO 36. Diamond-Squarella luotuja maastoja

4.2 Demonstraation toteutus

4.2.1 Ensimmäinen askel

Ihan tyhjästä ei kuitenkaan tarvinnut aloittaa demonstraation toteuttamista. Työnohjaaja Jani Ylikangas antoi aloituspohjaksi tekemänsä pienimuotoisen raycasting- demonstraation, josta oli hyvä aloittaa tutustuminen aiheeseen. Kuviossa 37 on esitetty hänen tekemänsä demonstraatiota, jossa on kolmiulotteiseen taulukoon sijoitettuna erivärisiä vokseleita, jonka jälkeen ne ovat renderöity.



KUVIO 37. Jani Ylikankaan raycasting-demonstraatio

Aloitin tältä pohjalta suunnittelun ja ideoinnin siitä, miten toteuttaisin mahdollisen demonstraation. Yksi idea jonka työn ohjaaja antoi, oli kehittää vokselipohjainen kartta lukemalla väri- ja korkeuskartta kuvatiedostoista, joiden pohjalta se luodaan. Aloitin ensin tutkimalla ja testaamalla, miten raytracing toimii. Kokeilin erilaisia ideoita, joita tuli mieleen, saadakseni esille erilaisia mahdollisuuksia siihen mitä voisi demonstraatiossa käyttää. Käytin testissä Livionin logoa ja tein siitä edellä mainitut väri- ja korkeuskuvatiedostot joista selviävät korkeus- sekä väriarvot. Korkeuskartta esitetään mustavalkoisena, ja siinä vaaleampi kohta on korkeampi ja tummempi kohta matalampi. Kuviossa 38 on esitettynä testissä käytetyt väri- ja korkeuskuvatiedostot.

KUVIO 38. Käytetyt väri- ja korkeuskartat

Näiden pohjalta pystyin raytracingillä vokseleiden avulla renderöimään kuvan, jonka lopputulos on esitetty kuviossa 39 sekä ilman läpinäkyvyyttä ja sen kanssa. Kauempaa katsottuna sitä ei kovin helposti erota, että objekti on muodostettu neliön muotoisista kuutioista, kuitenkin mitä lähemmäksi menisi, sitä helpompaa olisi erottaa yksittäiset kuutiot.

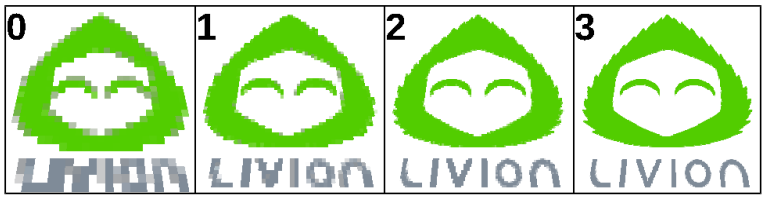
KUVIO 39. Kuvista renderöity kolmiulotteinen objekti

4.2.2 Vokseleiden optimointi

Edellä esitetyn renderöinnin jälkeen siirryin tutkimaan, miten sitä ja ruudun päivitysnopeutta voisi olla mahdollista parantaa. Edelliset olivat yksinkertaisuudestaan huolimatta raskaita varsinkin tosiaikaiseen renderöintiin. Kuvion mukaisen objektin renderöinnissä kestää kauan varsinkin hitaammalla prosessorilla, ja ruudun päivitys ei ole kuitenkaan nopeammallakaan prosessorilla niin, hyvä että liike olisi tarpeeksi sulavaa reaaliaikaiseen renderöintiin. Tässä tuli myös huomattua se asia, miten näytönohjain ei auta renderöinnissä ollenkaan ja prosessorin pitää hoitaa kaikki. Nykyisin on ainakin OpenCL:n (Open Computing Language) ja NVIDIAn Cuda:n avulla olisi mahdollista siirtää laskutoimituksia prosessorilta näytönohjaimelle ja näin nopeuttaa renderöintia, mutta nykyään mikään internetselain ei näitä tekniikoita pysty hyödyntämään. Khronos Groupilla, OpenGL:n, OpenCL:n, WebGL:n ja monien muiden kehittäjällä, on kuitenkin kehitteillä WebCL. Tämä mahdollistaisi OpenCL:n tarjoamien edut internetselainten käyttöön.

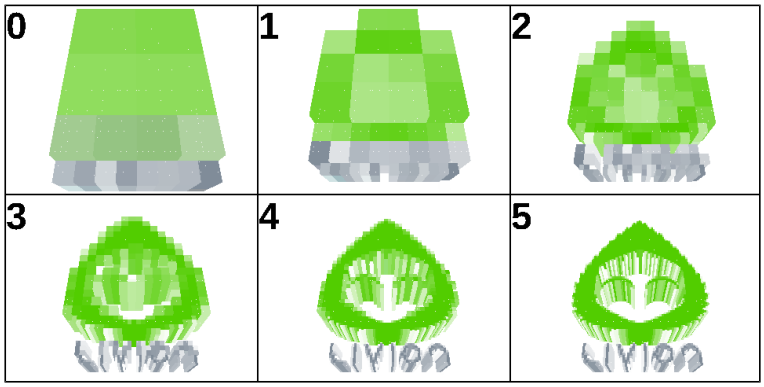
Aikaisemmin on mainittu, miten puurakenteellisen vokseliobjektin tarkkuutta pystyy säätämään rajoittamalla syvyyttä. Ennenaikainen etenemisen pysäyttäminen heikentää objektin tarkkuutta. Etuna rajoittamisella kuitenkin on ruudunpäivitystoiminnon kasvu. Koska algoritmi ei pääse etenemään syvimmälle tasolle asti, se myös saa tuloksen saavutettua nopeammin kuin jos se menisi aina niin syvälle kuin on mahdollista. Syvyyden rajoituksen ideana on, että jos esimerkiksi objekti on kaukana kamerasta, suurimmalla syvyydellä objektin yksityiskohtia ei olisi edes mahdollista erottaa. Tällöin voidaan rajoittaa syvyyttä ilman että laatu kärsii, ja samalla saadaan nopeutettua objektin piirtämistä. Kuviossa 40 on

SVO-puurakenteellinen vokseliobjekti, jonka syvyyttä on rajoitettu. Kuvion tarkkuushan heikkenee sitä enemmän, mitä aikaisemmin syvyysrajoitus tulee vastaan.



KUVIO 40. Puurakenteellisen objektin tarkkuuden rajoitus

Yksi mahdollisuus siihen miten SVO-puurakenteiden ja samalla mahdollisesti vokseleiden määrää yhdessä objektissa pystyy vähentämään, on tehdä siitä kooltansa niin iso kuin siihen kohtaan on mahdollista sopia. Aikaisemmin yhden puurakenteen koko on ollut aina 8 x 8 x 8 (leveys, korkeus ja syvyys). Kuitenkin on mahdollista, jos on tilaa, asettaa puurakenteen kooksi 16 x 16 x 16, 32 x 32 x 32 tai suurempikin kahden potenssissa olevalla koolla. Kuviossa 41 objekti on jaettu niin isoihin SVO-rakenteisiin kuin on mahdollista sopia. Nollatarkkuudella puurakenteiden koko erot huomaa selvästi: ylempänä on neljä 32 x 32 x 32 kokoista puurakennetta, niiden alla on neljä kooltaan 16 x 16 x 16 koolla olevaa ja niiden alla kuusitoista 8 x 8 x 8 kooltaan. Näistä päällimmäiset kahdeksan ovat tyhjiä eivätkä ole tämän vuoksi näkyvissä.



KUVIO 41. Muuttuvakokoiset vokselit SVO-puurakenteessa

Seuraavaksi aloin tutkia sitä, kuinka paljon vokseleiden määrä vähenee eri toteutustavoilla. Otin vertailuksi 8 x 8 x 8 kiinteäkokoisen rakenteen sekä muuttuvakokoisen rakenteen. Tutkin myös miten tyhjien vokseleiden pois jättäminen vaikuttaa määrään. Vertailuun otin mukaan kaksi eri kuvakokoa samasta kuvasta. Taulukossa 2 on pienemmän kuvan tarvitta-

vien vokseleiden määrät edellä mainituilla toteutustavoilla, ja taulukossa 3 on isomman kuvakoon vaatimat vokselimäärät. Taulukoista ilmenee miten objektin esittämiseen pienellä tarkkuudella muuttuvakokoisena tarvitaan huomattavasti vähemmän vokseleita, myös tyhjien kohtien pois jättäminen on kannattavaa alemman vokselimäärän saavuttamiseksi. Suurimmalla tarkkuudella taas ei ole mitään merkitystä sillä, onko objekti muuttuvakokoinen vai kiinteä, koska sen esittämiseen vaaditaan sama määrä vokseleita. Muuttuvakokoisella on enemmän syvyystasoja tietenkin suuremman puurakenteen vuoksi.

TAULUKKO 2. Pienemmän objektin vokselimäärät

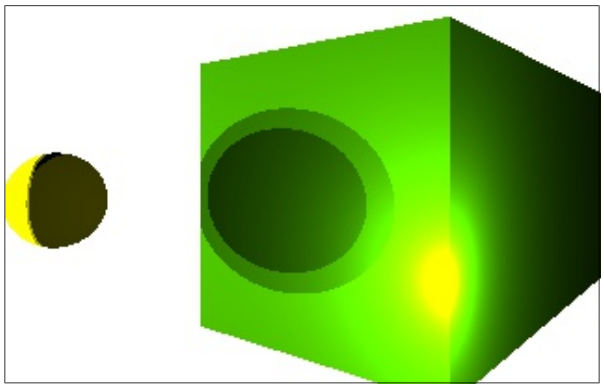
Kuvakoko	64 x 88 x 32		
Syvyystaso:		Perinteinen:	Muuttuvakokoinen:
0	Tyhjien kohtien kanssa	352	44
	Ilman tyhjiä kohtia	352	44
1	Tyhjien kohtien kanssa	2816	352
	Ilman tyhjiä kohtia	1934	296
2	Tyhjien kohtien kanssa	22528	2816
	Ilman tyhjiä kohtia	10061	1290
3	Tyhjien kohtien kanssa	180224	22528
	Ilman tyhjiä kohtia	56940	6344
4	Tyhjien kohtien kanssa	-	65536
	Ilman tyhjiä kohtia	-	22129
5	Tyhjien kohtien kanssa	-	180224
	Ilman tyhjiä kohtia	-	56779

TAULUKKO 3. Suuremman objektin vokselimäärät

Kuvakoko	200 x 256 x 32		
Syvyystaso:		Perinteinen:	Muuttuvakokoinen:
0	Tyhjien kohtien kanssa	3200	176
	Ilman tyhjiä kohtia	3200	176
1	Tyhjien kohtien kanssa	25600	1408
	Ilman tyhjiä kohtia	14652	820
2	Tyhjien kohtien kanssa	204800	11264
	Ilman tyhjiä kohtia	69413	3172
3	Tyhjien kohtien kanssa	1638400	90112
	Ilman tyhjiä kohtia	425265	14827
4	Tyhjien kohtien kanssa	-	262114
	Ilman tyhjiä kohtia	-	69000
5	Tyhjien kohtien kanssa	-	1638400
	Ilman tyhjiä kohtia	-	421849

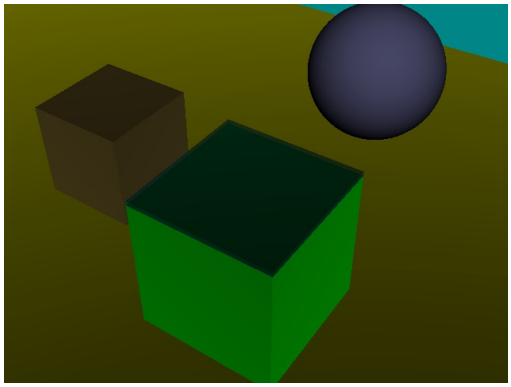
4.2.3 Valaistuksen lisäys

Työn ohjaaja oli erittäin kiinnostunut näkemään, miten valaistus toteutetaan ja minkälainen sen tarkkuus on. Otin tämän seuraavaksi askeleeksi. Valaistuksen toimintaidea on esitelty luvussa 3.4. Säteen törmäyspisteestä lähetetään säde kohti jokaista olemassa olevaa valoa kohti. Jos niiden välissä on jokin objekti, kyseinen kohta on varjossa, muuten lasketaan sille valaistusarvo. Kuviossa 42 on esitetty testi, jossa valaistuksena on kolme pistemäistä valonlähdettä, jotka lähettävät valoa jokaiseen suuntaan samalla tavalla kuin esimerkiksi aurinko. Itse valonlähteet eivät ole kuviossa näkyvissä. Yksi on vihreän suorakulmion lähellä, mikä ilmenee kirkkaana pisteenä, ja kaksi muuta ovat vasemmalla näkyvän pallon takana niin että ne luovat varjon suorakulmion pinnalle. Toinen valoista on vähän kauempana kuin toinen, minkä vuoksi molemmat luovat oman pyöreän muotoisen varjonsa.



KUVIO 42. Valaistu yksinkertainen testiskene

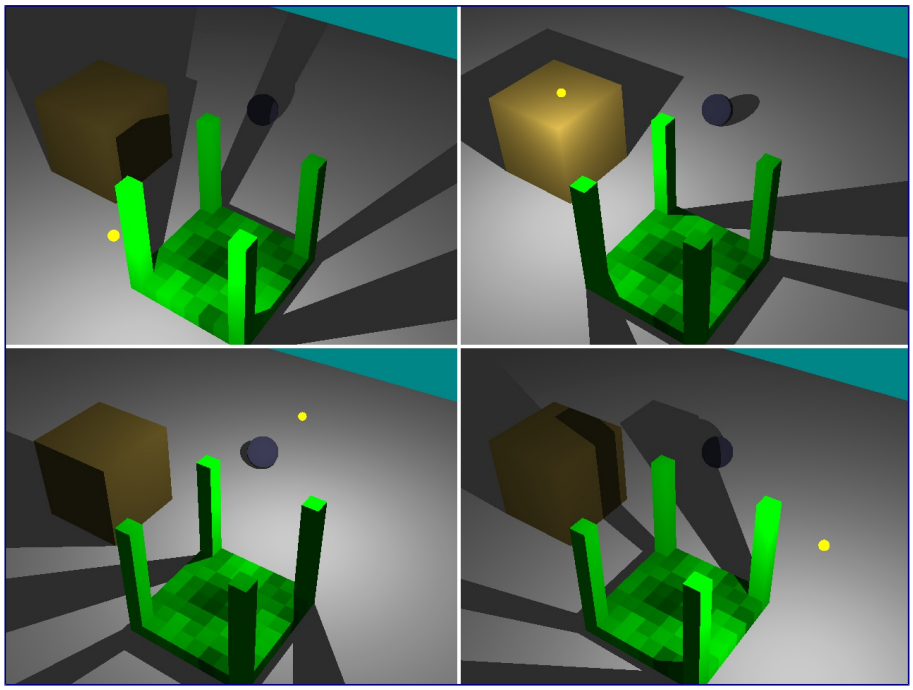
Seuraavaksi lähdin muodostamaan yksinkertaista skeneä, jota voisi käyttää eri ideoiden testaamiseen ennen lopullisen maaston luomista. Kuviossa 43 on esitetty muodostettu skene. Ruskea objekti on yksinkertainen neliö, ja vihreä objekti on vokseleita sisältävä objekti jonka tarkkuus on huonoimmalla tasolla, joten se muistuttaa neliötä. Keltasävyinen objekti on suorakulmainen objekti, jonka pituus ja leveys ovat isoja, mutta korkeus pieni muodostaen näin litteän tason. Sininen objekti on myös yksinkertainen pallo.



KUVIO 43. Testiskenen objektit

Testialustan muodostamisen jälkeen aloitin valaistuksen tutkimisen ja toteuttamisen. Valaistus kehittyi toiminnaltaan monipuolisemmaksi verrattuna aikaisemmin tehtyyn valaistustestiin. Kuviossa 44 on näkyvissä sama skene kuin aikaisemmassa kuviossa. Pallon kokoa on pienennetty ja tason väri on muutettu hopean väriseksi. Tämä värimuutos mahdollistaa valon sijoittamisen skeneen näkyväksi, ja koska se toimii samalla tavalla kuin aurinko, on sen väriksi valittu keltainen. Kuviossa on samasta skenestä neljä eri kuvaa, ja valon paikka vaihtelee yhdeksänkymmenen asteen välein. Tämän avulla saadaan visuaalisesti

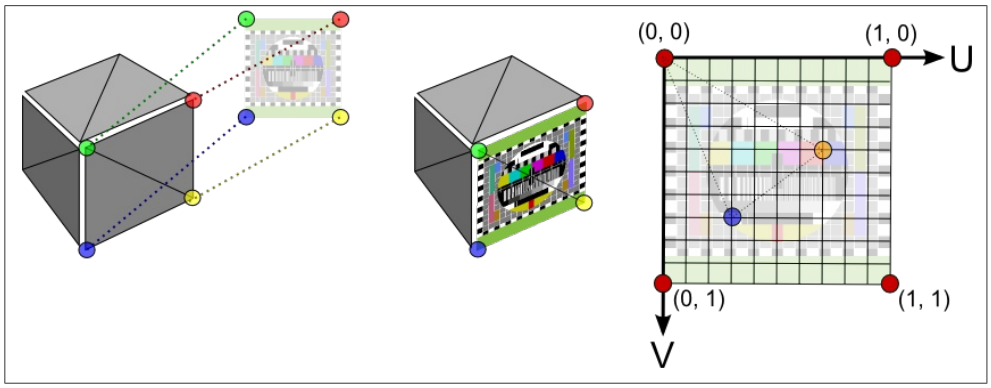
näkyville, miten valaistus ja sen luomat varjot käyttäytyvät valonlähteen siirtyessä eri paikkaan.



KUVIO 44. Valaistus ja varjot visuaalisesti esitettynä

4.2.4 Objektien tekstuuripinnoitus

Seuraavaksi mielenkiinnosta aloin tutkia, miten pintojen päälle voisi asettaa tekstuuripinnoitteen. Tekstuuripintojen asettaminen objekteille toteutetaan UV-kartoituksen avulla. UV-kartoitus on tekniikka, jota käytetään kuvan ”maalaamiseen” objektin päälle. Yleisin tiedostoformaatti tekstuuribittikarttojen tallentamiseen on PNG, joka tukee häviämätöntä pakkausmuotoa ja tärkeänä ominaisuutena alpha-kanavaa eli läpinäkyvyyttä. Tekstuuripintojen kartoitus objekteille on suoraviivainen prosessi. Säteen osuessa objektiin lasketaan siitä UV-koordinaatit, joiden avulla tekstuurista saadaan piirrettyä ruudulle oikea kohta. Kuviossa 45 on esitetty, miten tekstuuri asetetaan pinnalle. Esimerkki kuva on tehty OpenGL:n käyttämille polygoneille, mutta sama idea toimii suoraan raytracingille. UV-koordinaateista U:lla esitetään leveyssuuntaa ja V:llä korkeussuuntaa, ja niiden arvot ovat aina väliltä 0–1. (Xojo3D 2013.)



KUVIO 45. Tekstuuripinnoituksen UV-koordinaatit (Xojo3D 2013.)

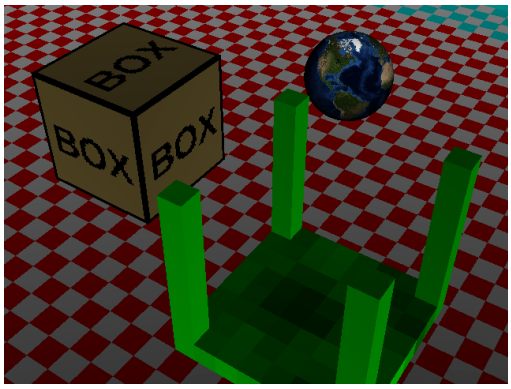
Tekstuurin asettaminen pallon pinnalle ei eroa suuresti edellisestä suorakulmaisen tai neliömäisen objektin UV-kartoituksesta. Pallon UV-kartoitusta varten lasketaan normaalivektoriksi, joka on kohtisuorassa pintaan nähden. (Morcillo 2012.) Normaalivektorin avulla saadaan seuraavien kaavojen avulla laskettua UV-koordinaatit, joiden avulla tekstuurin pystyy kiertämään pallon ympäri. Ensimmäinen kaava on alkuperäinen, ja jälkimmäinen on toteutettu JavaScript-ohjelmointikielellä. Kuviossa 46 on esitetty testiskene, jonka objekteille on lisätty tekstuuripinnoitus. Kaavalla muodostettu lopputulos ei toimi optimaalisesti pallon napojen kohdalla, vaan tämä vaatisi omat lisäykset ja kompensaatiot. Siitä huolimatta asiansa kohtuullisen hyvin.

$$U = 0{,}5 + \frac{2 \times \arctan\left(\frac{y}{\sqrt{\left(x^2 + y^2\right)} + x}\right)}{2 \times \pi}$$

$$V = 0{,}5 + \frac{\arcsin(z)}{\pi}$$

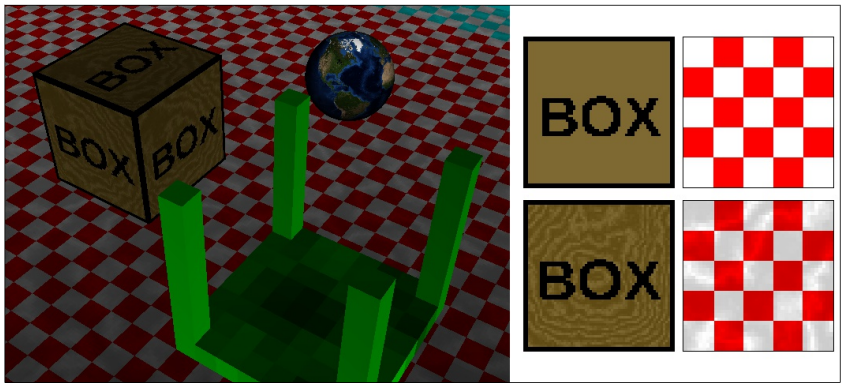
U = 0.5 + Math.atan2(normaali.x, normaali.y) / (2 * Math.PI);

V = 0.5 + Math.asin(normaali.z) / Math.PI;



KUVIO 46. Testiskenen objektien teksturointi

Yksinkertainen tekstuuripinnoite on kuitenkin aika staattisen ja elottoman näköinen. Jani Ylikangas ehdotti Perlin Noise kohinan lisäämistä tekstuuriin, ja tätä on mahdollista hyödyntää erilaisten luonnollisten kuvioiden muodossa. Perlin Noisen kohinan voi tehdä täysin erilliseksi tekstuurista, ja näin ne voidaan renderöinnin yhteydessä yhdistää tai sitten, kuten itse tein, yhdistää kohinan suoraan tekstuuriin sen jälkeen, kun se on ladattu tiedostosta. Kuviossa 47 on esitetty sama tekstuuripinnoitettu skene kuin edellisessä kuviossa, lisänä on tekstuureille lisätty Perlin Noisen kohina. Kuvion oikeassa laidassa on ylempänä neliön alkuperäinen tekstuuri ja sama sen jälkeen, kun siihen on lisätty Perlin Noisen kohina. Lopputulos on paljon elävämmän näköinen tekstuuripinnoite alkuperäiseen verrattuna

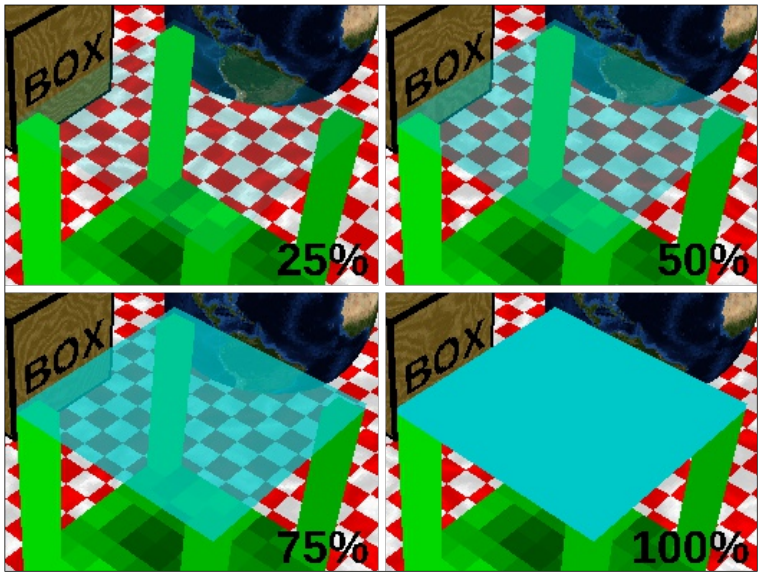


KUVIO 47. Perlin Noisen lisäys tekstuuripinnoille

4.2.5 Viimeiset lisäykset

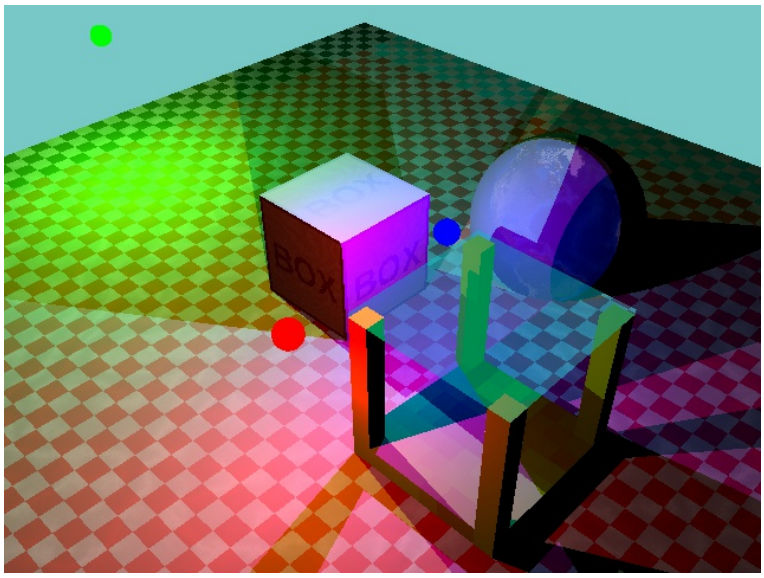
Useaan otteeseen mietin miten, läpinäkyvyys olisi mahdollista toteuttaa. Toteutin sen mahdollisimman yksinkertaisesti. Säteen osuessa läpinäkyvään objektiin, sen väri otetaan tal-

teen, ja sen jälkeen säde jatkaa matkaa eteenpäin. Minkäänlaista aitoa säteen taittumista tai hajaantumista ei lasketa, sillä nämä vaatisivat omat säteensä jotka eivät kuulu työhön rajattuihin säteisiin. Kuviossa 48 on esitetty testiskene johon on lisätty läpinäkyvä objekti, erilaisilla läpinäkyvyysarvoilla vokseliobjektin päälle, 100 % tarkoittaa täysin kiinteää objektia ja vastaavasti nolla täysin läpinäkyvää objektia, jota ei edes näe ollenkaan.



KUVIO 48. Läpinäkyvä objekti

Testiskenen viimeisenä ominaisuutena laajensin valonlähdettä jakamalla väriarvot yhteisestä kanavasta erillisiksi punaisiksi, vihreiksi ja sinisiksi kanaviksi, sillä näin valonlähteen ympärilleen säteilemän valon väriä pystyy muuttamaan. Aikaisemmin valonlähde säteili aina mustavalkoisen väriskaalan mukaista väriä. Tämä mahdollistaa monipuolisemman valaistuksen, koska valo voi olla jonkin muunkin värinen kuin valkoinen. Kuviossa 49 on testiskeneen lisätty kolme valon, lähdettä joista jokaisen väriksi on määritelty jokin RGB-värimaailman pääväri.



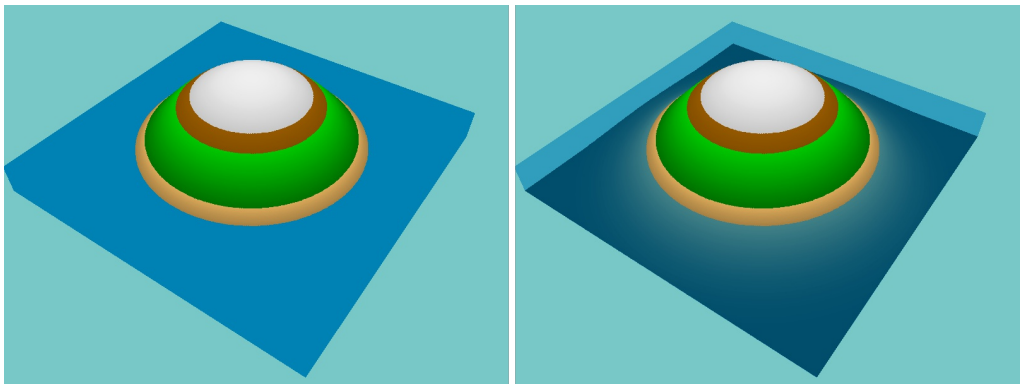
KUVIO 49. Eriväriset valonlähteet

4.3 Maastoympäristö

Edellisen testiskenen omaisuuksien kehittämisen jälkeen siirryin suunnittelemaan ja toteuttamaan maastoskenaariota. Päädyin kohtuullisen yksinkertaiseen pyöreän muotoiseen saareen, jossa olisi puita. Maaston muodostamisessa on käytetty kolmea eri tapaa: korkeuskartan (värikartta ei ole pakollinen, vaan sen voi muodostaa ohjelmallisesti korkeuskartan pohjalta), Diamond-Square algoritmin tai matemaattisen kaavan avulla muodostetut kartat.

4.3.1 Maasto kuvasta

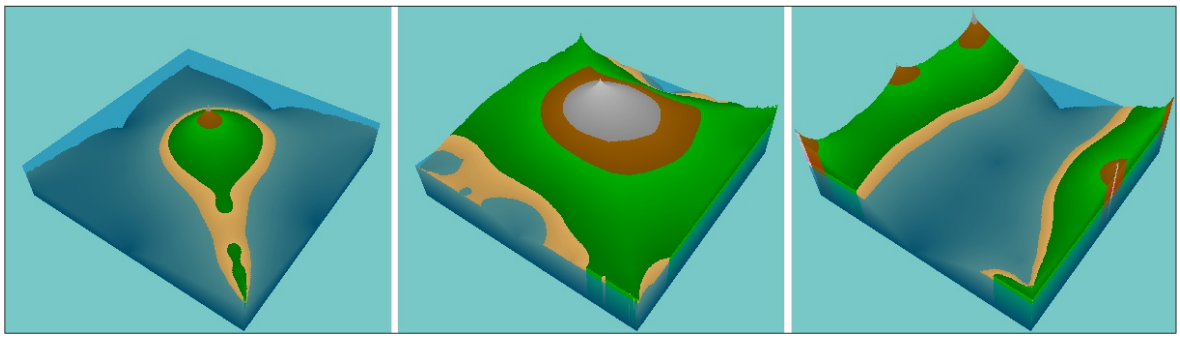
Ensimmäisen maaston loin käyttämällä korkeus- ja värikarttoja. Toteutin myös variaation, jossa värikartta luodaan ennalta määritettyjen parametrien perusteella, jos värikarttaa ei anneta ollenkaan. Kuviossa 50 on esitetty veden keskellä oleva saarimaasto. Ensimmäisessä läpinäkyvyys on kytketty pois päältä, ja toisessa läpinäkyvyys on päällä mahdollistaen veden pinnan alle näkemisen. Vesi on aikaisemmin esitelty läpinäkyvä suorakulmainen sinertävän värinen objekti, jonka korkeus on asetettu määriteltyyn vesitasoon.



KUVIO 50. Korkeuskartasta muodostettu saarimaasto

4.3.2 Sattumanvarainen maasto

Sattumanvaraisessa kartassa on tietenkin etuna se, ettei tarvitse piirtää tai käyttää kuvia, joiden pohjalta kartta muodostetaan, tai sitten, kuten seuraavaksi esitellyssä matemaattisessa maastossa, kaavojen muodostamista. Algoritmille voidaan antaa parametreja esimerkiksi siitä kuinka jyrkät maaston korkeuserot ovat. Kuviossa 51 on esitetty kolme eri algoritmin avulla luotua maastoa. Maastot on luotu Diamond-Square-algoritmilla ja jokaisella kerralla algoritmille on annettu samat parametrit, joiden pohjalta se luo hyvinkin erinäköisen maaston.



KUVIO 51. Sattumanvaraisesti luotu maasto

4.3.3 Matemaattinen maasto

Matemaattinen kartta on funktio, jolle annetaan pituus-, leveys- ja korkeuskoordinaatit. Karttaa ei ole tallennettu muistiin ollenkaan, ja kahden ensimmäisen koordinaatin avulla

funktio laskee yhdellä tai useammalla ennalta määritetyllä kaavalla korkeuskoordinaatin. Laskettua korkeutta verrataan funktiolle annettuun korkeusarvoon, ja se palauttaa totuusarvon sille, onko annettu arvo yli kaavalla lasketun arvon, jos on säde ei törmää maastoon, muussa tapauksessa säde törmää maastoon. Seuraavassa on esitetty maaston muodostamisessa käytetyt kaavat.

$$X_{kaava} = \cos\left(X_{annettu} - \frac{30}{115}\right) \times 125$$

$$Y_{kaava} = \cos\left(Y_{annettu} + \frac{340}{115}\right) \times 125$$

$$Z_{kaava} = Z_{vakiokorkeus} - X_{kaava} + Y_{kaava}$$

```
var Xkaava = Math.cos((sijainti.x - 30) / 115) * 125;
var Ykaava = Math.cos((sijainti .y + 340) / 115) * 125;
var Zkaava = perusZ - Xkaava + Ykaava;
```

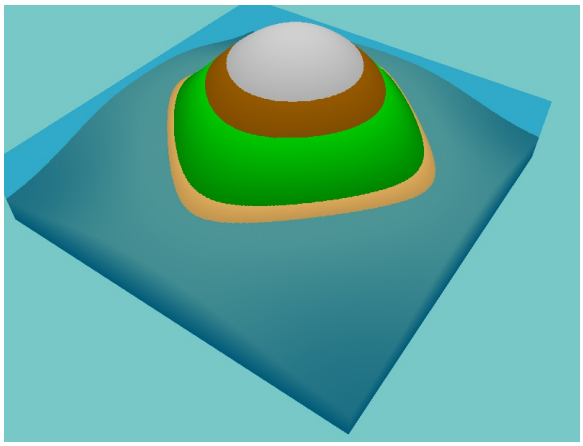
Jos säteen nykyinen sijainnin korkeus on alle määritetyn vesitason, lasketaan korkeusarvo eri kaavalla, eli on seuraavaksi esitetyllä kaavalla, jonka avulla saadaan erilainen pinta. Sen mukaan, kummalla kaavalla Z arvo lasketaan, sitä verrataan funktiolle annetun sijainnin Z-arvoon säteen törmäyksen määritykseen. Kuviossa 52 on esitetty maasto, joka on luotu esitettyjen kaavojen avulla.

$$Vektori_{kartankeskipiste}: X = \frac{leveys}{2} + 12,\ Y = \frac{korkeus}{3} + 2,\ Z = Vesitaso$$

$$etäisyys = Vektori_{kartankeskipiste} \text{ ja } Vektori_{annettu} \text{ välinen etäisyys}$$

$$Z_{kaava} = Vesitaso - \frac{etäisyys}{3} + 70$$

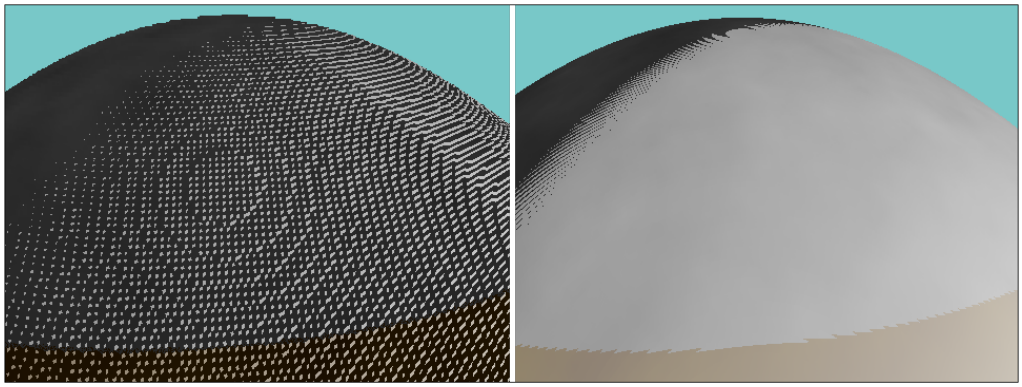
```
var keksipiste = new Vec3D(this.koko.x / 2 + 12, this.koko.y / 2 + 3, this.vesitaso);
var etaisyys = laskeEtaisyys(keskipiste, sijainti);
var Zkaava = this.vesitaso – (etäisyys / 3) + 70;
```



KUVIO 52. Matemaattisesti muodostettu saarimaasto

4.3.4 Maaston renderöintityylit

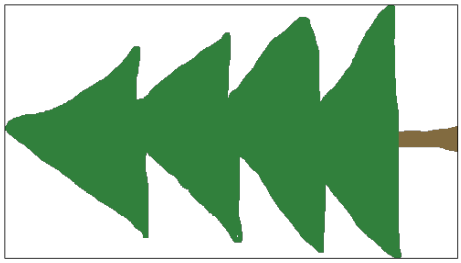
Alun perin maasto oli mahdollista renderöidä pelkästään vokselityylisinä neliöinä, mutta Jani Ylikangas ehdotti vaihtoehtoista sileämpää tapaa renderöintiin. Lisäsin siis nykyisen renderöinnin rinnalle sileämmän tavan renderöidä maasto. Kuviossa 53 on esitetty edellisen kuvion maaston vuoren huippu, joka on renderöity molemmilla tavoilla. Ero tyylien välillä on huomattava: vasemmanpuoleinen vokselityyli on huomattavasti kanttisemman ja neliömäisemmän näköinen verrattuna oikeanpuoleisen sileämpään tyyliin. Sileämpi renderöinti toimii tällä hetkellä pelkästään matemaattisen maaston kanssa, muilla kartanluontitavoilla on käytössä pelkästään vokselityylinen renderöinti.



KUVIO 53. Eri renderöintitavat

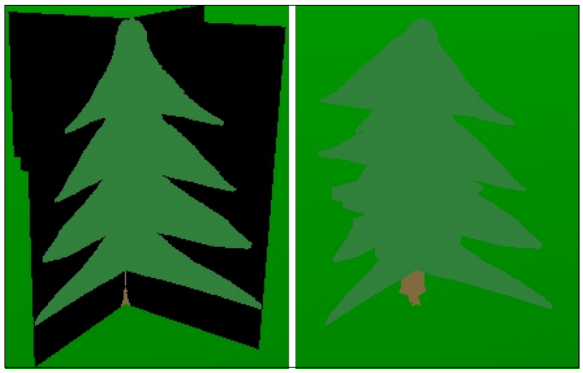
4.3.5 Maastoon lisätyt yksityiskohdat

Viimeisenä lisäyksenä demonstraatioon lisäsin puita elävöittämään aution näköistä maastoa. Puun muodostamisen toteutin mahdollisimman yksinkertaisesti, joten puu esitetään sprite-tyylisenä monimutkaisemman kolmeulotteisen mallin sijasta. Muodostettu puuobjekti ei kuitenkaan ole täysin kaksiulotteinen, vaikka näyttäisikin ensi näkemältä siltä, sen sijaan puu esitetään kolmiulotteisena objektina todella pienellä syvyysarvolla, joka saa sen näyttämään litteältä. Kuviossa 54 on esitetty tekstuuripinta, jota käytetään puun renderöinnissä.



KUVIO 54. Puun tekstuuripinta

Yleensä kolmiulotteiseen skeneen sijoitetut sprite-objektit olisivat aina kääntyneenä kohti kameraa päin, kuten esimerkiksi kaksiulotteisella raycastingilla tehdyissä ohjelmissa on. Muodostettu puuobjekti ei kuitenkaan ole oikeasti kaksiulotteinen, vaan litteä kolmiulotteinen objekti, se ei käänny kohti kameraa joten se näyttää oudolta. Puusta voidaan tehdä enemmän kolmiulotteisen näköinen sijoittamalla samaan kohtaan kaksi samanlaista puuobjektia ristikkäin toisiinsa nähden keskikohtien mennessä toistensa päälle. Kuviossa 55 on esitetty edellä mainittu kahden puun risteytys. Vasemmanpuoleisessa puu on renderöity ilman läpinäkyvyyttä, mikä saa läpinäkyvät kohdat näkymään mustina, ja oikeanpuoleisessa läpinäkyvyys on otettu käyttöön.

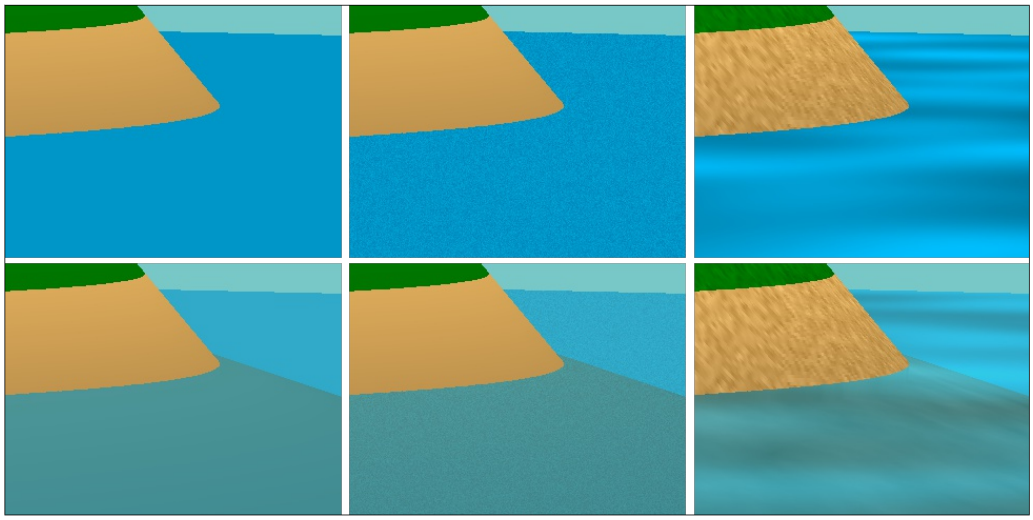


KUVIO 55. Maastoon renderöity puu

Puun ja maaston ulkonäkö on kuitenkin vieläkin litteän ja elottoman näköinen. Ulkonäköä ja realistisuutta voidaan kuitenkin parantaa lisäämällä pinnoille, mitä aikaisemmin mainitun Perlin Noisen kohinan avulla. Kuviossa 56 on esitetty molemmilla renderöintityyleillä pinnoille lisätty Perlin Noise. Puu on esitetty ilman valaistusta ja sen kanssa, jonka seurauksena puu tuottaa valon lähteen sijainnin mukaan aidon näköisen varjon.

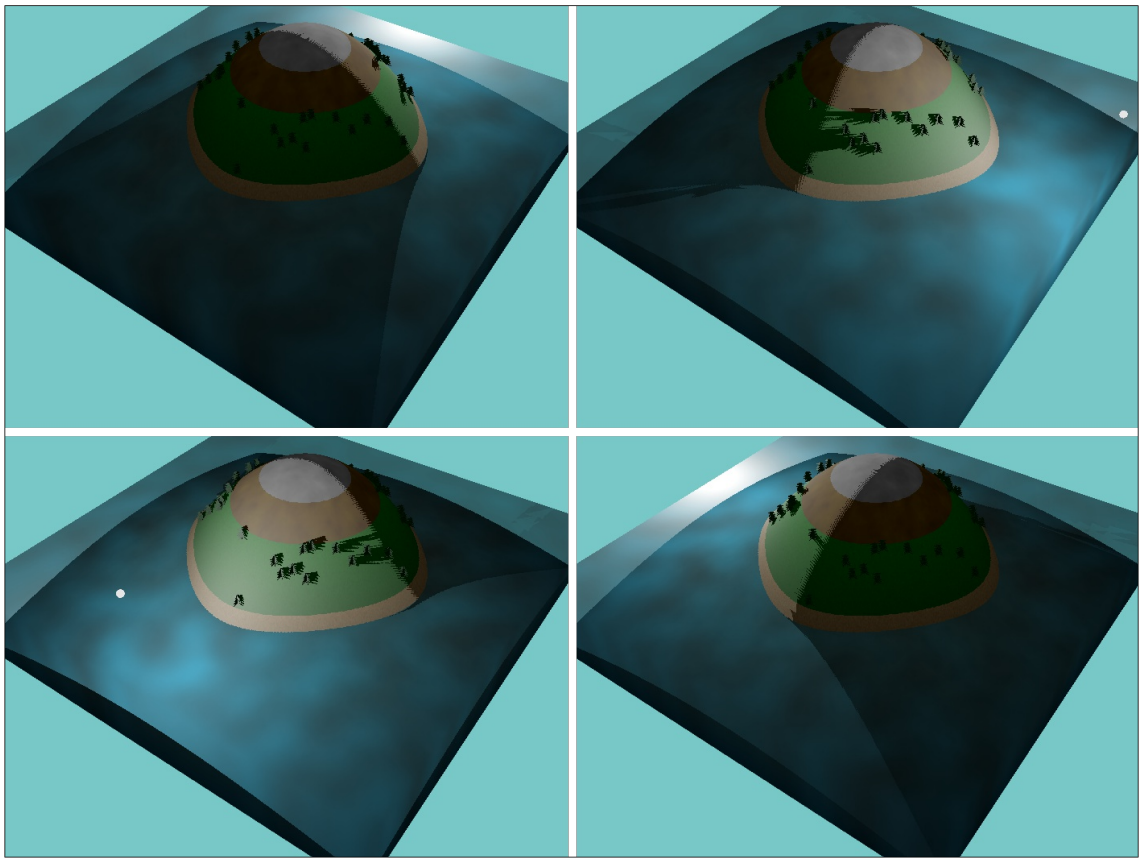
KUVIO 56. Perlin Noisen lisäys ja eri renderöinnit

Veden pinta on myös täysin staattisen näköinen eikä tämän vuoksi näytä kovinkaan realistiselta. Ulkonäön parantamiseksi lisäsin animaation veden pinnalle kahdella eri tavalla: yksinkertaisen satunnaisen muutoksen veden väriarvoon sekä Perlin Noisea hyödyntävän version. Molemmat tavat parantavat veden pinnan ulkonäköä, varsinkin Perlin Noise. Kuviossa 57 on esitetty veden pinta ilman mitään lisäefektiä, satunnaisen väriarvon muutoksen sekä Perlin Noisen avulla toimiva animaatio. Perlin Noisella toteutettu veden pinta on selvästi realistisempi ja näyttävämpi, ja satunnainen väriarvon vaihtelu parantaa myös jonkin verran veden pinnan. Tosiaikaisessa renderöinnissä animaatioiden toiminta olisi näkyvissä selvemmin kuin yksittäisissä kuvankaappauksissa.



KUVIO 57. Veden pinnan animaatioefektit

Näiden kaikkien ominaisuuksien ja ideoiden toteuttamisen jälkeen sain valmiiksi viimeisen version demonstraatioksi tarkoitetusta maastosta. Toimivuus on pieniä bugeja lukuun ottamatta hyvä. Valaistuksen kanssa on pienimuotoisia ongelmia, joita esiintyy esimerkiksi sileän maaston renderöinnissä kuviossa 46, varjon ja valaistun osan välissä esiintyy raja-arvo ongelmia. Kuviossa 58 on esitetty viimeinen versio demonstraatioksi tarkoitetusta maastosta valaistuksen ollessa jokaisessa kuvassa eri puolella maastoa. Vedenpinnan valkoinen osa ei ole tässä versiossa virhe, vaan valon säde ei reagoi oikealla tavalla läpinäkyvyyden kanssa, mikä johtuu yksinkertaisesta käsittelytavasta.



KUVIO 58. Viimeinen versio demonstraatiosta valaistuna

5 TULOKSET JA POHDINTA

Opinnäytetyön tarkoituksena oli tutkia ja selvittää miten, raytracing toimii internetselaimella ja minkälaista suorituskykyä olisi odotettavissa. Työssä käytiin läpi sitä, miten raytracing ja sen alaluokaksi luokiteltu raycasting toimivat sekä kaksi- että kolmiulotteisina. Lopputuloksena kehitettiin demonstraationa maasto ja pohja, jota voi mahdollisesti käyttää apuna tulevaisuudessa.

Opinnäytetyötä aloittaessani itselläni ei ollut oikein yhtään kokemusta kolmiulotteisen grafiikan tuottamisesta, puhumattakaan raytracingista, josta en myöskään tiennyt paljon mitään ennen aloitusta. Tekniikka kuitenkin näytti ja kuulosti sen verran mielenkiintoiselta, että otin aiheen tutkittavaksi. Lähdemateriaalia aiheesta en harmillisesti löytänyt mistään keskitetystä paikasta, ja kirjallisuutta aiheesta ei myöskään ole paljoa. Raytracingiä on kuitenkin mahdollista käyttää ammattilaistason 3D-mallinnus- ja kuvankäsittelyohjelmissa, kuten Adobe After Effects CS6:ssa. NVIDIA on tehnyt ja kehittää vieläkin eteenpäin raytracingille omaa Optix-rajapintaa, jonka avulla on mahdollista tehdä todella näyttävän näköisiä maisemia. Rajapintaa en kuitenkaan tutkinut syvemmin. Monien grafiikkaohjelmoijien mielestä polygonipohjaisten objektien renderöinti on mennyt nykyään todella monimutkaiseksi, ja osa on myös alkanut tutkia, olisiko raytracingin käyttäminen yksinkertaisempaa.

Nykyään myös vokselit ovat ruvenneet nostamaan suosiotaan. Yksi syy tähän on niiden helpompi käsittely polygoneihin verrattuna. Videopeleissä esimerkiksi hajoavat maastot on paljon helpompi toteuttaa käyttämällä vokseleita. Vokseleiden avulla on mahdollista toteuttaa objektit ja maastot dynaamisempina ja monipuolisempina kuin polygoneilla tehtyihin. Olen lukenut monista lähteistä, että polygoneilla toteutettaessa niiden manipulointi on huomattavasti vaikeampaa kuin vokseleiden manipulointi.

Omasta mielestäni raytracingin toteuttaminen ei tuntunut mitenkään hirvittävän vaikealta, kunhan vain pääsin ensiksi sisälle asiaan. Ennen opinnäytetyön aloitusta kuitenkin tiesin että tekniikan suorituskyky on todella hidasta, ja tämä asia tuli huomattua demonstraation kehityksen aikana. Demonstraatiota ajettiin tietokoneella, joka oli varustettu neljännen sukupolven Haswell-pohjaisella Intel i7:lla. Prosessori oli uusin ja nopein, mikä oli saatavil-

la. Nopeasta prosessorista huolimatta ruudunpäivitysnopeus jäi pienelläkin 320 x 240 resoluutiolla matalaksi. Koska tein toteutusta ensimmäistä kertaa tekniikalla, olen varma, että kokemuksen parantuessa joitain kohtia koodissa voisi varmasti optimoida tai tehdä paremmin, ja se parantaisi nopeutta. Jani Ylikangas on itse ollut myös vahvasti sitä mieltä, että nopeutus ei tulisi kuitenkaan tulisi olemaan mitenkään hirvittävän iso, tekniikka vain on liian raskas prosessorille, vaikka laskentatehot ovatkin nousseet vuosi vuodelta. Janin mielestä yhtenä ongelmana raskaudelle on myös jatkuva resoluution kasvu, joka tuo lisää raskautta pikseli pikseliltä, koska seurattavien säteiden määrä kasvaa aina jokaista lisättyä pikseliä kohden.

Yksi mielenkiintoiselta kuulostava moduuli JavaScriptiin on Mozillan kehittämä asm.js. Moduuli rajoittaa JavaScriptin toimintaa yksinkertaistamalla sitä samalle tasolle kuin C ja C++. Testitulosten lukemisen perusteella JavaScriptin suorituskyky paranee huomattavasti. Nopeus ei kuitenkaan ole tällä hetkellä vielä samalla tasolla kuin natiivilla C- tai C++-kielillä tehdyissä ohjelmissa. Siitä huolimatta asm.js näyttää mielenkiintoiselta, ja sen demonstraationa on jopa videopelien laajasti käytetty Unreal Engine 3 on sen avulla saatu toimimaan internetselaimella hyvällä nopeudella. Tulos on vaikuttava, nopeus näyttäisi parantuvan jatkuvasti, ja asm.js:stä voi yhtä hyvin tulla suurikin tekijä tulevaisuudessa.

Opinnäytetyön aloitusvaiheessa asetettiin kysymyksiksi, miten raytracing soveltuu internetselaimelle, minkälainen sen suorituskyky olisi ja miten se soveltuisi mobiililaitteille, kuten iPadille tai Androidille. Internetselainten nopeus tuntuu olevan omasta mielestäni ihan hyvällä tasolla, ja selainten kehittäjät lupaavat jatkuvasti kehittää JavaScriptin ajonopeutta paremmaksi niin hyvin kuin osaavat. Tämän puolesta selaimille suunnatut sovellukset ovat lupaavan tuntuisia. HTML5-standardin myös valmistuessa eri alustojen selainten kautta sovellukset toimisivat samalla koodilla samalla tavalla oli alla minkälainen käyttöjärjestelmä vain. Tällä hetkellä standardi on vielä kehityksessä ja eroja löytyy eri selainten implementaatiosta sen verran, ettei ilman selain- tai alustakohtaisia viilauksia vielä pärjää. Mobiililaitteille raytracing sopii lähinnä yksittäisten kuvien renderöintiin, tosiaikaiseen renderöintiin ei ole nykyisessä tilanteessa kovin hyviä mahdollisuuksia. Sama tilanne on kuitenkin pöytäkoneidenkin kanssa, joten tilanne ei ole paljon parempi. Tilanne voisi olla parempi jos näytönohjainpiiri voisi avustaa raytracingin säteiden laskemisessa. Nykyisin se on kuitenkin optimoitu ja suunniteltu polygonien laskemista varten, ja tämän vuoksi prosessori joutuu yksin tekemään myös renderöinnin. WebCL voisi olla myös yksi tekijä, joka paran-

taisi nopeutta, koska laskentakuormaa voisi sen avulla siirtää näytönohjaimelle. Se on kuitenkin tällä hetkellä vielä kehitysvaiheessa.

Opinnäytetyön kirjoitusvaiheessa yksi vaikeimmista asioista oli erilaisten termien suomenkielisten vastineiden löytäminen, ja osalle ei tuntunut edes olevan mitään vastinetta. Demonstraation toteutuksen aikana tuli paljon erilaisia ideoita siihen, mitä olisin halunnut kokeilla. Ne kuitenkin piti yrittää mahdollisimman nopeasti siirtää syrjään, tai työstä olisi tullut liian laaja, joten se olisi mennyt liian pitkälle. Tämä oli suuri haaste, sillä jos aihe on mielenkiintoinen, siihen haluaisi ottaa paljon enemmän materiaalia mukaan kuin alkuperäinen idea sisälsi, nytkin mukaan on tullut muutama asia, jotka eivät kuuluneet alkuperäiseen visioon.

Raytracing ja vokselit voivat hyvinkin olla tulevaisuuden asia. Molemmat tuntuvat kohtuullisen yksinkertaisilta ymmärrettäviltä, varsinkin jos rautapuolen tuki näytönohjaimen puolelta paranisi internetselainten kohdalla. Itselle ainakin olisi mielekästä jatkaa toteuttamista eteenpäin parantamalla sekä lisäämällä mielenkiintoisia ideoita, mahdollisesti niitä, jotka piti jättää sivuun kokonaan opinnäytetyön toteutuksen aikana.

LÄHTEET

3D Realms. 2013. Wolfenstein. Www-dokumentti. Saatavissa: http://www.3drealms.com/wolf3d/. Luettu 23.10.2013.

Abi-Chahla, F. 2009a. When Will Ray Tracing Replace Rasterization. Www-dokumentti. Saatavissa: http://www.tomshardware.com/reviews/ray-tracing-rasterization,2351.html. Luettu 5.3.2014.

Abi-Chahla, F. 2009b. Next-Gen 3D Rendering Technology: Voxel Ray Casting. Www-dokumentti. Saatavissa: http://www.tomshardware.com/reviews/voxel-ray-casting,2423-4.html. Luettu 16.12.2013.

Accomazzi, V. 2011. Voxel-Based Graphics on Intel Architectures. Www-dokumentti. Saatavissa: http://software.intel.com/en-us/articles/voxel-based-graphics-on-intel-architectures. Luettu 9.12.2013.

Anteru. 2008. Voxels, sparse octrees, virtualization. Www-dokumentti. Saatavissa: https://anteru.net/2008/07/25/242/. Luettu 10.12.2013.

Bailey, M., Glassner, A. & Lathrop, O. 1999. Introduction to Computer Graphics. Pdf-dokumentti. Saatavilla: http://www.inf.ed.ac.uk/teaching/courses/cg/Web/intro_graphics.pdf. Luettu 11.3.2014.

Bilderzucht-blog. 2010. 3D Pixel / Voxel. Www-dokumentti. Saatavissa: http://www.bilderzucht.de/blog/3d-pixel-voxel/. Luettu 20.1.2014.

Boaz. L. 2008. Mental Ray for Maya, 3ds Max and XSI : A 3D Artist's Guide to Rendering. Wiley.

Buck, J. 2000. The Recursive Ray Tracing Algorith. Www-dokumentti. Saatavissa: http://reocities.com/SiliconValley/haven/5114/raytracing.html. Luettu 22.11.2013.

Camargo, D. 2009. CSC 471 Final Project. Www-dokumentti. Saatavissa: http://users.csc.calpoly.edu/~zwood/teaching/csc471/final09/dcamargo_webpage/. Luettu 1.11.2013.

Cross, D. 2013. Fundementals of Ray Tracing. Pdf-dokumentti. Saatavissa: http://www.cosinekitty.com/raytrace/raytrace_a4.pdf. Luettu 22.11.2013.

Darnell, N. 2011. Robust Inside and Outside Solid Voxelization. Www-dokumentti. Saatavissa: http://www.nickdarnell.com/2011/09/robust-inside-and-outside-solid-voxelization/. Luettu 10.12.2013

Fleet, D. & Hertzmann, A. 2006. Computer Graphics Lecture Notes. Pdf-dokumentti. Saatavissa: http://www.dgp.toronto.edu/~hertzman/418notes.pdf. Luettu 24.2.2014.

Furht, B. 2010. Handbook of Multimedia for Digital Entertainment and Arts. Springer-Verlag New York Inc.

Gonzalez, G., Lucchi, A. & Fua, P. 2013. Computer Vision Laboratory. Www-dokumentti. Saatavissa: http://cvlab.epfl.ch/research/completed/medical/8tree. Luettu 10.12.2013

Habib's Water Shader. 2009. Www-dokumentti. Saatavissa: http://habib.wikidot.com/techniques. Luettu 5.3.2014.

Hiltunen, K., Latva, S. & Kaleva, J-P. 2013. Suomen peliteollisuuden kehityspolku 2013. Pdf-dokumentti. Saatavissa: http://www.tekes.fi/Julkaisut/peliteollisuus_kehityspolku.pdf. Luettu 6.3.2014.

Hughes, M. 1998. 3D Graphic Java: Render fractal landscapes. Saatavissa: http://www.javaworld.com/article/2076745/learn-java/3d-graphic-java–render-fractal-landscapes.html. Luettu 7.2.2014.

Jackiw, N. 1997. The Geometry of Computer Graphics. Www-dokumentti. Saatavissa: http://dynamicgeometry.com/General_Resources/Recent_Talks/Sketchpad_40_Talks/Computer_Graphics.html. Luettu 26.2.2014.

Keswani, S. 2013. Is the Video Game Industry Leaving the Movie Industry Buried in the Rubble. Www-dokumentti. Saatavissa: http://blog.dlink.com/are-video-games-beating-out-the-movie-industry/. Luettu 10.3.2014.

Lanier. L. 2008. Advanced Maya Texturing and Lightning (2nd Edition). 2., uudistettu painos. Wiley.

Macey, J. 2013. Ray-tracing and other Rendering Approaches. Pdf-dokumentti. Saatavissa: http://nccastaff.bournemouth.ac.uk/jmacey/CGF/slides/RayTracing.pdf. Luettu 22.11.2013.

Magnor, M. 2006. Introduction to Computer Graphics. Pdf-dokumentti. Saatavissa: http://www.mpi-inf.mpg.de/departments/irg3/ws0506/cg/slides/01-intro.pdf. Luettu 10.3.2014

Marsan, C. 2009. The Evolution of Internet. Www-dokumentti. Saatavissa: http://www.networkworld.com/slideshows/2009/020909-evolution-internet.html. Luettu 20.2.2014.

Martz, P. 1997. Generating Random Fractal Terrain. Www-dokumentti. Saatavissa: http://www.gameprogrammer.com/fractal.html#diamond. Luettu 7.2.2014.

Morcillo, C. 2012. School of Computer Science. Topic 3: Materials & Textures. Pdf-dokumentti. Saatavissa: http://www.esi.uclm.es/www/cglez/downloads/docencia/AC/topic3_materials.pdf. Luettu 15.1.2014.

NVIDIA. 2010. Design Garage. Www-dokumentti. Saatavissa: http://www.nvidia.com/coolstuff/demos#!/design-garage. Luettu 24.10.2013.

NVIDIA. 2012. NVIDIA Optix Ray Tracing Engine: Programming Guide 3.0. Pdf-dokumentti. Saatavissa: http://developer.download.nvidia.com/assets/tools/files/optix/3.0.1/NVIDIA-OptiX-SDK-3.0.1-OptiX_Programming_Guide.pdf. Luettu 8.11.2013.

Oatley, C. 2009. Www-dokumentti. Saatavissa: http://craigo917.blogspot.fi/2009/11/ray-tracing-ray-casting.html. Luettu 12.11.2013.

Optics HQ. 2011. What is field of view. Www-dokumentti. Saatavissa: http://www.opticshq.co.uk/acatalog/What_is_field_of_view_.html. Luettu 28.10.2013.

Permadi, F. 2010a. Ray-casting and ray-tracing. Www-dokumentti. Saatavissa: http://www.permadi.com/tutorial/raycast/rayc2.html. Luettu 24.10.2013.

Permadi, F. 2010b. Limitations of Ray-Casting. Www-dokumentti. Saatavissa: http://www.permadi.com/tutorial/raycast/rayc3.html. Luettu 25.10.2013.

Permadi, F. 2010c. Defining Projection Attributes. Www-dokumentti. Saatavissa: http://www.permadi.com/tutorial/raycast/rayc5.html. Luettu 25.10.2013.

Pohl, D. 2012. Quake Wars Gets Ray Traced. Www-dokumentti. Saatavissa: http://software.intel.com/en-us/articles/quake-wars-gets-ray-traced/. Luettu 16.12.2013.

Purgathofer, W. 2011. Laborübung Computergraphik 1. Pdf-dokumentti. Saatavissa: http://www.cg.tuwien.ac.at/courses/CG1/textblaetter/englisch/10%20Ray%20Tracing%20(engl).pdf. Luettu 10.2.2014.

Scratchapixel. 2012. Noise. Www-dokumentti. Saatavissa: http://www.scratchapixel.com/lessons/3d-advanced-lessons/noise-part-1/introduction/. Luettu 20.1.2014.

Shirley, P. & Marschner, S. 2009. Fundamentals of Computer Graphics. 3., uudistettu painos. Taylor & Francis Group.

Summers. D. 2004. Texturing: Concepts and Techniques. Charles River Media / Cengage Learning.

Taivalsaari, A. & Mikkonen, T. 2011. The Web as an Application Platform: The Saga Continues. 170 – 174.

technopedia. 2014. Volume Pixel. Www-dokumentti. Saatavissa: http://www.techopedia.com/definition/2055/volume-pixel-volume-pixel-or-voxel. Luettu 7.2.2014.

The Economist. 2013. The meaning of ”Doom”. Www-dokumentti. Saatavissa: http://www.economist.com/blogs/babbage/2013/12/video-games. Luettu 7.3.2014.

The Stack. 2008. Strife: Doom Engine. Www-dokumentti. Saatavissa: http://www.wurb.com/stack/archives/406. Luettu 20.11.2013.

UnknownRoad. 2014. Constructing Eye Rays. Www-dokumentti. Saatavissa: http://www.unknownroad.com/rtfm/graphics/rt_eyerays.html. Luettu 11.2.2014.

Vandevenne, L. 2007. Lode's Computer Graphics Tutorial. Www-dokumentti. Saatavissa: http://lodev.org/cgtutor/raycasting.html. Luettu 23.10.2013.

Valve Developer Community. 2012. Field of View. Www-dokumentti. Saatavissa: https://developer.valvesoftware.com/wiki/Field_of_View. Luettu 28.10.2013.

Woodford, C. 2013. Computer graphics. Www-dokumentti. Saatavissa: http://www.explainthatstuff.com/computer-graphics.html. Luettu 24.2.2014.

Xojo3D. 2013. Www-dokumentti. Saatavissa: http://www.xojo3d.com/tut011.php. Luettu 13.1.2014.