Tuomas Turppa

Laserkeilaus osana pelinkehitystä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Mediatekniikan koulutusohjelma

Insinöörityö

26.4.2014

Metropolia

Tekijä Otsikko Sivumäärä Aika	Tuomas Turppa Laserkeilaus osana pelinkehitystä 87 sivua 26.4.2014
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Mediatekniikka
Suuntautumisvaihtoehto	Digitaalinen media
Ohjaajat	Professori Hannu Hyyppä Koordinaattori Marika Ahlavuo Yliopettaja Harri Airaksinen

Insinöörityön tarkoituksena oli tutkia erilaisilla laserkeilausmenetelmillä tuotetun materiaalin hyödyntämistä pelinkehityksessä. Päämääränä oli suunnitella ja toteuttaa pelimoottorilla näitä teknologioita hyödyntävä peli. Insinöörityön tavoitteena oli myös havainnollistaa pelin tekemisen eri vaiheita, kuten ohjelmointia, mallintamista, animoimista ja pelin rakenteen suunnittelua. Työ tehtiin Aalto-yliopiston maankäyttötieteen laitoksella rakennetun ympäristön mittauksen ja mallinnuksen instituutissa.

Laserkeilauksella tuotettiin pienehköistä esineistä malli teollisuuskeilaimella. Maalaserkeilausta hyödynnettiin pelissä rakennuksen julkisivun, henkilön ja muistomerkin pistepilven tekemiseen. Käsin tehtyä kuvapohjaista tarkkaa mittaustekniikka hyödyntäen mallinnettiin Olavinlinna ja Helsingin Sinebrychoffin puistossa sijaitsevasta torni. Korkeapolyogniset mallit siirrettiin 3D-mallinnusohjelmaan, jossa niille tehtiin kolmioverkon optimointia, pelimaailmaan sopivan mallin luomiseksi. Tämän jälkeen mallille luotiin kartoituksia erilaisia tekniikoita käyttäen.

Työn aikana havaittiin, että lasermittausmenetelmillä tuotetusta korkeapolygonisesta mallista voidaan tuottaa pienipolygoninen, peliympäristöön soveltuva malli ja hyödyntää korkeapolygonista mallia normaalikartan luonnissa. Ongelmakohdaksi nousi tietyissä tilanteissa automaattisen UVW-kartan laatu, ja se jouduttiinkin yleensä uudelleen rakentamaan manuaalisesti. Pelin ohjelmoinnissa hyödynnettiin C#:a ja Unityscriptiä. Lasermittauksella on kiistaton etu nykymaailmaan sijoittuvien pelimaailmojen mallintamisessa, koska se kykenee tarjoamaan oikeilla mittasuhteilla varustettuja, automaattisesti luotuja malleja.

Avainsanat	laserkeilaus, pelinkehitys, ohjelmointi, 3D-grafiikka, interaktiivisuus, Unity, 3D-mallinnus

Author Title Number of Pages Date	Tuomas Turppa Laserscanning as a part of game development 87 pages 26 April 2014
Degree	Bachelor of Engineering
Degree Programme	Media Technology
Specialisation option	Digital Media
Instructors	Hannu Hyyppä, Professor, Project supervisor Marika Ahlavuo, Coordinator Harri Airaksinen, Principal Lecturer

The game industry is one of the most formidable forces in pushing computer related technologies forward with their constant need for better performance and graphical quality. Laser scanning is a rapidly evolving technology to collect point data from different surfaces with a laser beam. From this point data it is possible to generate highly detailed models almost automatically therefore reducing the amount of time and resources needed for modelling.

The goal of this thesis is to study the applications of laser scanned models in games and to design and develop a game using these technologies. The study consists of eight chapters. First, there will be an introduction to laser scanning technologies in general, describing the basic principle of the laser scanner and the tools used with it. Next, Unity, the game engine used in this project, will be presented.
It is vital to understand the work performed by the game engine and graphic libraries behind the scenes. That is why there will be a more in depth look to the graphic pipeline, mostly from OpenGL- graphic library perspective. After covering the background information, techniques to achieve better graphical quality and performance are shown. Finally, the workflow of the project is reviewed.

In conclusion, laser scanning can improve the speed of the game modelling step, as well as provide more accurate models when replicating something from the real world. The greatest advantage is the auto-generation of a high polygon model. This speeds up the production of the normal map for the game model. In the case of stationary models, it can also be used to produce low polygon models.

Keywords	Laser scanning, Games, 3D-modelling, progamming,Unity

Sisällys

Sanasto

3D-grafiikka	3D-grafiikka eli kolmiulotteinen grafiikka pyrkii hyödyntämään kolmea ulottuvuutta visualisoinnissan. Tietokonegrafiikassa kyse on yleensä, virtuaalisen, kolmiulotteisen tilan esityksestä kaksiulotteiselle pinnalle projisoituna, esimerkkinä tietokoneen monitori.
3D-mallinnus	Kolmiulotteinen mallinnus tarkoittaa virtuaalisessa kolmiulotteisessa avaruudessa tapahtuvaa mallinnusta jossa malli muodostuu sen määrittävistä kulmapisteistä sekä näiden välisestä kolmiopintaverkosta
FPS (Frames Per Second)	Ruudunpäivitysnopeus. Kertoo kuinka monta kertaa pelin näkymä pystytään piirtämään sekunnissa. Eri peligenreillä toivottava minimi FPS vaihtelee aina 60 ja 24 piirron per sekunti välillä.
Frustum	Katkaistun pyramidin muotoinen tila kolmiulotteisessa avaruudessa, joka edustaa kameran, eli katsojan näkökenttää, siten että katsojan näkymä sijaitsee pyramidin katkaistulla huipulla.
Kolmioverkkopinta	Useasta polygonista muodostuva suurempi kokonaisuus, 3D-mallin pinta. (Mesh)
Komponentti	Pohjaluokka kaikkiin peliobjektehin lisättäville komentosarjoille.
GUI	(Graphical User Interface) Graafinen käyttöliittymä.
UI	(User Interface) Käyttöliittymä

Laserkeilaus	Mittaustapa, jolla tuotetaan kohteen pinnasta lasersäteiden avulla mittatarkkaa kolmiulotteista tietoa kohteeseen koskematta.
Peliobjekti	Perusluokka kaikille Unity-editorin sisäisille objekteille. Käytetään komponenttien kapsulointiin ja näin rakenteen ylläpitämiseen. (Game Object)
Polygonit	Geometrinen n-kulmainen tasokuvio, jonka sijainnin ja mitat muodostavat näissä kulmissa sijaitsevat pisteet. Tämä on 3D-mallinnuksen perusrakennuselementti.
Prefab	Esirakennettu peliobjekti, josta voidaan luoda peliobjekteja ja hallinnoida näiden ominaisuuksia.
RMS	Tehollisarvo (Root Mean Square) kuvaa värähtelysignaalin suurutta.
Tekseli	Värikartan mukaisten pikselien muunnos näytöllä 3D-mallissa.
Tekstuuri	3D-mallin pinnan tekselien värin määrittävä värikartta.
Transform	Komponentin perivä luokka, joka sisältää sijainnin, orientaation sekä skaalauksen. Jokaisella peliobjektilla on oma Transform-instanssi.
Triangulaatio	Polygonien muuntuminen yksinkertaisimmaksi tasoksi eli kolmikulmioksi. Mallit muunnetaan aina kolmikulmioksi ennen näytölle piirtämistä.
Unity	Unity on pelimoottori, joka tarjoaa integroidun graafisen editorin, debuggerin, koodieditorin sekä intuitiivisen käyttöliittymän pelien tekemiseen.

Varjostin	Ohjelmoijan ohjelmoitavissa oleva pienoisohjelma joka on osa piirtoliukuhihnaa. Käsittelee kuvadataa ennen sen viemistä näytölle. On olemassa erilaisia varjostimia. Pistevarjostimet määrittävät piirrettävän mallin rautalankamallin. Pikselivarjostimet määrittävät rautalangan pisteiden väliset tasot.
Verteksi	Merkitsevä piste 3D-avaruudessa. Verteksit määrittävät esimerkiksi polygonit toimimalla näiden kulmapisteinä.
Riggaus	Järjestelmien luontia 3D-mallin pinnan deformaation hallintaan eli animaatioita varten.

1 Johdanto

Nykypäivän pelien ulkoasu kehittyy jatkuvasti ja lähestyy nopeasti realismia. Tämä johtaa kasvavaan laskentatehovaatimukseen laitteistopuolella, ja siksi peliteollisuus onkin ollut tietokoneen komponenttien, kuten näytönohjainten ja prosessorien, kehityksen tärkeimpiä edesauttajia. Visuaalisesti 3D-pelimaailma on dynaaminen, ja sitä voi tarkastella joka suunnasta. Immersio on tärkeä osa virtuaaliympäristöjä, ja illuusiota elävästä ympäristöstä koetetaan lisätä monella eri tavalla. Maailma ei ole staattinen, pilvet liikkuvat taivaalla, ruohot tuulen mukana, tuuli ujeltaa kantaen kaukaisuudesta etäisiä ääniä, puun lehdet luovat varjoja ruohonkorsiin lintujen lentäessä taivaalla. Erityisesti illuusiota todellisuudesta lisäävätkin nimenomaan liike, valo, heijastumiset, varjot, äänet, hahmot, rakennukset, esineet ja kasvusto ja näiden välinen interaktio toistensa kuten myös tarkastelijan kanssa.

Laserkeilauksen mahdollisuuksia on hyvä tutkia pelinkehityksen kannalta, koska reaaliaikaisuus ja virtuaalimaailmat luovat myös muita kaupallisia mahdollisuuksia pelillistämisen ohessa, kuten todellisen maailman päivitettävää visualisointia esimerkiksi paikkatiedossa ja rakennusprojekteissa, joita peliin yhdistämällä saadaan aivan uudenlaista lisäarvoa. [1 s.14; 2; 3.]

3D-mallinnus on merkittävä osa peliteollisuutta, ja siihen investoidaan valtavasti resursseja peliprojekteissa. Pelivalmiin mallin luominen on moniulotteinen prosessi, joka vaatii sekä teknistä ja taiteellista tietoa että osaamista. Todellista maailmaa 3D-ympäristöön muunnettaessa lasermittaustekniikat poistavat tarpeen korkeapolygonisen mallin luomiselle käsin ja tarjoavat mallin sisäisesti oikeat mittasuhteet. [4. s.26.]

Insinöörityön tavoitteena on perehtyä lasermitatun materiaalin soveltuvuuteen erilaisina liikkumattomina malleina sekä animoituina malleina peliympäristössä. Insinöörityön tavoitteena on myös havainnollistaa pelin tekemisen eri vaiheita, kuten ohjelmointia, mallintamista, animoimista ja pelin rakenteen suunnittelua.

Toteutettavaa 3D-malliosaamista ja pelimoottorialustaa hyödynnetään Aalto-yliopistossa ja Geodeettisessa laitoksessa "wikikartoituksen" (engl. crowdsourcing, neogeography, volunteered geographic information) eli käyttäjien tuottaman karttatiedon tutkimuksessa. Tutkimus toteutetaan Rakennetun ympäristön mittauksen ja mallinnuksen instituutissa.

Instituutti on yhteinen Aalto-yliopiston ja Geodeettisen laitoksen yksikkö. Instituutin tavoitteena on tuottaa kansainvälisen huipputason tutkimus- ja kehitystyötä maanmittaustekniikan ja IT:n alalla rakennus- ja ympäristötekniikan tarpeisiin. Rakennetun ympäristön mittauksen ja mallinnuksen instituutti ja Geodeettinen laitos muodostavat rungon Suomen Akatemian laserkeilaustutkimuksen huippuyksikölle (2014 –2019).

Suomen Akatemian laserkeilaustutkimuksen huippuyksikköön (kuva 1) kuuluu Oulun ja Helsingin yliopiston, Aalto-yliopiston ja Geodeettisen laitoksen yksikköjä. Huippuyksikkö hyödyntää modernia maanmittausta, IT:tä ja paikannusta ja navigointia toiminnassaan [2].

Kuva 1. Suomen Akatemian laserkeilaustutkimuksen huippuyksikkö.

Tämä tutkimus toteutetaan Suomen Akatemian, Suomen kulttuurirahaston ja RYM-Shokin rahoittamissa Roadside-, Fokus- ja Energizing Urban Ecosystem -projekteissa.

2 Laserkeilaus

Laserkeilaus ei ole aivan uusi asia peliteollisuudelle, vaan sitä on jo osattu hyödyntää jonkin aikaa, esimerkiksi joissakin rallipeleissä. Myös suuret pelitalot, kuten Activision käyttävät laserkeilausta peleissään. [3.]

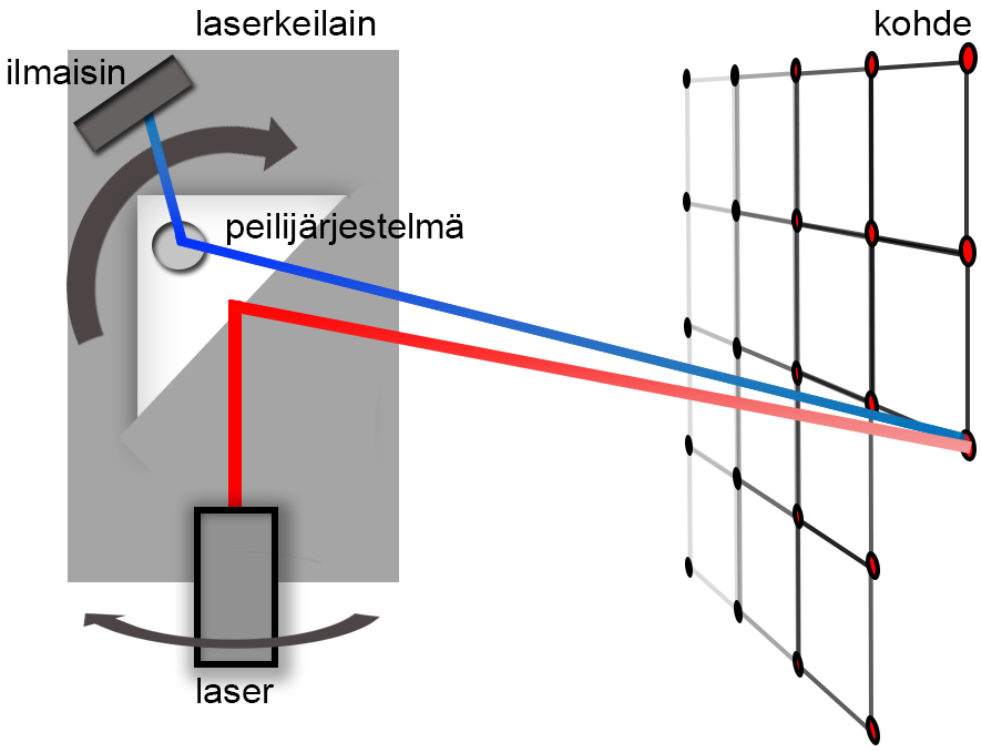
Laserkeilausmittauksen kehityksen ovat mahdollistaneet viimeaikaiset asennon ja kallistuksen seuraamisjärjestelmät (inertiajärjestelmä, IMU), paikannus- ja navigointijärjestelmät (GPS) ja laserkeilaimien pistemittaustaajuuden kehitys. Laserkeilauksessa on kyse kohteen geometrian 3D-mittaamisesta (x, y, z) laserkeilainkeskiseen kolmiulotteiseen koordinaatistoon. Koordinaattien lisäksi tallennetaan jokaiselle pisteelle myös intensiteettiarvo. Laserkeilaimiin on usein

liitettävissä joko ulkoinen tai sisäinen digitaalikamera, jolloin pinnat saadaan teksturoitua luonnollisemman näköisiksi nopeaa visualisointia varten. Kerätyn koordinaattiaineiston muodostamaa pistekuvaa kutsutaan pistepilveksi. Kun kohteesta on saatu pistepilvi, se voidaan algoritmien ja ohjelmistojen avulla kääntää 3D-malliksi eli luoda kolmiopinta eri pisteiden välille. Usein pistepilvistä luodut mallit ovat erittäin yksityiskohtaisia ja näin tiedostokooltaan suuria. Puhutaankin miljoonista tai jopa miljardeista polygoneista. [6, s. 10–11; 7, s. 14.]

Laserkeilain koostuu yleensä kolmesta komponentista, laserpulssin tuottavasta lasertykistä, laseria poikkeuttavasta keilainosasta sekä ilmaisinosasta. Ilmaisinosa määrittää vastaanotetun signaalin perusteella etäisyyden pintaan, josta laser on taittunut. Etäisyys lasketaan yhtälöllä

aika × valonnopeus/2 eli r = dt × c/2.

Kuljetun matkan ja säteen kulman perusteella voidaan laskea heijastumakohdan koordinaatit keilainkeskisessä koordinaatistossa. Takaisin palaavaan säteeseen on myös kertynyt informaatiota heijastumakohdan materiaalista ja sävyarvosta säteen intensiteetin muutoksena. Tästä on etenkin pistepilveä tarkasteltaessa suurta hyötyä hahmottamisen kannalta, sillä se auttaa luomaan syvyysvaikutelman. Kuvassa 2 näkyy laserkeilauksen perusperiaate. Uusimmat mobiili- ja maalaserkeilaimet saavuttavat millimetrien tarkkuuden [7, s. 13]. Kun tieto on tarkkaa, on se myös laskennallisesti hyvin raskasta. Tällöin ennen kuin voidaan luoda pistepilvelle pinta, täytyy sitä myös algoritmein harventaa. [6, s. 10–11.]



Kuva 2. Maalaserkeilaimen perusperiaate

Laserkeilausta voidaan inertia- ja paikannusteknologian ansiosta tehdä kustannustehokkaasti liikkuvista alustoista kuten selkärepusta tai autosta. Aineiston tarkkuuden kannalta on olennaista keilaimen huolellinen kalibrointi, mahdollisimman korkea sijainti katvealueiden minimoimiseksi sekä paikannussatelliittien hyvä näkyvyys. Merkittävimpiä ongelmakohtia mitattaessa liikkeestä ovatkin katvealueiden, kuten metsäisestä ympäristöstä tai tunnelista johtuva heikko GP-satelliittisignaali. Tällöin joudutaan luottamaan pelkkään inertiamittauksen tuottamaan sijaintitietoon. Lyhytaikaiset sisätiloissa tehdyt mittaukset ovat kuitenkin tuottaneet lupaavia tuloksia, kun inertiamittauksen ylläpitämää sijaintitietoa on korjattu tunnettujen pisteiden avulla. [8, s. 13–14.]

Maan päällä tapahtuvan laserkeilauksen toimintasäde yltää noin sataan metriin pistetiheyden yltäessä jopa tuhansiin pisteisiin jokaista neliömetriä kohden. Aineistojen tarkkuus on noin 2–5 senttimetriä korkeudessa ja alle 10 senttimetrin luokkaa tasosijainnissa. [8, s. 13–14.]

Teknologian yleistyessä keilausjärjestelmien hinnat laskevat ja keilausjärjestelmien koot kutistuvat, ja näin teknologia muuttuu helpommin yhden henkilön operoitavaksi Kuva 3).

Perinteisiin kartoitusjärjestelmiin verrattaessa laserkartoitusjärjestelmät tarjoavat merkittävästi paremman resoluution, jota voidaan käyttää myös jo kerätyn tiedon täydentämiseen.

Kuva 3. Liikkuvan keilauksen Roamer-järjestelmä. © FGI ja Aalto. Hannu Hyyppä.

Laserkeilauksessa on omat haasteensa. Esimerkiksi sää voi keilaushetkellä haitata hyvinkin paljon. Luonnollisesti, jos kohde on lumen peitossa, säteillä ei ole pääsyä kohteen pintaan. Jo lumisade on hankala, koska säteet heijastuvat takaisin hiutaleistakin, jolloin pistepilveen tulee epäkiinnostavaa tietoa, kuten niin sanottuja harhapisteitä. Monimutkaisiin tai korkeisiin pintoihin taas jää helposti katvealueita, joihin säteet eivät pääse. Laserkeilauksessa ongelmaksi nousevat myös läpikuultavat ja heijastavat pinnat, jolloin säde saattaa läpäistä pinnan tai heijastua. Säällä ja kohteella on siis oma merkityksensä keilattaessa. [10, s. 25.]

Laserkeilaus erottuu muista mittausmenetelmistä nopeuden, tarkkuuden, aineiston nopean siirron sekä turvallisuuden puolesta. Mittaustapahtuma ei vahingoita kuvattavaa

kohdetta, ja sen voi toteuttaa etäältä. Tästä on ollut suurta hyötyä etenkin arkeologian ja maanmittauksen puolella. Laserkeilauksen tarkkuutta on osattu hyödyntää merkittävästi myös proteesien valmistuksessa. [10, s. 24– 26.]

Laserkeilauksen ja pelimoottorilla toteutetun sovelluksen yhdistäminen on rakennetun ympäristön alalla uudehko, mutta erityisesti kaupunkeja kiinnostava kehityskohde. Geodeettisen laitoksen (GL) liikkuvan kartoituksen tutkimusryhmä ja myös omalla tahollaan Sito Oy ovat luoneet 3D-mallin Tapiolan keskuksesta. Samanlaisia 3D-malleja on tehty myös Tampereesta ja Oulusta. Mallien visualisointi on toteutettu pelinkehitystyökaluilla. Niihin voi yhdistää henkilönavigointia, muutostulkintaa sekä fyysistä ja täysin virtuaalista ympäristöä. GL:n Tapiolan 3D-malli on vapaasti ladattavissa Android-käyttöjärjestelmän puhelimiin ja taulutietokoneisiin osoitteesta https://play.google.com/store/apps/details?id=com.FGI.Tapiola3D. Mallin vaatimat 3D-mittaukset on tehty Geodeettisessa laitoksessa ja Aalto-yliopistossa rakennetulla liikkuvalla ROAMER-kartoittimella. Pistepilvet on työstetty malliksi lähes automaattisella prosessointiketjulla. [1, s. 15.]

3D-pelimoottoriin siirretty malli havainnollistaa uusia paikkatietosovelluksia, jotka tarjoavat mahdollisuuden yhdistää virtuaalisen ja fyysisen maailman reaaliajassa [1, s. 14].

Liikkuvan laserkeilauksen avulla voidaan tuottaa rakennusten geometria, johon yhdistetään samaan aikaan otetuista kuvista tekstuurikartta. Lopuksi malli tarkistetaan ja korjataan manuaalisesti. Ympäristön hahmottamista on yksinkertaistettu poistamalla näkymästä epäoleellinen informaatio kuten autot, katukalusteet ja ihmiset. [1, s. 14.]

2.2 Laserkeilaimella mittaaminen

Laserkeilaimella mitataan joko kiinteästä tai liikkuvasta alustasta (kolmijalka, ihminen, lentokone, helikopteri, miehittämätön lentoalus, ajoneuvo tai lennokki). Laajoille alueille otollisin mittausalusta on lentokone, vuoristoisille helikopteri. Lentoaluksilla mitattaessa mittaus tehdään yleensä kolmella tai neljällä risteävällä ylilennolla. Miehittämätön lentoalus ja lennokki taas ovat kustannuksiltaan kaikkein kilpailukykyisimmät. [9, s. 5.]

Maalaserkeilauksessa mittaus tehdään maasta käsin. Usein mittaus tehdään kolmijalalle asetetulla keilaimella. Kuvassa 4 on Faron maalaserkeilain.

Maalaserkeilausjärjestelmä rakentuu keilaimen lisäksi pakkokeskitysalustasta, jalustasta, virtalähteestä ja tietokoneesta. Pistepilvi tallennetaan yleensä suoraan keilaimen muistiin. [6, s. 16–17.]

Maalaserkeilauksessa pisteiden mittaus tehdään säännöllisen ruudukkomaisesti (ks. kuva 2 s. 4). Mitä lähempänä laserkeilainta mittausalue on, sitä tiheämpää tietoa saadaan. Keilaimesta riippuen ruudukon tiheyttä pystyy kuitenkin säätämään. Tavallisesti mittausetäisyys vaihtelee metristä satoihin metreihin , ja mittaustarkkuus on parhaimmillaan alle senttimetrin. [6, s. 16–17.]

Laserkeilauksen soveltuvuus eri käyttötarkoituksiin määräytyy sen teknisten ominaisuuksien, kuten mittausetäisyyden, lasersäteen divergenssin eli säteen hajoamiskulman ja keilauskulman, perusteella. Kulmaresoluutio määrittää kohteen yksityiskohtien mittauksen tarkkuuden. Mitä tarkempi keilauskulman jako on, sitä tiheämmin pystytään mittaamaan. Keilauskulma on myös merkittävä tekijä mittaustapahtuman kestossa. Kapealla näkemällä varustetuilla keilaimilla työmäärä lisääntyy, kun keilainta joudutaan siirtämään ja kääntelemään enemmän. [6, s. 13–17.]

Kuva 4. Faron maalaserkeilain. © FGI ja Aalto. Hannu Hyyppä.

Koska keilainta joudutaan liikuttamaan, jotta koko kohde saadaan mitattua ja vältytään pahimmilta katvealueilta, tulee ongelmaksi erillisten koordinaattiavaruuksien synty joka mittauskohdasta ja näin koordinaateiltaan epäsynkronoidut pistepilvet. Pistedatasta

saadaan yhtenäinen asettamalla yhtymäkohdat mittauskohtien välille eli jokin referenssikohde eri koordinaatistojen välille. Tähän tarkoitukseen käytetään tähyksiä. Tähys on yleensä esimerkiksi ympäristöstä helposti erottuva symmetrinen objekti tai heijastinlevy. Tämä piste toimii siis synkronisoivana tekijänä pistepilvikoordinaatistoja yhdistettäessä.

Kohteesta riippuen mittauspisteitä tarvitaan useita. Laitetta siirretään sopiviin paikkoihin, jotta kohde saataisiin mitattua joka puolelta mahdollisimman vähillä katvealueilla. Maalaserkeilain tallentaa pisteet omaan sisäiseen koordinaatistoonsa. Jos pistepilvet halutaan yhdistää johonkin tiettyyn koordinaattijärjestelmään, täytyy apuna käyttää aikaisemmin mainittuja tähyksiä ja takymetrimittauksia. Tähykset ovat usein pallon muotoisia, ja niitä sijoitetaan mitattavalle alueelle useita siten, että eri mittauksilla saadaan tuloksia myös muutamasta samasta tähyksestä. Tähysten keskipisteet mitataan takymetrilla, jolloin laserkeilaimen tuottamat pistepilvet voidaan myöhemmin muuntaa haluttuun koordinaatistoon ja saada näin synkronisoitua. [6, s. 16.]

Teollisuuslaserkeilainkin on periaatteessa maalaserkeilain, se on tosin tarkoitettu pääasiallisesti pienikokoisten kohteiden tarkkaan mittaamiseen. Teollisuuskeilaimen eli niin sanotun 3D-esineskannerin tavallisin toimintaperiaate on pyyhkäistä säteellä kohdetta ja ohjata heijastunut valo objektiivin kautta sensoreille, jotka muodostavat pistepilven. Menetelmää kutsutaan optiseksi kolmiomittaukseksi, koska laserin heijastumakohta, kamera ja laserin lähde muodostavat kuvitteellisen kolmion. Esineskannerin mittaustarkkuus voi olla jopa alle millimetrin sadasosia. Mittaus tehdään kuitenkin enimmillään alle 30 m:n etäisyydeltä. Sovellutuksia teollisuuskeilamille on useita. Laitteita hyödynnetään pääasiallisesi metalliteollisuuden pienten objektien mittaamisessa, lääketieteessä, arkeologian sovelluksissa mutta myös useilla muilla aloilla erilaisten pintojen ja muotojen muutosten tarkkailuun. [6, s. 17.]

2.3 Pistepilven laatu

Pistepilven jatkojalostusta, kuten kolmiopintaisen 3D-mallin luontia, varten on kehitetty erilaisia ohjelmia, ja niiden pääasiallinen tarkoitus on suodattaa ja yhdistää aineistoa, muuntaa eri koordinaatistoihin, analysoida pistepilveä, kuten havaitsemalla tasoja, ja tämän pohjalta luoda kolmiopinnat, UVW-kartoittaa ja teksturoida ne.

Pistepilvien laatua määrittää etenkin niiden tarkkuus. Koska pistepilvet saavuttavat helposti äärimmäisiä mittasuhteita hidastaen niiden käsittelyä merkittävästi, tulisi tarkkuuden aina skaalautua käyttötarkoituksen mukaan. Myös halutun mittauskohteen tarkka rajaus edesauttaa ylimääräisen tiedon karsimisessa. Tarkkuutta pystytään säätämään jo mittaustilanteessa. [6, s. 11; 15, s. 27–28.]

Toisena aineiston laatuun vaikuttavana tekijänä on pisteiden välimatka. Pisteiden tiheys heikkenee suhteessa etäisyyteen. Vaihe-eromenetelmään pohjautuvilla keilaimilla pystytään mittaamaan 50 metrin etäisyydeltä 8 mm:n tarkkuudella ja valon kulkuaikaan perustuvilla jopa kolme kertaa tiheämmin. Äärietäisyyksiä mitattaessa laatu vaihtelee laitteiston toimintaperiaatteiden mukaan. [6, s. 18; 11, s. 27–28.]

Muita mainittavia pistepilvien laadun määrittäviä tekijöitä ovat mittauksen kohina, pisteiden hajonta sekä lasersäteen halkaisija. Hajontaan vaikuttaa mittaussäteen osumiskulma kohteeseen. Mitattaessa tärkeäksi nouseekin tämän perusteella tapahtuva jäännösvirheiden eli residuaalien seuranta pistepilvestä mallia luotaessa. Ne esittävät tasoitetun ja mitatun pisteen eroa. Residuaalien tulisi noudattaa normaalijakaumaa, joten systemaattinen virheiden paikannus on mahdollista. [12, s. 22–24; 6, s. 13.]

Geodeettisen laitoksen Mobile mapping -ryhmä on kehittänyt lähes automaattisia menetelmiä muun muassa maastomallien muutostulkintaan, puuston mittaukseen, pylväsmäisten kohteiden havainnointiin ja tieympäristön inventointiin liikkuvalla laserkeilauksella kerätyistä pistepilviaineistoista. [1, s. 13.]

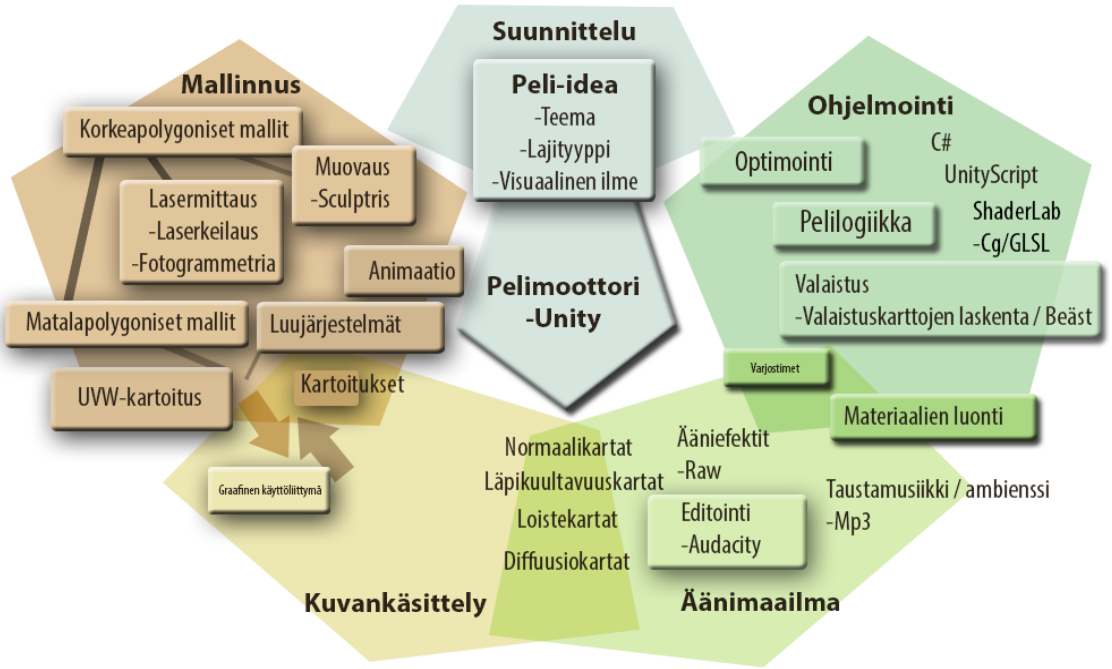
Laserkeilauksen etuna on mittatarkka, yksityiskohtainen malli, josta on suuri hyöty rakennetun ympäristön dokumentoinnissa. Tulevaisuudessa automatisointi voidaan kehittää niin pitkälle, että vastaavan mallin prosessointiin operatiivisesti menee vain murto-osa nykyisestä ajasta, ja sen käytön potentiaali peliympäristöissä kasvaa ilman jälkikäsittelyjä.

3 Pelinkehitys ja Unity-pelimoottori

Pelejä pidetään usein vain ajanvietteenä, mutta ne voivat sisältää myös opetuksellisia aspekteja. Simulaattoreilla voidaan kouluttaa eri laitteiden käyttöä turvallisesti [4, s. 11–12], räiskintäpelit harjaannuttavat reaktionopeutta, strategiapelit taktikointia. Usein pelit myös parantavat kielitaitoa, kun pelin kieli on lähes aina englanti. Poikkeuksena tietenkin englantia äidinkielenään puhuvat. Pelit mielletään usein epäsosiaalisiksi, väite jonka nykypäivän usean henkilön yhtäaikaiset verkkopelit haastavat.

Peli koostuu yleensä jonkinlaisista haasteista, tavoitteista, joihin pelaaja pyrkii. Peleissä voidaan pyrkiä herättämään kilpailu- tai keräilyviettiä, ja niissä usein onkin jonkinlaista keräämistä ja vastustajia läsnä. Pelin on oltava myös palkitsevaa, eli saavutuksista on saatava jonkinlaista palautetta, olkoon se sitten lisäpisteitä, virtuaaliesineitä tai pelin tarinalle jatkoa. Erittäin tärkeää on myös minimoida pelaajan pelaamista häiritsevien asioiden määrä. Tällaisia asioita ovat muun muassa huono ohjausjärjestelmä, vääränlainen grafiikka tai äänet, liika toisto, kameran käyttäytyminen, tekoäly, virheet, eli ”bugit” ja tietenkin pelin sulavuus.

Pelinkehityksen voidaan ajatella alkavan suunnitelmasta ja sen jälkeen sen toteutuksen määrittämisestä. Toteutuksen alkaessa pyritään valvomaan tavoitteiden toteutumista testauksella. Pelinkehityksen voi karkeasti jakaa esimerkiksi suunnitteluun, mallinnukseen, ohjelmointiin, kuvankäsittelyyn ja äänien luontiin (kuva 5).

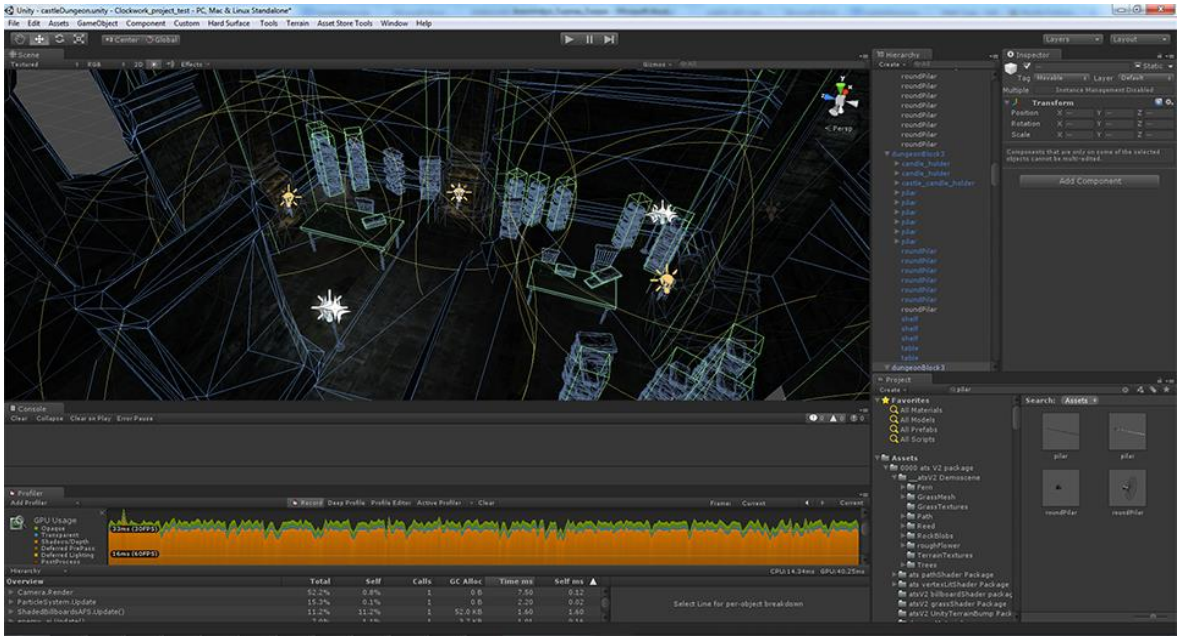


Kuva 5. Pelinkehityksen osa-alueita.

Unity (http://Unity.com/) on pelimoottori, joka tarjoaa integroidun graafisen editorin, ”debuggerin” eli virheenkorjaustyökalun, koodieditorin ja intuitiivisen käyttöliittymän pelien tekemiseen. Sen vahvimpia puolia ovat helppokäyttöisyys ja laaja tuki eri alustoille. Alustoja ovat Windows, Mac OS X, verkkoselaimet, iOs, Android, Playstation 3, Xbox 360, Windows Phone 9 ja Wii U. Pelin rakentaminen perustuu peliobjekteihin liitettävien komentosarjoihin. Pelimoottori hoitaa peleissä usein toistuvat rakenteet, kuten pelisilmukan, grafiikkakirjastojen ja säikeiden hallinnan, tiedostojen luvun, äänen toiston ja muut matalan tason ohjelmointia vaativat toiminnot. Se myös tarjoaa omat kirjastonsa, kuten fysiikka- ja matematiikkakirjastot sekä niiden rakenteet ohjelmoijan käyttöön, mikä vaikuttaa jossain määrin sovelluksen koodin rakenteeseen.

Unitysta on saatavilla sekä ilmainen että Pro-versio. Ilmainen versio on hieman riisutumpi versio Prosta, eikä se sisällä esimerkiksi maastolle reaaliaikaisia varjoja, kuvatehosteita, reitinhakualgoritmeja eikä profiloijaa. Ilmaista lisenssiä voi käyttää maksullisen tuotteen julkaisuun, kun vuosittaiset tuotot eivät ylitä 74 000 euroa. Kuvassa 6 on Pro-versio, jonka näkymä on jaettu siten, että pelikentän visualisointi eli tasonäkymä (Scene view) on vasemmassa yläkulmassa, konsoli (console) tasonäkymän alapuolella ja profiloija (Profiler) sen alapuolella. Oikealla puolella ikkunaa sijaitsevat yläkulmassa vasemmalta

oikealle tason peliobjektien Hierarkia-ikkuna (Hierarchy) ja peliobjektien tarkasteluikkuna (Inspector view). Peliprojektin tiedostot kokoava Projekti-ikkuna (Project) on sijoitettu näiden alapuolelle.



Kuva 6. Unity Pro -version näkymä.

3.1 Ohjelmointi Unityssä

Unityssä pelin ohjelmointi tarkoittaa käytännössä lyhyiden komentosarjakomponenttien kirjoittamista ja lisäämistä eri peliobjekteihin. Komponentit voivat keskustella keskenään, vaikka olisivatkin eri kielillä kirjoitettuja. Niiden vuorovaikutuksen tulee kuitenkin ottaa huomioon eri kielten välinen käännösjärjestys tai käyttää GameObject.SendMessage-metodia. Unity määrittää kansiorakenteen sijainnin komentosarjan ajon ajankohdan mukaan. Unity ajaa komentosarjat eri kansioista seuraavassa järjestyksessä:

1. ajonaikaiset komentosarjat: kansioista Standard Assets, Pro Standard Assets ja Plugins
2. editoriin liittyvät komentosarjat: kansioista Standard Assets/Editor, Pro Standard Assets/Editor ja Plugins/Editor
3. muut komentosarjat, jotka eivät sijaitse Editor-kansiossa
4. Editor-kansion komentosarjat [15].

Unity tukee esimerkiksi seuraavia kieliä:

UnityScript on Unityn oma, JavaScriptia ulkoisesti muistuttava korkean tason kieli, jota voisi luonnehtia tyypitystä (muuttujien muodon esimäärittämistä) tukevaksi, vähemmän joustavaksi JavaScriptiksi ilman prototyyppien tuomia mahdollisuuksia.

C# on vahvasti olio-orientoitunut Microsoftin .Net-konseptia varten luoma korkean tason kieli, joka on muistuttaa merkittävästi Java-ohjelmointikieltä.

Boo on syntaksiltaan Python-ohjelmointikieltä muistuttava, olio-orientoitunut ja staattisesti tyypitettävä, korkeamman tason ohjelmointikieli.

3.2 Unityn työkalut

Asset Store on Unity-kehittäjille suunnattu kauppapaikka, jossa voi myydä toisille kehittäjille omia tuotoksiaan, kuten 3D-malleja, animaatioita, varjostimia, komentosarjoja, projekteja ja äänitiedostoja eli kaikkia pelinkehityksessä tarvittavia elementtejä.

Animator on Unityn mukana tuleva aikajanamuotoinen animointityökalu, jolla pystyy kätevästi luomaan yksinkertaisia tween-animaatioita.

Beast Illuminate on Labsin kehittämä, Unityyn täysin integroitu valonkartoittaja. Sen avulla pystyy esilaskemaan valonlähteiden valaistuksen eri pinnoilla ja tallentamaan ne valokarttoihin, jotka lisätään tekstuurien päälle. Tämä vähentää merkittävästi ajon aikaista laskentaa, koska valon heijastumiset ovat laskennallisesti raskaita ja mahdollistavat korkeampilaatuiset varjot ja illuusion useammista valonlähteistä. [16.]

Mecanim on Unity 4.0:n mukana tullut, perinteisen Animation-järjestelmän lisäksi editoriin integroitu Mecanim-animaatiojärjestelmä, jota voi käsitellä vuokaaviomaisesti omassa näkymässä. Se vaatii huomattavasti vähemmän ohjelmointia kuin Animator-komponentti. Yksi kenties merkittävin Mecanim-ominaisuus lienee sen kyky siirtää eri luujärjestelmien animaatioita toisille luille. [16.]

Pelisilmukka on jokaisen pelimoottorin ydin. Se on ohjelman ajon ajan toistuvasti kutsuttava metodi tai funktio, joka mahdollistaa reaaliaikaisen interaktiivisuuden ja animaation. Pelisilmukka koostuu yleensä useamman säikeen ajosta, jotka käsittelevät erikseen piirron, pelilogiikan, äänet ja fysiikan. Yksinkertaisimmillaan pelisilmukka voisi olla vain while-silmukan sisässä toistuva funktioiden kutsuminen, mitä se ei kuitenkaan juuri koskaan todellisuudessa ole, koska se ei huomioi laitteistojen välisiä eroja. Tällöin, jos peliä pelaa erittäin tehokkaalla laitteistolla, kutsutaan silmukkaa huomattavasti nopeammin kuin vähäisempään laskentaan kykenevässä. Tästä seuraisi pelin tempon muuttuminen laskentatehon mukaan, mikä johtaisi huomattavan epätasa-arvoisiin pelikokemuksiin. Tästä syystä pelisilmukka on yleensä huomattavasti monimutkaisempi arkkitehtuuriltaan.

Jotkin toiminnot, kuten ruudunpiirto, voidaan haluta toimittaa joka ruudunpäivityksellä. Joihinkin pelilogiikoihin riittää tarvittaessa paljon harvempi kutsuminen pelikokemuksen siitä kärsimättä, kun taas esimerkiksi syötteeseen reagoimisen halutaan olevan mahdollisimman responsiivista. Pelisilmukka pyrkii siis pelkkien funktiokutsujen sijaan hallinnoimaan myös piirtoa ja eri säikeiden synkronisointia sekä ylläpitämään haluttua tempoa (eli FPS) tietyllä tasolla kutsumalla eri funktioita näiden painotusten sekä sillä hetkellä tarjolla olevien resurssien mukaan ja sitomalla kaikkein merkityksellisimmät funktiokutsut aikasidonnaisiksi. Tämä johtaa siihen, että eri funktioiden kutsujen määrä jokaista silmukka-ajoa kohden saattaa vaihdella. [17; 18, s. 435–439.]

4.3 Rakenne

Unity Api ei ole ”säieturvallinen”, eikä sitä voi kutsua toisesta säikeestä sen sisäänrakennetun eston vuoksi. Erillisissä säikeissä ei ole siis mahdollista hyödyntää Unityn sisäisiä rakenteita, vaikka joitakin laskentoja voi suorittaa ohjelmointirajapinnan ulkopuolisesti. Seuraavaksi havainnollistetaan, miten Unity rakentuu erilaisista metodi- ja funktiokutsuista jolloin saadaan hyvä peruskäsitys koko pelimoottorin toiminnasta. [16.]

Tason latautuminen

Awake	Kutsutaan aina ennen *Start*-funktiota ja aina myös prefabin alustuksen jälkeen niissä peliobjekteissa, jotka ovat aktiivisia.
OnEnable	Kutsutaan vain, jos objekti on aktiivinen ja heti komponentin toiminnan alettua. Tämä tapahtuu, kun *MonoBehaviour*-instanssi on luotu, kuten kentän latauduttua tai peliobjektin instantoiduttua.

Ennen pelisilmukan käynnistymistä

Start	Kutsutaan vain, jos komentorivin instanssi on asetettu toimivaksi. *Start* -funktiota käytetään esimerkiksi muuttujien alustukseen ja muihin kertaluontoisiin operaatioihin kuten objektireferenssien hakuun.

Ruutujen välissä

OnApplicationPause	Kutsutaan ruudun lopussa, kun havaitaan pysähdys pelissä. Tämän jälkeen ajetaan vielä yksi ruutu, joka mahdollistaa pysähdyksen aikaisen kuvan piirron. Pysähdys saadaan aikaan asettamalla *Time.timeScale*-arvo nollaan.

Pelisilmukan rakenne

FixedUpdate	Tavallisesti kutsutaan useammin kuin *Update* funktiota. FixedUpdate-funktiota saatetaankin kutsua useamman kerran ruudun aikana, mikäli ruudunpäivitysnopeus on alhainen, tai ei ollenkaan, jos se on korkea. Fysiikan laskenta tehdään heti FixedUpdaten jälkeen, joten funktio on hyvä sijainti käsitellä pelissä tapahtuvaa liikettä ja *RigidBody*-objekteja, jolloin vältytään tarpeelta kertoa arvoja *Time.deltaTime*-arvolla, koska funktiota kutsutaan ruudunpäivitysnopeuden huomioivalla ajastimella tasaisin väliajoin.

Update	Kutsutaan kerran jokaista ruutua kohden. Täällä tapahtuu yleensä suurin osa toistuvista tapahtumista.
LateUpdate	Kutsutaan myös kerran jokaista ruutua kohden heti *Update*-funktion jälkeen. Updatessa tapahtuneet laskutoimitukset on laskettu tähän mennessä, ja *LateUpdatea* käytetäänkin esimerkiksi kolmannen persoonan kameran liikuttamiseen, jolloin varmistutaan, että kamera seuraa sulavasti.

Piirto ruutua kohden

OnPreCull	Kutsutaan ennen frustrum-karsintaa.
OnBecameVisible	Kutsutaan, kun objekti tulee näkyväksi kameralle.
OnBecameInvisible	Kutsutaan, kun objekti poistuu kameran näköpiiristä.
OnWillRenderObject	Kutsutaan kerran jokaista kameraa kohden, kun objekti tulee näkyviin.
OnPreRender	Kutsutaan ennen kuin kameran näkymää aletaan piirtää.
OnRenderObject	Kutsutaan, kun kenttä on piirretty. Käytetään myös, jos halutaan piirtää itsemääritettyä geometriaa GL-objektilla tai Graphics.DrawNewMeshNow-metodilla.
OnPostRender	Kutsutaan, kun kameran kuva on piirretty.
OnRenderImage(Pro)	Kutsutaan, kun kameran kuva on piirretty ja käytetään kuvan jälkiprosessointiin.

OnGui	Käytetään graafisen käyttöliittymän piirtämiseen ja joskus-käyttöliittymätapahtumien takia saatetaan kutsua useamsman kuin kerran ruudunpäivityksen aikana.
OnDrawGizmos	Editorin peli-ikkunan aikainen kutsu, jolla voidaan visualisoida erilaisia tapahtumia, kuten säteiden ampumista.

Objektin tuhoutuessa

OnDestroy	Kutsutaan objektin olemassaolon viimeisen ruudun aikaisten päivitysten jälkeen.

Ajon lopetus

OnApplicationQuit	Kutsutaan juuri ennen sovelluksen sulkemista, pelimoodista poistumista editorissa tai sovelluksen verkkosivun sulkemista.
OnDisable	Kutsutaan, kun **Behaviour**-luokan perivä objekti asetetaan toimimattomaan tilaan tai inaktiiviseksi.

Komentojen viivästäminen (Coroutinet)

Yield	Coroutine jatkuu seuraavan ruudunpäivityksen päivityskutsujen jälkeen.
Yield WaitForSeconds (aika)	Suorittaa toiminnot tietyn aikaviiveen jälkeen. Hyvä merkki tästä on *callback***-**funktio eli funktio, jota kutsutaan jonkin tapahtuman, kuten ääniefektin, soiton päätteeksi.
yield WWW	Suoritaa toiminnot verkkosivun latauduttua.

Yield StartCoroutine (funktio) Suorittaa toiminnot kyseisen funktion toteuduttua.

4 3D-grafiikka

Nykypäivän pelien ulkoasulle asetetaan jatkuvasti kovempia vaatimuksia. Varjoista, valoista ja materiaaleista halutaan yhä realistisempia. Resoluutiot kasvavat ja näytönohjainten muistit niiden mukana. Kuvan on pystyttävä tarjoamaan pelaajan valinnoille vastinetta sekä tilannetta havainnollistavaa ja tunnelmaa luovaa kuvaa. Riittävän immersion saavuttamiseksi virtuaaliympäristössä alustan täytyykin pystyä laskemaan muutokset ja piirtämään uusi kuva useita kymmeniä kertoja sekunnissa näytölle pelin lajityypistä riippuen sulavan pelikokemuksen takaamiseksi. Grafiikka ei suinkaan ole ainoa asia, joka täytyy päivittää useita kertoja sekunnissa. Pelilogiikka täytyy laskea yleensä vähintään yhtä monta kertaa kuin kuva yhden sekunnin aikana, äänistä puhumattakaan.

Nopeatempoisissa peleissä toivottava minimi uudelleenpiirtokertojen määrälle sekuntia kohden on noin 60. Hidastempoisissa peleissä uudelleenpiirrot voivat tuntua sujuvalta jopa alle 30 kuvaa sekunnissa toteutettuina. Tätä tapaa arvioida pelin sulavuutta kutsutaan FPS:ksi (Frames Per Second), ja se on ehkä tunnetuin mittayksikkö pelin raskaudelle suhteessa alustan laskentatehoihin.

Peleihin on kehitetty laskentatehon nostamisen lisäksi tekniikoita, joilla pystytään säästämään merkittävästi laskentatehoa ja jotka lopulta mahdollistavat nykyiset, visuaalisesti erittäin rikkaat virtuaalimaailmat. Jotta nämä luvussa 5 esiteltävät tekniikat ymmärrettäisiin, tutkitaan hieman tarkemmin 3D-grafiikan saattamista näytölle.

Vaikka kolmiulotteinen peli sijaitseekin kolmiulotteisessa avaruudessa, se täytyy kääntää tasaiselle, kaksiulotteiselle pinnalle, näytölle. Tarvittava informaatio kulkee aina pelimoottorista näytönohjaimeen ja sieltä kuvaruudulle. Tämän prosessin aikaisia tapahtumia kutsutaan piirtoliukuhihnaksi. Seuraavaksi käydään läpi perustietoa näytönohjaimesta sekä edellä mainitun prosessin päävaiheita.

4.1 Näytönohjain

Näytönohjain on kehitetty vapauttamaan prosessori erityisesti kuvan piirtoon liittyvästä laskennasta, ja se on näin ollen optimoitu toimittamaan tähän liittyvää laskentaa. Näytönohjain koostuu pääosin grafiikkaprosessorista, video-BIOS:sta, näyttömuististista, RAMDAC:sta ja omasta jäähdytysratkaisusta.

Grafiikkaprosessori eli GPU (graphical processing unit) on näytönohjaimen komponentti, jossa laskutoimitukset tehdään. GPU on erikoistunut liukulukujen laskentaan ja matriisien käsittelyyn.

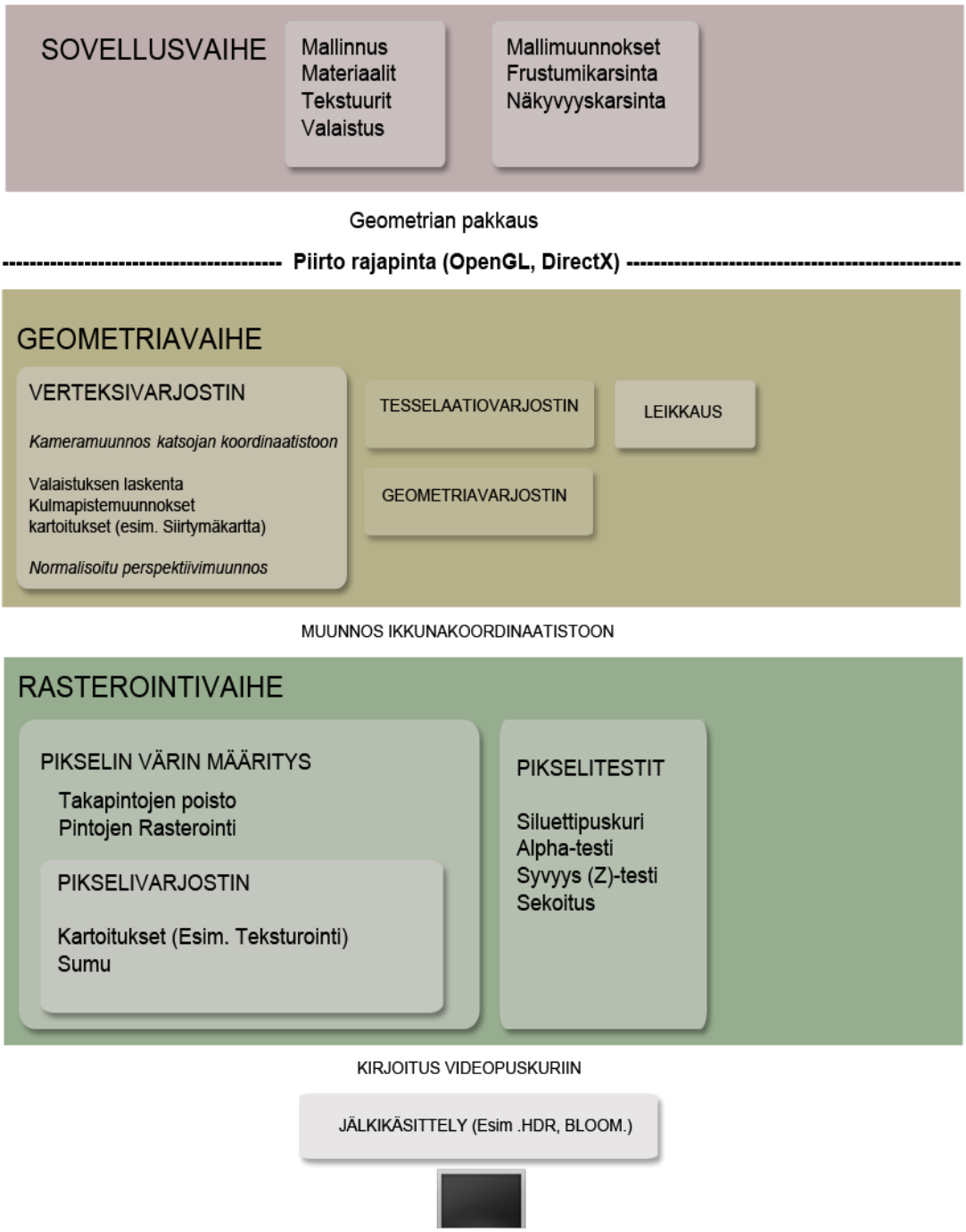
Video-BIOS toimii itse CPU:n ja GPU:n välisenä komentotulkkina ja usein myös sisältää tiedon muistin ajastuksesta, GPU:n ja välimuistin jännitteestä sekä kellotaajuudesta.

Näyttömuistia käytetään kuvadatan varastoimiseen näytönohjaimelle, mutta sitä voidaan hyödyntää myös Z-puskurin tallentamiseen. Z-puskuri käsittää kuvan syvyystiedon ja toimii yhtenä määrittäjänä sille, mitkä pikselit piirretään lopulliseen kuvaan. Z-puskuri käydään tarkemmin läpi luvussa 4.5.4. [17, s.225; 18.]

Näytönohjain on käytännössä tietokone itsessään oman prosessorinsa ansiosta ja sisältää näin ollen tietokoneen helpoimmin ylikuumenevan osan. Prosessorin ytimien lämpötilat voivat helposti nousta yli 100 celciusasteen ilman jäähdytystä samalla aiheuttaen pysyviä vaurioita. Näytönohjaimissa on CPU:n jäähdytykseen verrattava järjestelmä, joka tavallisesti muodostuu jäähdytyslevyistä, joiden tarkoitus on siirtää lämpöä pois piirilevystä ja tuulettimesta tai useammasta, jotka puhaltavat lämmön pois. [18.]

4.2 Piirtoliukuhihna

Piirtoliukuhihna kuvaa prosessia, jonka kuvainformaatio käy läpi muuntuakseen kuvaksi kuvaruudulle. Liukuhihnaksi sitä kutsutaan sen proseduraalisuuden vuoksi, eli vaiheet tapahtuvat peräkkäin ja edellisen vaiheen tulos on aina seuraavan syöte. Liukuhihnan yhtenä mainittavan etuna on, että sen läpi matkaavia dataklustereita, eli esimerkiksi verteksejä, voidaan käsitellä rinnakkaisesti, mikä nopeuttaa lopputuloksen syntymistä.



Kuva 7. Piirtoliukuhihna muodostuu sovellus-, geometria- ja rasterointivaiheesta.

Tämä prosessi voi erota hieman esimerkiksi grafiikkakirjastojen, kuten OpenGl:n ja DirectX:n, välillä, mutta pääpirteiltään se on sama. Niitä on kolme: sovellus-, geometria- ja rasterointivaihe (kuva 7).

4.3 Sovellusvaihe

Sovellusvaihetta ei aina mielletä osaksi piirtoliukuhihnaa. Se on enemmän alustus tiedolle ennen varsinaisten toimenpiteiden alkamista. Sitä ei voi ajatella samalla tavalla peräkkäisiksi tapahtumiksi kuin myöhäisempiä vaiheita. Sovellusvaihe alkaa halutun kuvan, tässä tapauksessa 3D-mallin, määrittämisellä. Kaikki virtuaalimaailman objektit sijaitsevat tavallisesti samassa koordinaatistossa, mutta jokaisella objektilla on oma objektiavaruutensa eli koordinaatistonsa, jossa sen geometrian kuvaavat verteksit eli kulmapisteet on määritelty. Yhteistä koordinaatistoa kutsutaan maailmanavaruudeksi, jonne paikallismallit lopulta sijoitetaan.

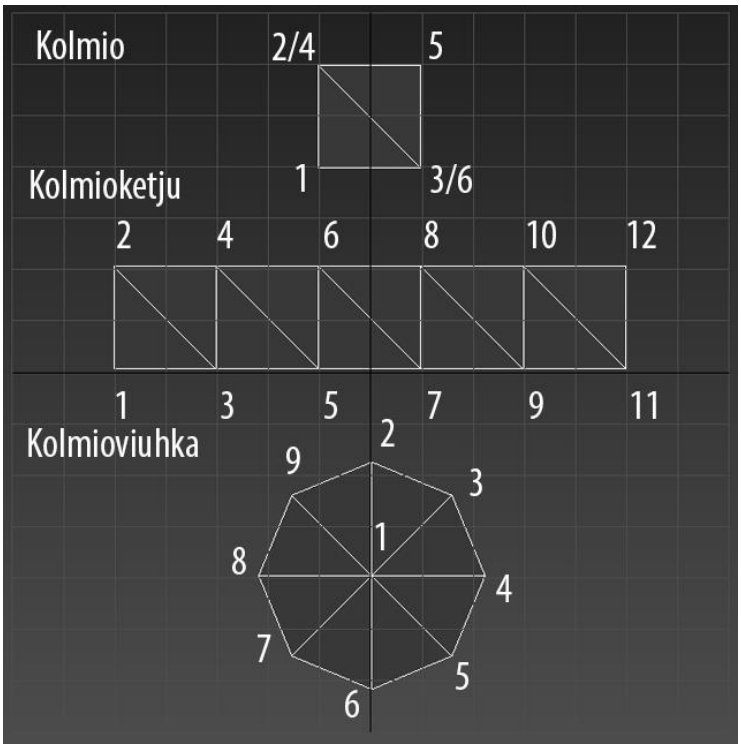
Raskaiden mallien kolmioverkkoja saatetaan yksinkertaistaa suhteessa niiden etäisyyteen kamerasta. Jos malli sijaitsee kaukana kuvaruudun sijainnista, se ei tarvitse kaikkia yksityiskohtiaan, joten mallia voidaan karsia yksinkertaisemmaksi kuvanlaadun siitä juurikaan kärsimättä. Puut ovat hyvä esimerkki tästä. Kaukana olevat puut usein kutistetaan jopa neliöpinnaksi (Billboard) asti. Geometrian vähentämistä tällä tavoin kutsutaan LOD:ksi eli Level of Detail. Samankaltaista operaatiota käytetään myös tekstuurien tarkkuuden vähentämiseen. Suodatusta kutsutaan MIP-kartoitukseksi (Multum in Parvo eli ”paljon vähässä”). MIP-tekniikassa tekstuurikuvasta luodaan aina resoluutioiltaan puolet toista pienempiä valmiskuvia, joita käytetään täysiresoluutioisen tekstuurin tilalla suhteessa tekstuuripinnan etäisyyteen kamerasta. MIP-tekniikan tavoite on valita tekstuuri, jonka resoluutio on suhteessa pikseliresoluutioon, joka on lähellä Nyqvistin taajuutta (kuvaruudun pikselien määrän tulisi olla vähintään kaksi kertaa tekstuurikuvan tekseleiden määrä). [18, s. 164–167; 20.]

Ennen näytönohjaimeen vientiä 3D-mallien polygonit pilkotaan viimeistään grafiikkakirjastorajapinnassa (OpenGL, DirectX) kolmioksi. Kolme pistettä muodostavat aina yksiselitteisen tason. Neljä kulmapistettä tarjoaa useamman mahdollisen skenaarion näiden välisen pinnan piirtoon. Itse mallinnustilanteessa pyritään usein nelikulmioista koostuviin malleihin. Nelikulmiot toimivat usein paremmin mallinnusohjelmien työkalujen

kanssa. Ne mahdollistavat mallien rakentamisen niin, että ne kestävät parhaiten deformaatioita. Kasvonpiirteitä animoitaessa nelikulmiodun mallin edut nousevat esille [18, s. 164–167; 20].

Kolmioverkkopintojen esitystapoja

Näytönohjaimella on useita tapoja tulkita sille lähetettyä geometrista tietoa (kuva 8). Seuraavassa käydään läpi kolme yleisintä tapaa.



Kuva 8. Kolmio, kolmioketju ja kolmioviuhka ovat kolmioverkkojen esitystapoja.

Kolmio-piirtotekniikka edustaa kaikkein yksinkertaisinta tapaa tulkita koordinaattitaulu. Siinä jokainen kolmio määräytyy kolmesta koordinaatista. Jokainen kolmio siis tulkitaan erillisenä pintana, ja vaikka visuaalisesti viereisten kolmioiden jakaessa jokin koordinaatti kulmapisteidensä kesken, on kyseinen koordinaatti määritetty erikseen jokaiselle kolmiolle. Jos mietitään kuution muotoista primitiiviä, tarvitaan sen esittämiseen tällä tulkinnalla kuusi koordinaattia sivua kohden eli yhteensä 36 erillistä koordinaattia. [20; 22, s. 82–84.]

Kolmioketjussa vain ensimmäisen kolmion jokainen kulmapiste annetaan, minkä jälkeen jokainen kolmio jakaa edellisen kanssa kaksi kulmapistettä, mikä johtaa siihen, että uuden kolmion määrittäminen ketjuun vaatii vain yhden lisäpisteen. Tason näkyvä puoli määräytyy ensimmäisten pisteiden piirtojärjestyksen mukaan. Esimerkiksi OpenGL:ssä piirtojärjestys on vastapäivään. Jos edellisen kohdan kuutioesimerkkiä mietitään kolmioketjutulkintana, tarvitaan vain 24 erillistä koordinaattia. [18, s. 52–53; 20; 22, s. 82–84.]

Kolmioviuhkapiirrossa ensimmäinen piste on yhteinen kaikkien kolmioiden kanssa ja muodostaa näin viuhkan keskipisteen. Muuten määritys on sama kuin kolmioketjulla. Edelliset koordinaattitiedon tulkintamenetelmät vähentävät turhaa toistoa, jota syntyy edelleen esimerkiksi päällekkäisten kolmioketjujen väliin. Ratkaisuksi voidaan tarjota esimerkiksi pistedatan indeksointia, jolloin jokainen piste saa oman uniikin viittauksensa. Kuutio saataisiin piirrettyä viuhka-tulkinnalla vain 16 erillisellä koordinaatilla, mutta tätä ei tavallisesti käytetä, koska kulmapisteiden määrä ei yksinkertaisesti riittäisi kunnolliseen tekstuurien ja valaistuksen laskemiseen [18, s. 52–53; 20; 22, s. 82–84].

Mallista halutaan mahdollisimman yhtenäinen, jotta yllä mainituista kahta jälkimmäistä voitaisiin hyödyntää mahdollisimman paljon. Yleensä pyritään yhtenäiseen kolmioverkkoon, vaikka erillisistä osista koostuvassa malleissa saattaisikin olla vähemmän kolmioita. Tilanne kuitenkin vaihtelee mallikohtaisesti, riippuen siitä, kumpi lopussa tarjoaa paremman suorituskyvyn. Erillisistä osista mallintaminen on kuitenkin helpompaa ja lopputulos siistimpi.

Kulmapistetieto ei ole ainoa asia, joka välitetään piirtoliukuhihnaan. Näihin liittyvää oheistietoa, kuten tekstuurikoordinaatteja, väri- ja normaalivektoreita sekä valaistusinformaatioita lähetetään myös ohessa. Mikäli kyseessä on animoitava malli, valitaan vielä sovellusvaiheessa haluttu ruutu animaatiosta. Mahdolliset fyysikan simuloinnista aiheutuvat muutokset lasketaan ennen geometriavaihdetta.

Lopuksi ylimääräistä tietoa yritetään karsia frustrumikarsinnalla eli jättämällä näkökentän ulkopuolelle tai toisten kappaleiden taakse piiloon jäävät mallit huomiotta. Kulmapisteet ja niihin liittyvä tieto on tavallisesti eriteltynä omissa taulukoissaan erilaisissa tiedosto-

muodoissa, mutta ne voidaan vielä ennen näytönohjaimeen saattamista yhdistää yhdeksi taulukoksi, johon määritetään erilaisia lukuharppauksia eri tietojen mukaisesti. [18, s. 188.]

4.4 Geometriavaihe

Geometriavaihe on ensimmäinen varsinainen piirtoliukuhihnan vaihe. Sen pääasiallinen tarkoitus on luoda piirrettävästä kuvasta lopullinen rautalankamalli, joka myöhemmin rasteroidaan. Geometrian muunnokset kohdistuvatkin vertekseihin leikkausoperaatiota lukuun ottamatta, joka käsittelee kokonaisia primitiivejä. [18.]

Koordinaatistomuunnosvaihe

Geometriavaiheen alussa frustumkarsinnan jälkeen jättämä maailmankoordinaatisto siirretään kameran eli katsojan koordinaatistoon koordinaatistomuunnoksella.

Katsojan koordinaatisto on tavallisesti oikeakätinen, eli x-akseli osoittaa katsojasta oikealle, y-akseli ylöspäin ja z-akseli katsojaa kohden. Katsojan eli kameran sijainti avaruudessa on yleensä määritetty pisteenä, jolla on katsomissuunta ja sen normaalin suuntainen ”ylös”-suunta. Katsojan koordinaatiston ”ylös”-suunnasta saadaan normalisoimalla katsojan koordinaatiston y-suuntainen yksikkövektori. [18, s. 173–175; 19.]

$$\rightarrow u_y = \frac{\rightarrow u}{\left(\rightarrow u\right)}$$

Täten katsomissuunnan negaatiosta saadaan z-suuntainen yksikkövektori:

$$\rightarrow u_z = \frac{-\rightarrow v}{\left(\rightarrow v\right)}$$

Edellä mainittujen ristitulosta saadaan puuttuva x-suuntainen yksikkövektori:

$$\rightarrow u_x = \rightarrow u_y \times \rightarrow u_z$$

Muunnosmatriisi katsojan koordinaatistoon saadaan asettamalla edellä johdetut yksikkövektorit matriisin riveiksi:

$$U = \begin{pmatrix} u_{x,1} & u_{x,2} & u_{x,3} & 0 \\ u_{y,1} & u_{y,2} & u_{y,3} & 0 \\ u_{z,1} & u_{z,2} & u_{z,3} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Lopullinen katsojan koordinaatistosta käännetty maailmankoordinaatisto saadaan kertomalla jokainen verteksi muunnosmatriisilla. Jos verteksien mukana tulee normaalitietoa, täytyy sekin muuntaa samalla, ortonormaalisella muunnosmatriisilla. Näin syntyvässä koordinaatistossa katsoja on origossa. [18, s. 173–175; 19.]

Mikäli käytössä on reaaliaikainen valaistus, voidaan tässä vaiheessa laskea kulmapisteiden väriarvoihin niiden valoisuus. Valoisuusarvot saadaan selville tuntemalla kulmapisteen normaalivektori, valonlähteiden sijainnit ja ominaisuudet sekä pinnan ominaisuudet, kuten heijastuneisuus kohti katsojaa. [18, s. 177.]

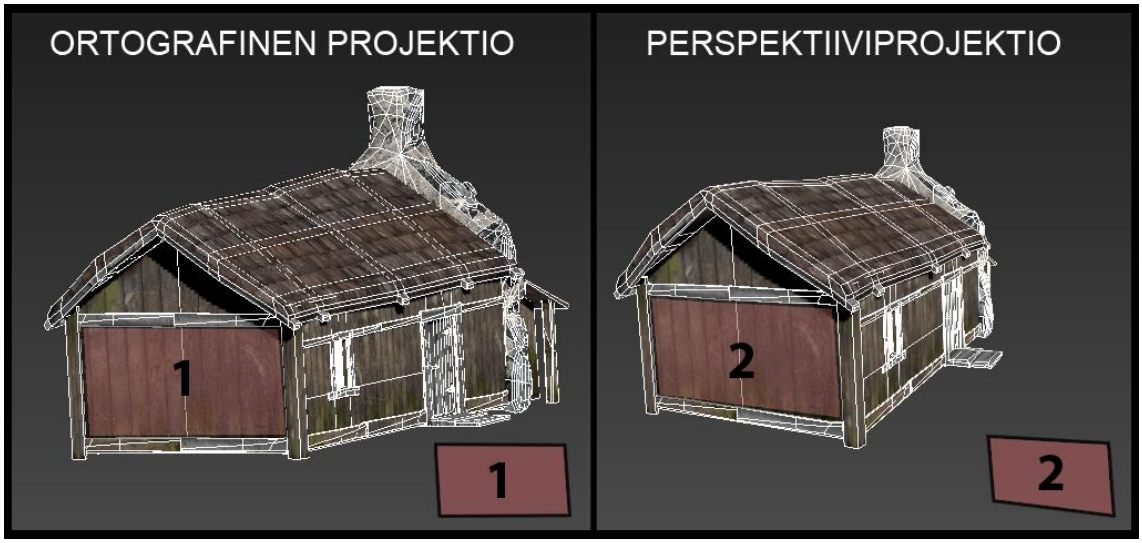
Projektiot

Kolmiulotteisesta näkymästä saadaan kaksiulotteinen kuva projisoimalla se johonkin pintaan. Yleisin pinta tietokonegrafiikassa on taso. Kuvan projisoimiseen tasoon 3D- grafiikassa käytetään pääasiassa kahta erilaista projektiota: ortografista ja perspektiiviprojektiota.

Ortografinen projektio on paralleeliprojektion alaluokka. Ortografisessa projektiossa kuvalla ei ole perspektiiviä. Tällöin ei synny illuusiota objektien välisestä etäisyydestä. Näiden mittasuhteet pysyvät etäisyydestä riippumatta yhtenäisinä ja vakioina. Matemaattisesti voi ajatella kuvan jokaisesta kulmapisteestä vedettävän samansuuntaisen projektiosuoran johonkin projisoitavaan tasoon, tason normaalin suuntaisesti. Ortografinen projektio on laskennallisesti halvempi vaihtoehto. Sen saavuttaminen vaatii yksinkertaisimmillaan vain projektion suuntaisen pääakselin koordinaattiarvojen pudottamisen eli yleensä z-akselin. Ortografinen projektio pisteestä (x, y, z) olisi (x, y) ja kuvasta katoaisi illuusio syvyydestä.

Mikäli projektio ei ole minkään pääakselin suuntainen, on kyseessä aksonometrinen projektio. Aksonometrisestä projektiosta on vielä erikoistapaus: isometrinen projektio, jossa projektion suunta muodostaa yhtä suuren kulman jokaisen projisoitavan pääakselin kanssa. Tätä käytetään usein, kun maailmaa tarkastellaan lintuperspektiivistä. Se oli tietokonepelien alkuaikoina hyvinkin yleinen projisointitapa kuvata peliä. Jonkin muun kuin pääakselien suuntaisissa projektioissa ortografinen projektio saadaan kiertämällä mallia, kunnes projektiosuunta kiertyy z-akselin suuntaiseksi. Kun otetaan katsomissuunnaksi projektiosuunta ja katsojan yläsuunnaksi kuvan yläsuunta, voidaan kiertää malli aikaisemmin esitetyllä koordinaatistomuunnosmatriisilla, jolloin projektio muuntuu z-akselin suuntaiseksi projektioksi.

Perspektiiviprojektio on laskennallisesti hieman kalliimpi vaihtoehto, mutta se tarjoaa realistisemman projektion, joka ottaa kuvassa esiintyvien mallien koon huomioon suhteessa etäisyyteen katsojasta. Kauempana olevat asiat näyttävät pienemmiltä kuin kuvan etualalla olevat. Samansuuntaisiksi miellettävät suorat eivät ole enää samansuuntaisia, pois lukien täsmälleen kuvan tason kanssa samansuuntaiset viivat, kuten ortografisessa projektiossa, vaan ne leikkaavat kuvitteellisessa katoamispisteessä toisensa. Projisoitaessa pisteitä tasolle ne eivät myöskään leikkaa projektiotasoa ortografisen projektion tavoin suorassa linjassa, vaan projektioviivat kohdistuvat katsojan sijaintiin, jonka voi mieltää sijaitsevan seuraavaksi käsiteltävän näköfrustumi-pyramidin huipulla. Alla kuvassa 9 havainnollistuvat näiden kahden projektion eroavaisuudet. Talo on kuvattu samasta kohtaa ortografisella projektiolla ja perspektiiviprojektiolla. Punaisen nelikulmion (1, 2) eroavaisuudet paljastavat, miten projektiot vaikuttavat kuvaan. [18, s. 188–189.]



Kuva 9. Ortografisen ja perspektiiviprojektion eroavaisuudet.

Seuraavaksi tarkastellaan perspektiiviprojektiomatriisin luontia.

Perspektiiviprojektion muunnos pisteelle:

$$x = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

projektiotason pisteeksi x_p tapahtuu jakamalla jokaisen akselin arvo pisteen syvyysakselin (z) arvon ja kameran etäisyyden projektiotasosta osamäärällä. Z-arvo on negatiivinen, koska yleensä oletuskoordinaatistona on oikeankäden koordinaatisto ja kameran suunta on negatiivisen z-akselin suuntainen.

$$x_p = \begin{bmatrix} \frac{x}{(-z/d)} \\ \frac{y}{(-z/d)} \\ \frac{z}{(-z/d)} \end{bmatrix} = \begin{bmatrix} \frac{x}{(-z/d)} \\ \frac{y}{(-z/d)} \\ -d \end{bmatrix} \quad d = \textit{katsojan etäisyys projektiotasosta.}$$

josta voidaan luoda seuraavanlainen muunnosmatriisi:

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/d & 0 \end{bmatrix}$$

Vaikka projektion jälkeen kuva on lopullisessa muodossaan kaksiulotteiselle tasolle projisoituna, säästetään siitä silti yleensä syvyysarvot, eivätkä ne säily edellä esitetyllä menetelmällä. Kun syvyysarvot säästetään, perspektiiviprojektiomuunnos aiheuttaa näköfrustumin muuntumisen suorakulmaiseksi särmiöksi, eli projektioviivat tasolle muuntuvat z-akselin suuntaisiksi, aivan kuin ortografisessa projektiossa, ja lopullinen projektio tasolle tapahtuu myös samalla tavalla. Tällöin ei siis tapahdu siirtymää kolmiulotteisesta kuvasta kaksiulotteiseen vaan kolmiulotteisesta kolmiulotteiseen. [18, s.190–191; 23.] Tällaisen syvyystiedon säästävä muunnosmatriisi näyttää seuraavalta:

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & f+n/n & f \\ 0 & 0 & -1/n & 0 \end{bmatrix} \quad \begin{array}{l} f = \textit{takaleikkaustason etäisyys katsojasta} \\ n = \textit{etuleikkaustason eli projektiotason etäisyys katsojasta} \end{array}$$

Perspektiiviprojektio normalisoidaan, jolloin saadaan origossa sijaitseva kuutio, jonka sivujen leveys ja korkeus on 2 ja kaikki koordinaatit sijaitsevat akseleiden 1 ja -1 välissä.

Tällöin lopullisessa näköalueen rajauksessa ei tarvitse tuntea frustumin taka- tai etuseinämän todellisia koordinaatteja tai ikkunan dimensioita. [18, s. 192; 18.]

Normalisointi tehdään seuraavanlaisella matriisilla:

$$M = \begin{bmatrix} \frac{1}{w} & 0 & 0 & 0 \\ 0 & \frac{1}{h} & 0 & 0 \\ 0 & 0 & \frac{2}{(f-n)} & \frac{(f+n)}{(f-n)} \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{matrix} w = \textit{etäisyys ikkunan reunasta kuvaikkunan keskipisteeseen vaakassuunnasa} \\ h = \textit{etäisyys ikkunan reunasta kuvaikkunan keskipisteeseen pystysuunnassa} \end{matrix}$$

Näin saadaan lopullinen, normalisoiva perspektiiviprojektiomatriisi:

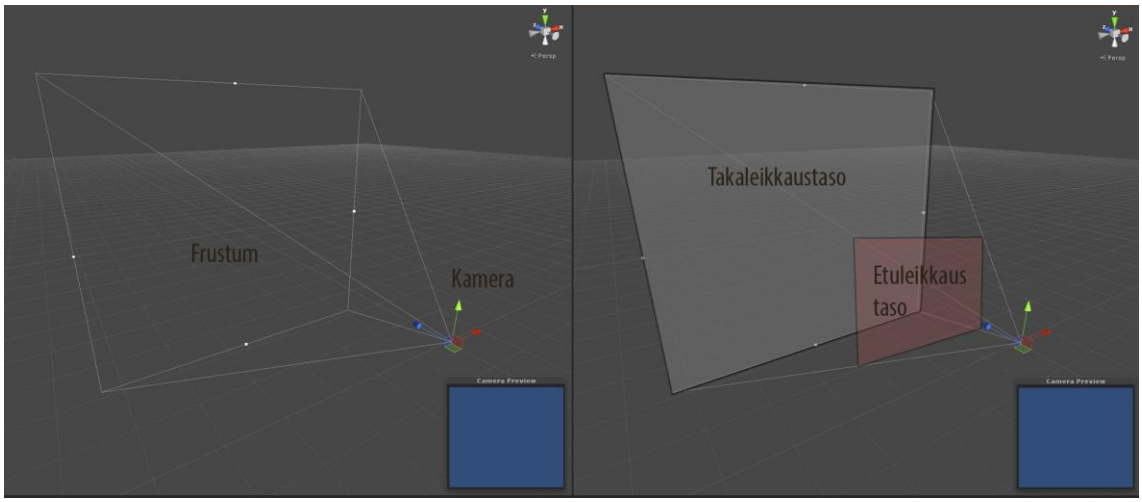
$$N_p M = \begin{bmatrix} 1/w & 0 & 0 & 0 \\ 0 & 1/h & 0 & 0 \\ 0 & 0 & -\frac{(f+n)}{n(f-n)} & -\frac{2f}{(f-n)} \\ 0 & 0 & \frac{-1}{n} & 0 \end{bmatrix}$$

Syvyysarvoja tullaan käyttämään toisten objektien taakse jäävien objektien tai objektien osien hylkäämiseen. Tähän tullaan käyttämään Z-puskuria eli syvyyskarttatekniikkaa rasterointivaiheessa. [18, s.193–194; 22.]

Näköfrustumi

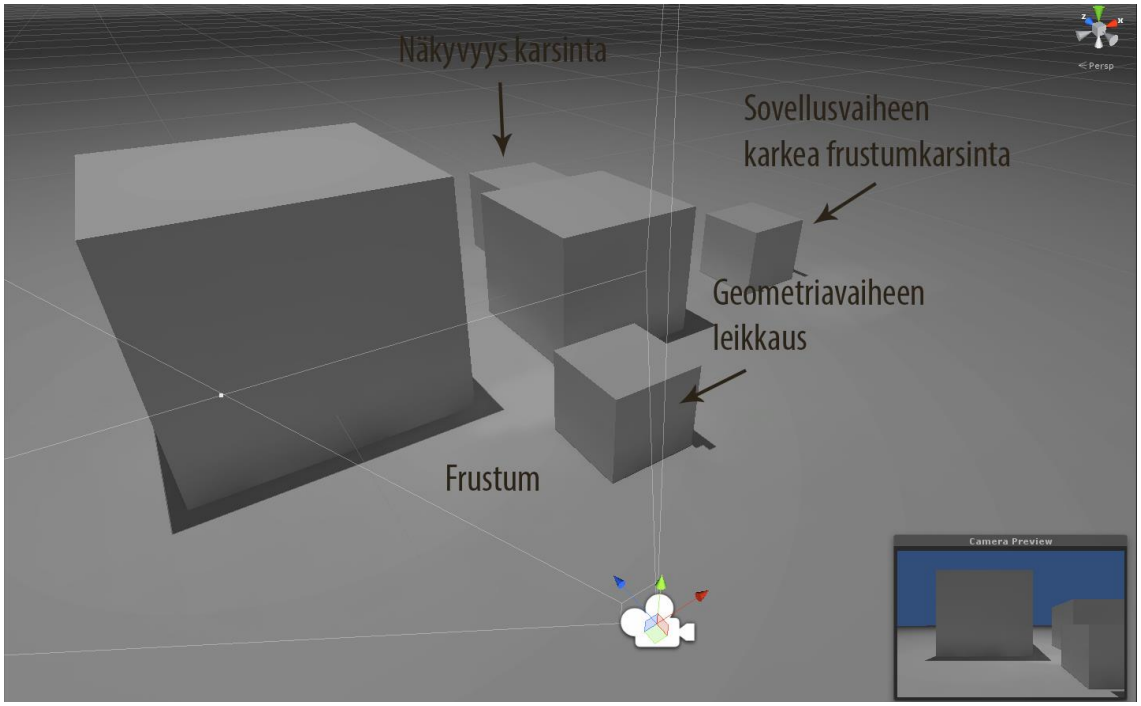
Näköfrustum määrittelee rajat katsojan näkymälle näin rajaten ne virtuaalimaailman objektit, joita piirtoliukuhihna käsittelee jokaista piirtoa kohden. Näköfrustumin muotoon vaikuttaa kameran eli katsojan linssi, mutta muoto on yleensä pyramidimainen. Näköfrustum jatkuisi äärettömyyteen katsojan eteen, mutta on turhaa laskea liian kaukaista avaruutta, koska kaukaisuuteen sijoittuvat objektit ovat yleensä merkityksettömiä. [24,s.132–133]

Näköfrustumille määritellään takaleikkaustaso, joka toimii näkökentän maksimietäisyytenä, ja eturaja, joka rajaa liian lähellä olevat objektit ja katsojan taakse jäävät objektit pois ja toimii yleensä myös projektiotasona. Tätä on havainnollistettu kuvassa 10.



Kuva 10. Frustumi ja leikkaustasot.

Sovellusvaiheessa suoritettaan jo karkea frustumikarsinta. Tuolloin karsittiin kaikki kuvasta varmasti pois jäävä. Unityssä frustumikarsinnan nopeuttamiseksi on jokaiselle pelimallille määritetty osuma-laatikko, joka kuvastaa mallin kokoa hyvin pelkistetyllä tasolla mutta jolla pystytään nopeasti määrittämään, kuuluuko malli piirrettävään kuvaan. Kuvaan saattoi kuitenkin jäädä esimerkiksi osittain näkyvissä olevia objekteja, joista voidaan karsia geometriavaiheessa primitiivitasolla osia pois. Eri karsinnat on esitetty kuvassa 11. Normalisoidun frustumin ulkopuolelle jäävät alueet leikataan Cohen-Sutherlandnimistä algoritmia hyväksikäyttäen. [18, s. 195.]



Kuva 11. Karkea frustumikarsinta, näkyvyyskarsinta ja geometriavaiheen leikkaus.

Karsintojen avulla rasteroitavan pinnan määrä vähenee ja piirtoliukuhihna nopeutuu.

4.5 Rasterointivaihe

Takapintojen poisto eli Backface culling

Laskennallistehokkuuteen liittyvistä syistä 3D-mallin kolmiopinta on tavallisesti näkyvä eli rasteroituva vain kolmiopinnan toiselta puolelta ja sillä on vain tällä puolella normaalivektorit. Tämä mahdollistaa katsojasta poispäin osoittavien pintojen poiston ennen rasterointia, mikä säästää näin laskentaa jokaista kuvaa kohden ja toimii näin useimmissa tilanteissa täysin kuvaa köyhdyttämättömänä optimointina.

Poispäin osoittavien kolmioiden poisto tehdään yksinkertaisesti vähentämällä katsojan vektori jostain pinnan pisteestä ja laskemalla erotuksen ja pinnan normaalivektorin välinen pistetulo. Riippuen siitä, käytetäänkö myötäpäiväistä vai vastapäiväistä piirtojärjestystä, saatavan pistetulon etumerkki kertoo, poistetaanko kolmiopinta. [22, s. 71–72.]

Monikulmion rasterointi

Geometriavaiheen jälkeen frustumi on projisoitu kuvaruudun koordinaatistoon, mutta kulmapisteiden väleissä ei ole vielä pintaa. Rasterointivaiheen tärkeimpänä tarkoituksena voitaisiinkin pitää pinnan luontia ja lopullisten pikselien väriarvojen määritystä. Rasterointi tehdään vaakarivitäytöllä, jolloin täytettävät vaakarivit määritetään monikulmioiden pystysuuntaisten reunojen koordinaatteja vaakarivin rajoina käyttäen. [18, s. 202.]

Siluettipuskuri

Siluettipuskuria käytetään esimerkiksi maskeeraamaan haluttu piirrettävä alue ruudulta. Siluettipuskuri on kuvaruudun kokoinen, laitteistotasolla rakentuva bittikartta, jossa tavanomaisesti on vähemmän bittejä käytössä jokaista pikseliä kohden. Maskeeraus tehdään asettamalla bittikartan pikselien arvoiksi 1 kohdista, joista halutaan pikselit piirtää ruudulta. Pikseleihin, jotka halutaan sulkea pois piirrosta, asetetaan arvoksi 0. Tällainen oli erittäin hyödyllistä esimerkiksi vanhoissa lentosimulaattoreissa, joissa pelin muuttuva

maailma sijaitsi ikkunan takana, mutta ikkunaa ympäröivä ohjaamo pysyi melko muuttumattomana. [20.]

Z-puskuri ja Alpha-testi

Z-puskuri on yksinkertaisesti piirrettävän kuvan kokoinen matriisi, jonka jokainen alkio sisältää vastaavan pikseliin piirrettävän pinnan etäisyyden kamerasta eli syvyysarvon. Kun kuvaan piirretään seuraavaa objektia, tehdään objektin pikseleille z-testi eli verrataan sen syvyysarvoja syvyyskartan vastaavaan. Jos uuden objektin syvyysarvo osoittaa sen sijaitsevan lähempänä kuin aikaisemman, se korvaa senhetkisen pikselin arvon ja päivittää vastaavan alkion syvyyskartasta. Jos uuden objektin pinnan pikseli sijaitsee kauempana kuin syvyyskartan vastaava, se hylätään. Syvyyskarttaa voi myös hahmottaa mielessään ajattelemalla sitä piirrettävän kuvan mustavalkoisena valoisuuskarttana. Tummemat pikselit jäävät taakse ja valoisammat tulevat eteen. [25.]

Alpha-testissä tarkastellaan pikselien läpikuultavuus- eli alpha-arvoja ja tarpeeksi läpinäkyvät pikselit voidaan hylätä piirrosta ja z-puskuriin niiden kohdalle jää ”reikä”. Tämä mahdollistaa myöhemmin piirrettävien, mutta taaempana olevien pikselien piirron. Alpha-testi on tehokas tapa luoda esimerkiksi ritilöitä, kaltereita ja muita geometrialtaan reiällisiä pintoja pelkän bittikartan avulla. [20.]

Sekoitus- ja videopuskuri

Pikselin läpäistyä kaikki edellä mainitut tekniikat se viedään videopuskuriin, joka edustaa lopullista, ruudulla näkyvää kuvaa.

Tavallisesti pikseli ylikirjoittaa suoraan edellisen arvonsa, mutta joissain tapauksissa halutaan kerrostumista, kuten mikäli objektilla on useampi tekstuuri, jotka kuultavat toistensa läpi. Tässä tapauksessa edellisen värin päälle lisätään uusi väri, joka ensin painotetaan alpha-arvolla. [18, s. 228.]

Peleistä puhuttaessa videopuskureita on yleensä kaksi, joista toista piirrettäessä näytölle toista jo valmistellaan, jolloin vältytään kuvan välkkymiseltä.

4.6 Varjostimet

Varjostimet ovat liukupiirtohihnaan sijoitetttuja, ohjelmoitavia pienoisohjelmia, jotka määrittävät itse kuvadatan käsittelyn. Varjostimia on neljää erilaista, joista ensimmäinen, toinen ja kolmas, verteksi-, geometria- ja tesselaatiovarjostin sijaitsevat liukupiirtohihnan geometriavaiheessa, ja neljäs, pikselivarjostin, joka sijaitsee rasterointivaiheessa. Varjostimien kirjoittamiseen käytetään siihen erikoistuneita ohjelmointikieliä, joita ovat GLSL OpenGL-kirjastoa käytettäessä ja HLSL-DirectX-kirjastoa käytettäessä (HLSL ei ole suoraan tuettu Unityssä). Näiden lisäksi on kolmansien osapuolien kieliä, jotka kääntyvät edellä mainituiksi. Niitä ovat esimerkiksi HLSL:ää muistuttava, Nvidian kehittämä Cg ja Unityssä käytettävä ShaderLab. Varjostimet mahdollistavat edellisessä kappaleessa käytettävien kartoitusten käytön ja huomioon ottamisen laskennassa. [16; 20.]

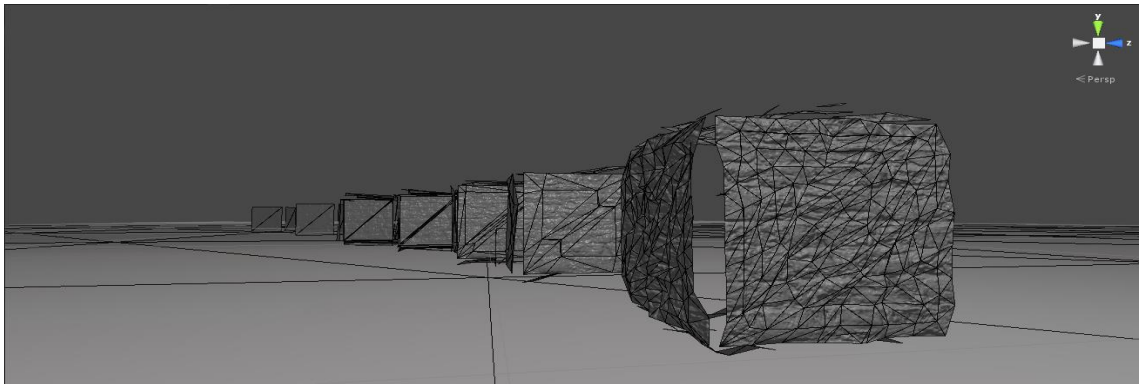
Verteksivarjostin käsittelee piirrettävän kuvan mallia kulmapistetasolla. Verteksivarjostin ottaa parametrikseen yhden kulmapisteen. Niillä voidaan vaikuttaa esimerkiksi kulmapisteen koordinaatteihin, väriin, sen valaistukseen ja tekstuuriin. Hyviä käyttökohteita ovat esimerkiksi luujärjestelmistä johtuvat deformaatiot mallin pinnassa, kuten kasvojen uurteiden ja ryppyjen vaihtelu ilmeen perusteella, veden pinnan aaltoilu ja väreily. Verteksivarjostimen jälkeen kuvadata jatkaa matkaansa liukupiirtohihnassa kohti pikselivarjostinta tai mikäli käytössä on geometriaa muokkaavat geometriavarjostimet, kohti niitä. [26.]

Tesselaatiovarjostimet ovat tulleet vaihtoehtoiseksi osaksi liukupiirtohihnaa OpenGl 4.0:n ja DirectX 11:n myötä. Tesselaatiolla voidaan jakaa primitiivit useammaksi primitiiviksi reaaliajassa. Käyttöesimerkkinä voidaan esimerkiksi ottaa tesselaation yhdistäminen siirtokarttaan. Lähestyttäessä mallia voidaan lisätä pinnan tiheyttä, jolloin siirtymäkartta saa tarpeeksi kulmapisteitä yksityiskohtaisempaan esitykseen. Piirtoliukuhihnan tesselaatio-osuus on kolmiosainen, joista kaksi osaa on ohjelmoitavissa.

Tesselaation hallintavarjostin määrittää ensimmäisessä vaiheessa, kuinka tiheäksi primitiivi jaetaan. Se pystyy lisäämään tai vähentämään kulmapisteitä, mutta ei tuhoamaan niitä. Hallintavarjostin pitää myös huolen, että tesseloitujen primitiivien väleihin ei synny epäkonsistenttisuutta, kuten halkeamia. [20.]

Tesselaation primitiivien tuottaja ottaa hallintavarjostimen syötteen ja jakaa primitiivin, mikäli hallintayksikkö on määritetty, hallintayksikön määrittämien parametrien mukaan tai oletusarvoilla. [20.]

Tesselaation evaluointivarjostin vastaanottaa abstraktit koordinaatit primitiivigeneroinnista ja hallintavarjostimen tulosteen ja muokkaa lopulliset arvot kulmapisteille. Uudet koordinaatit, normaalit ja tekstuurikoordinaatit luodaan täällä. Evaluointivarjostin on pakollinen osa tesselaatiota, ja mikäli sitä ei ole määritetty, tesselaatio ei toteudu. Tesselaatiota testattiin peliprojektissa lyhyesti. Kuution pinnan monikulmioverkon tiheys skaalautuu suhteessa mallin etäisyyteen kamerasta, jolloin voidaan esittää siirtymäkartalla yksityiskohtaista pintaa mallin vaikuttaessa merkittävästi kuvassa, kun taas kauempana ollessa riittää vain pelkistetty malli, koska yksityiskohdat eivät erottuisi muutenkaan (kuva 12). [20.]



Kuva 12 Tesselaatiovarjostintesti.

Geometriavarjostin on esitellyistä kolmesta varjostimesta ainoa valinnaisesti tapahtuva ohjelma. Se sijaitsee liukupiirtohihnalla verteksivarjostimen, tai mikäli tesselaatio on käytössä, sen jälkeisenä varjostimena ja ottaa parametrina yhden primitiivin ja palauttaa nolla tai useamman primitiivin. Geomeriavarjostin kykenee siis poistamaan kolmioita tai lisäämään niiden määrää sekä myös toimimaan verteksitasolla. Se pystyy myös muuttamaan primitiivien tyyppiä, kuten kolmioita viivoiksi tai pisteiksi. [20.]

Pikselivarjostin käsittelee pisteiden välisiä pintoja pikselitasolla, verteksivarjostimen käsitellessä piirrettävän kuvan mallien kulmapisteitä. Verteksivarjostin ajetaan kerran jokaista kulmapistettä kohden, kun taas pikselivarjostin ajetaan kerran jokaista kuvaruudun pikseliä kohden. Tämä johtaa valtavaan työmäärään jokaista kuvaruutua kohden. Näytönohjaimissa on kuitenkin huomioitu tämä, ja pikselivarjostinten tehokkaaseen toi-

mintaan onkin panostettu esimerkiksi verteksivarjostinta enemmän. Pikselivarjostin toimii myös käsittelijänä suurimmalle osalle erilaisista, luvussa 5.2 tarkemmin esiteltävistä kartoitustekniikoista. Esimerkiksi tekstuurin väri asetetaan pikseliin tässä vaiheessa, samoin normaalikartan luomat valaistusmuutokset. [20.]

5 3D-mallinnus pelimaailmaan

5.1 3D-pelimallinnus ja 3D-lasermallit ympäristön visualisoinnissa

Pelimallinnus lienee yksi haastavimpia mallinnuskohteita. Siinä ei ainoastaan visuaalinen ilme ole tärkeä, vaan myös teknisestä puolesta on oltava vahva ote. Animaatioelokuvissa ja muissa kohteissa, joissa mallinnusta hyödynnetään, ei reaaliaikaisuus ole painolastina, kun kuvat voidaan piirtää etukäteen. Mallit saavat olla niin tarkkoja, kuin taiteilijan mielestä on tarpeen. Pelimaailmassa taas joudutaan turvautumaan jokaiseen oljenkorteen, jolla laskennan määrää saadaan vähennettyä jokaista pelisykliä kohden responsiivisuuden takaamiseksi. Miljoonien polygonien 3D-mallit eivät ole vaihtoehto, ja monikulmioverkkojen tuleekin koostua vain informaatioltaan tarpeeksi merkittävistä kulmapisteistä. On kehitetty erilaisia tekniikoita, joilla saadaan korkeatasoista grafiikkaa laskennallisesti huomattavasti edullisemmin keinoin. Pelimaailmassa esiintyy myös animaatiota, ja siitä aiheutuvien kolmioverkon deformaatioiden takia mallin kolmioinnilla on suuri merkitys lopputuloksen kannalta.

Luotaessa pelimaailmaan malleja pyritään pinnan geometriassa mahdollisimman optimaalisiin ratkaisuihin, jotta saataisiin mahdollisimman yksityiskohtaista ja tehokasta jälkeä. Pelimallien tulisi olla mahdollisimman yhtenäisiä, jotta koordinaattidata saataisiin pakattua mahdollisimman pieneen tilaan ja hyödyttömältä toistolta vältyttäisiin. Jossakin tilanteissa tietenkin tämän säännön seuraaminen johtaa niin monen lisäkolmion syntyyn, että sitä voisi kutsua optimoinniksi. Mallien kolmiomäärää vähentämällä pyritään pitämään piirrettävien kolmioiden määrä ruutua kohden mahdollisimman pienenä. 3D-peleissä tämänhetkinen kolmioiden kokonaismäärä kerralla ruudussa liikkuu noin sadoista tuhansista aina hieman yli miljoonaan. Nykyisten pelimallien kolmiomäärät liikkuvatkin mallista riippuen tavallisesti noin 1–15 000 kolmion välimaastossa.

Mallin pinta on hyvä rakentaa neliöistä. Mallinnusohjelmien työkalut toimivat paremmin mallin kanssa, koska syntyy vähemmän ”napoja” eli kulmapisteitä, joissa yhdistyy useampi kuin neljä reunaa. Esimerkkinä on paljon käytetty silmukkavalinta, jolla pystytään nopeasti määrittämään mallin pinnasta suoraa linjaa mukaileva reuna-alue. Mikäli matkan varrella on napoja, silmukkatyökalu ei toimi toivotulla tavalla vaan todennäköisesti risteää ja katkeaa navan kohdalla.

5.2 Kartoitukset

Erilaiset kartoitukset ovat pääasiallinen keino mallinnusvaiheessa tuoda pelimalleihin yksityiskohtaisuutta ja visuaalista ilmettä. Ne perustuvat eri tavoin luotuihin bittikarttoihin, joiden väriarvoihin on tallennettu aina kyseisen kartoituksen mukaista informaatiota sekä niitä varten ohjelmoituihin varjostimiin, jotka osaavat hyödyntää kyseistä karttaa. Karttojen erityispiirteenä tavanomaisiin kuvatiedostoihin verrattuna on niiden resoluutio. Kartoitusten dimensioiden tulisi olla kahden potensseina, koska näytönohjain on optimoitu toimittamaan matriisi- ja vektorioperaatioita. Mikäli bittikartta ei ole kahden potenssi, nykyaikaisessa näytönohjaimessa varataan sille lisää tilaa aina seuraavan neliölliseen pikselimäärään asti ja syntyy tyhjää täytettä. Yksi tapa kiertää tämä rajoite ja nostaa jopa tietyissä tilanteissa laskentanopeutta on käyttää useammalle mallille tarkoitettua atlasta, joka on edelleen toisen asteen potenssissa dimensionaaleiltaan mutta atlaksen sisällä eri mallit voivat varata tilaa vapaasti. Useamman mallin tekstuurit sisältävä atlas syntyy, kun mallien UVW-kartat kutistetaan neliön sisällä eri kohtiin, jolloin voidaan käyttää kaikilla samaa tekstuurikarttaa koska kartan pikselit on jaettu mallien kesken. [3, s. 391.]

Bittikarttojen pikseleitä kutsutaan tekseleiksi, koska ne eivät esiinny lopullisessa kuvassa pikseleinä. Ennen ruudulle piirtoa niitä skaalataan pinnan sijainnin ja UVW-kartoituksen mukaan.

Kartoitusten resoluutio vaihtelee tavanomaisesti suhteessa niitä hyödyntäviin malleihin ja niiden toistettavuuteen. Käytännössä nykypäivänä tekstuurien koot vaihtelevat peleissä 1 x 1, 4096 x 4096 välillä . Hyvänä yleissääntönä voitaisiin pitää riittävän resoluution määrittämiseen sitä, että virtuaalimaailmassa objektia mahdollisimman läheltä tarkasteltuna yksi bittikartan pikseli vastaisi ruudulla yhtä pikseliä. Näytönohjainten alati

kasvanut muisti on mahdollistanut tekstuurikokojen nopean kasvun, ja esimerkiksi tietokonepeleissä tekstuurien pitkään kestänyt painotus 512 x 512 -resoluutiosta onkin noussut välille 1024 x 1024 ja 4096 x 4096. [21.]

UVW-kartta

UVW-kartoitus toimii 3D-mallin geometrian käännöksenä kaksiulotteiseksi kuvaksi. Helppo tapa hahmottaa tämä on ajatella origamia. Origami on kolmiulotteinen, mutta origamin suoristaessa kolmas ulottuvuus katoaa ja jättää paperiin taitoskohdat.

Kirjaimet UVW edustavat siirtymistä XYZ-koordinaatistosta toiseen. UVW-kartta on neliönmuotoinen, ja sen koordinaattiarvot alkavat U- ja V-akselien 0-kohdasta (neliön vasen alakulma) ja jatkuvat oikeaan yläkulmaan, joka edustaa arvoa 1.

UVW-kartta ei ole suinkaan rajattu tälle neliön muotoiselle alueelle, vaan tietyissä tilanteissa jotkin kartan osat saattavat sijaita sen ulkopuolella, vaikka näidenkin koordinaatit katkaistaan tavallisesti 0:n ja 1:n väliin. UVW-karttaa luotaessa on hyvä pyrkiä joitakin poikkeuksia lukuun ottamatta kartan tilan jakamiseen siten, että mallin eri pintojen mittasuhteet säilyisivät mahdollisimman muuttumattomina, jolloin lopullinen malli ei näyttäisi paikoin sumealta ja paikoin terävältä. Kuten mainittiin, on kuitenkin jotain poikkeuksia, joista varmaan paras esimerkki on hahmojen päät, erityisesti kasvot. Koska ihmissilmä kiinnittää näihin alueisiin erityistä huomiota, niille saatetaan varata mallin mittasuhteet rikkoen isompi alue kartasta.

Mallina UVW-kartoittaessa kolmiulotteisen avaruuden akselit kääntyvät seuraavasti:

U = X

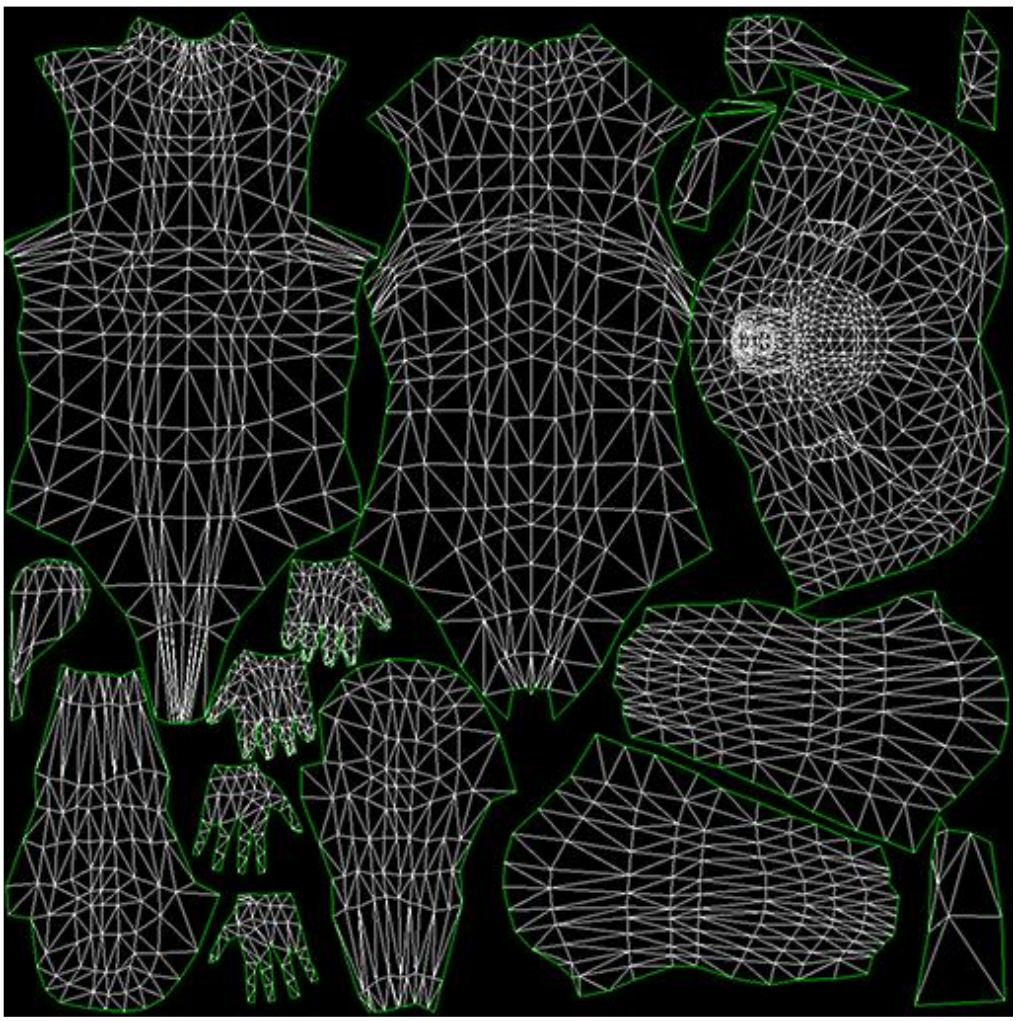
V = Y

W = Z.

W-akseli edustaa UV-tasoon nähden kohtisuoraa akselia, ja se mahdollistaa UV- kartan kääntelyn suhteessa sen geometriaan.

Kuvassa 13 on peliprojektin muumion UVW-kartta. Kartassa on pyritty piilottamaan saumat katvealueille samalla yrittäen täyttää UVW-neliö mahdollisimman tehokkaasti. UVW-karttaa olisi voinut vielä parantaa yhdistämällä esimerkiksi kädet ja reidet torsoon,

jolloin olisi saatu piilotettua vielä muutama näkyvä sauma, hieman huonomman UVW-karttatilan käytön kustannuksella.



Kuva 12. Muumion UVW-kartta.

UVW-kartan pääasiallinen tarkoitus on toimia pohjana muille, mallin grafiikkaa sääteleville kartoille, kuten tekstuuri, eli diffuusiokartalle, Specular-, AO-, normaali- ja Displacement-kartoille. Se kertoo, miten edellä mainittujen pikselikarttojen pikselit kääntyvät 3D-mallin pinnan tekseleiksi.

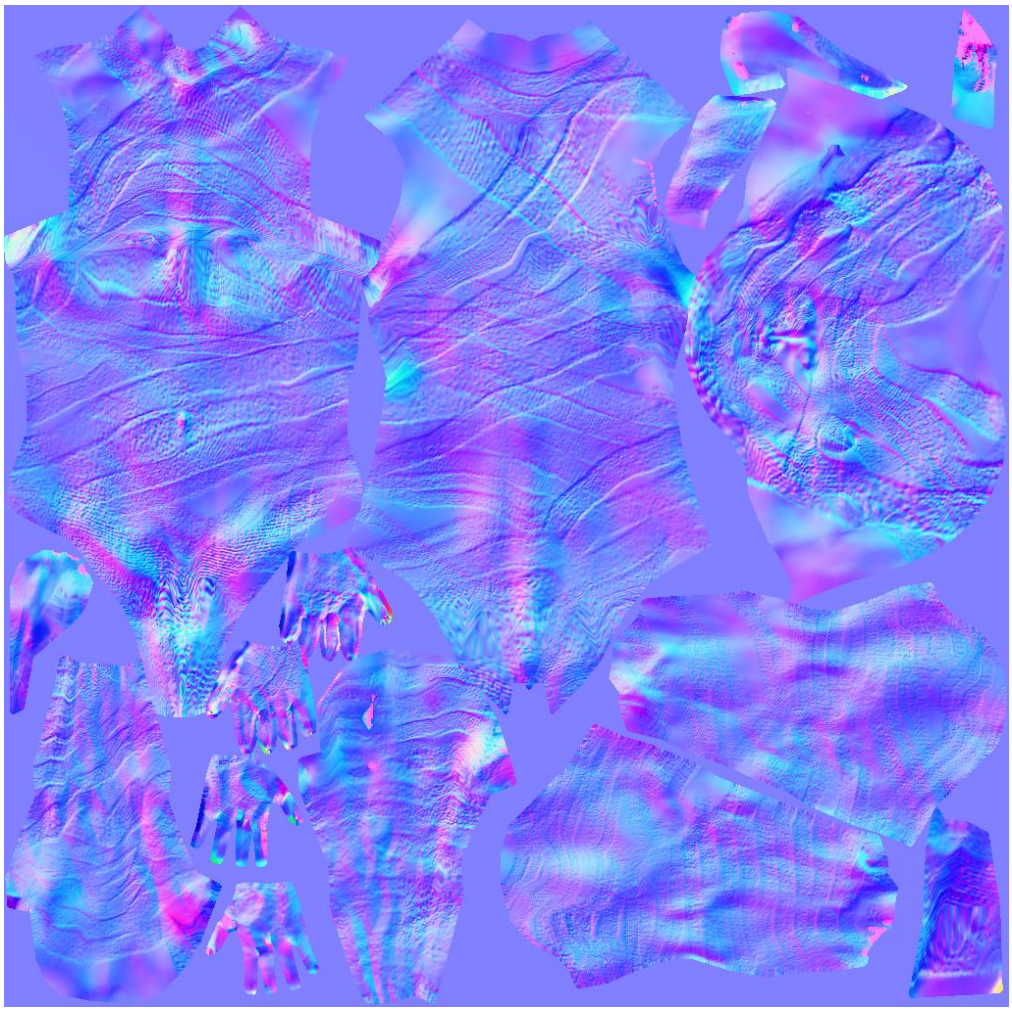
Diffuusio- eli tekstuurikartta

Tekstuurikartta rakentuu UVW-kartan pohjalle siten, että se määrittää UVW-kartan pikseleille väriarvot. Ne toimivat lopullisessa kuvassa mallin näkyvän pinnan tekseleiden pohjaväriarvoina ja tavanomaisesti myös korvaavat verteksien väreistä interpoloidut pinnanvärit. Esimerkki tekstuurikartasta on kuvassa 14, jossa on muumion tekstuurikartta.

Kuva 13. Muumion tekstuurikartta.

Normaalikartta

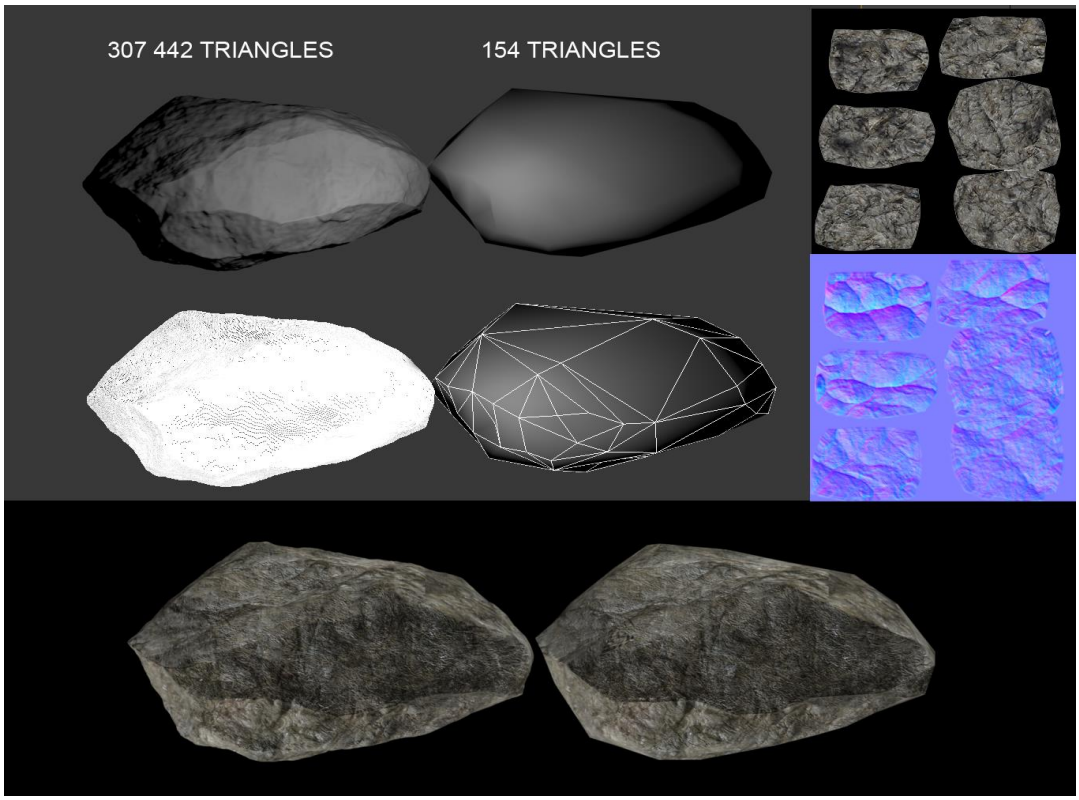
Jokaista verteksiä kohden on vähintään yksi normaalivektori määrittämässä valon heijastumista läheisiin kolmioihin. Polygonien ja normaalien määrän välillä vallitsee siis lähtökohtaisesti suhde. Normaalikartta rikkoo tämän suhteen korvaamalla peliin optimoidun mallin normaalit korkeapolygonisen mallin normaaleilla ja varastoimalla ne normaalikartan pikselien väriarvoihin. Esimerkki normaalikartasta on kuvasta 15, jossa on muumion normaalikartoitus. Pinta saadaan näin taittamaan valoa korkeapolygonimallin mukaisesti säilyttäen silti alhainen kolmiomäärä. 3D-mallille lasketaan normaalikarttojen lukuun kykenevässä varjostimessa valaistus suhteessa jokaisen kulmapisteen normaalin sijaan jokaisen tekselin normaaliin. [4, s. 130.]



Kuva 15. Muumion normaalikartoitus.

Normaalikarttoja on tangentti- ja objektiavaruudellisia. Objektiavaruuden normaalikartan tunnistaa sateenkaarimaisesta värityksestä, mutta sitä ei juurikaan käytetä sen joustamattomuuden takia. Se toimii, mikäli mallia käännellään. Tavallisesti kuitenkin deformaatiot rikkovat sen. Objektiavaruuden normaalikartta toimiikin parhaiten staattisissa objekteissa. Se kuluttaa hieman vähemmän laskentatehoa kuin tangenttiavaruuden vastaava. Tangenttiavaruuden normaalikartassa punainen värikanava tavallisesti varastoi normaalin X-akselin arvon, vihreä Y-akselin ja sininen Z-akselin. [4, s. 130; 17, s. 248–258.]

Kuvassa 16 on esitetty normaalikartoitusprosessi. Vasemmalla puolella on kuva korkeapolygonisesta mallista ja oikealla matalapolygonisesta. Kuten kuvasta käy ilmi, on korkeapolygonisen kivimallin pinnalla hyvin tiheä verteksiverkko jolloin pinta kykenee tarkempaan interaktioon valonlähteiden kanssa. Matalapolygonisen kivimallin pinnassa on vain muutama, välttämättömiä muotoja tuova verteksi, jolloin valoinformaatio jää hyvin vähäiseksi. Siksipä korkeapolygonisen mallin verteksidata ”tallennetaan” normaalikarttaan ja matalapolygoninen malli piirretään normaalikartoitusta tukevalla varjostimella pelimoottorissa. Näin saavutetaan lähes yhtä laadukkaan lopputulos vähäisemmällä laskennalla.



Kuva 16 Normaalikartoitusprosessi.

Valaistuskartat

AO-kartta eli ambientti okkluusio luo valmiiksi laskettuja pehmeitä varjoja mallin pintaan pinnan syvyyksien mukaan siten, että varjoilla ei ole selvää valonlähdettä. AO-kartta voidaan yhdistää esimerkiksi diffuusiokartan kanssa, jolloin väritys saa automaattisesti varjostuksen tekstuurien väriin vaikuttamatta sillä kartta käyttää vain mustan ja valkoisen väliarvoja. Tavalliset valo- ja varjokartat eroavat AO-kartasta siten, että niihin osuvalla valolla on selkeä suunta. Liikkuvissa malleissa voidaan käyttää vain AO-karttaa (kuva 17), koska valon suuntaa ei ole määritelty ja voidaan olettaa kartan sijainneissa sijaitsevan jonkinasteista tummenemista joka tapauksessa.

Kuva 17. AO-kartta muumiosta.

.

Specular-kartta eli pinnan kiiltoisuuden kartoittaminen määrittää, kuinka paljon valoa heijastuu takaisin tietystä kohdasta mallia. Pinnan kiiltoisuus määräytyy bittikartan pikselin musta – 0 % kiiltoa sekä valkoinen – 100 % kiiltoa välisillä arvoilla.

Vaaleat kiiltoisuuskartat esittävät pintoja, joissa esiintyy kiiltoa runsaasti, kuten metallit ja muovit, kun taas tummemmat kartat pintoja, joissa on vain vähän valon takaisin heijastumista, kuten sementti, puu ja kankaat. Myös kiiltoisuuskartan väriarvoja hyödynnetään määrittämään heijastuvan valon väri. Kuvassa 18 metallille on annettu hieman sinertävä kiiltoisuuskartta.

Kuva 18. Koristellun kirveen tekstuuri- ja kiiltoisuuskartta.

Siirtymäkartoitus eli displacement-mapping

Siirtymäkartta ei varsinaisesti kevennä laskentaa. Yleistyvien tesselaatiovarjostimien yhteydessä käytettynä sillä voidaan lisätä yksityiskohtien laatua. Siirtymäkartoitus siirtää mustavalkoisen kartan perusteella polygonien pisteitä korkeussuunnassa tangenttiavaruudessa eli syvyyssuunnassa ihmisen ajattelulla. Se luo epätasaisuutta ja tuo tiettyjä kohtia ulos mallista. Se ei lisää pisteitä vaan kykenee siirtämään vain jo olemassa olevia. Tesselaatio taas lisää polygonitiheyttä automaattisesti malliin pintaan sen mukaan, kuinka suuren tilan se peittää kuvaruudusta eli kuinka läheltä sitä tarkastellaan. Näin

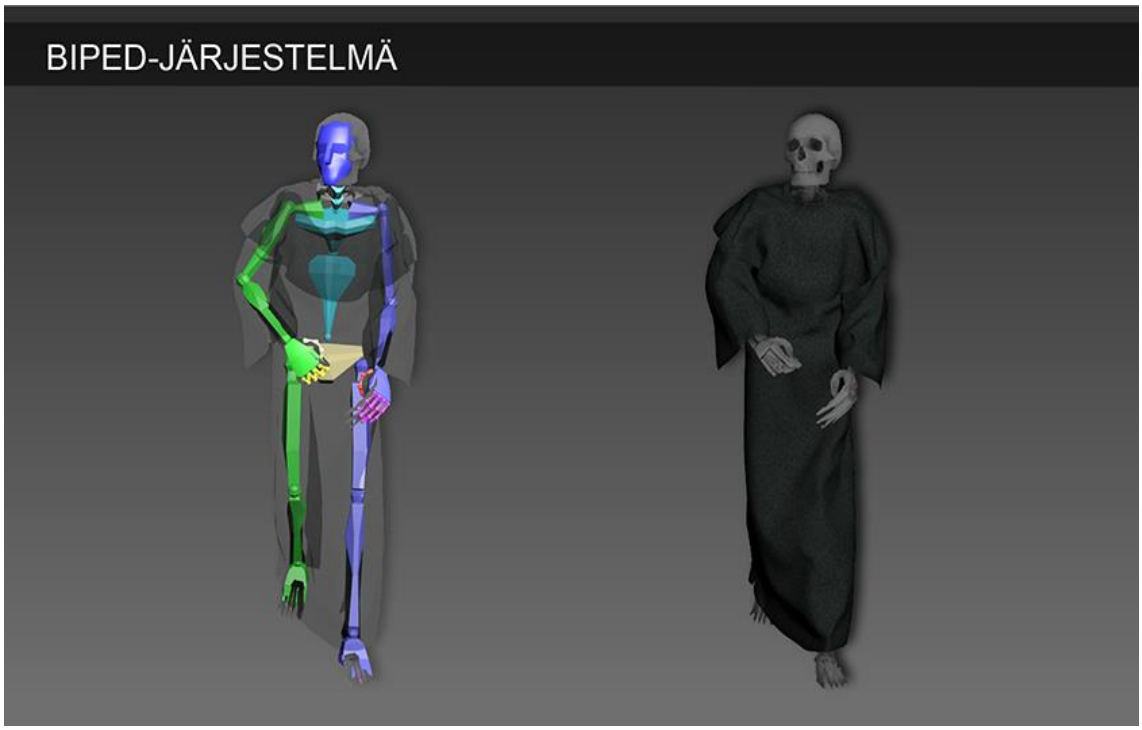
saadaan lisättyä yksityiskohtaisuutta laskennallistehokkaasti ja hyödynnettyä siirtymäkarttaa paremmin.

5.3 Riggaus

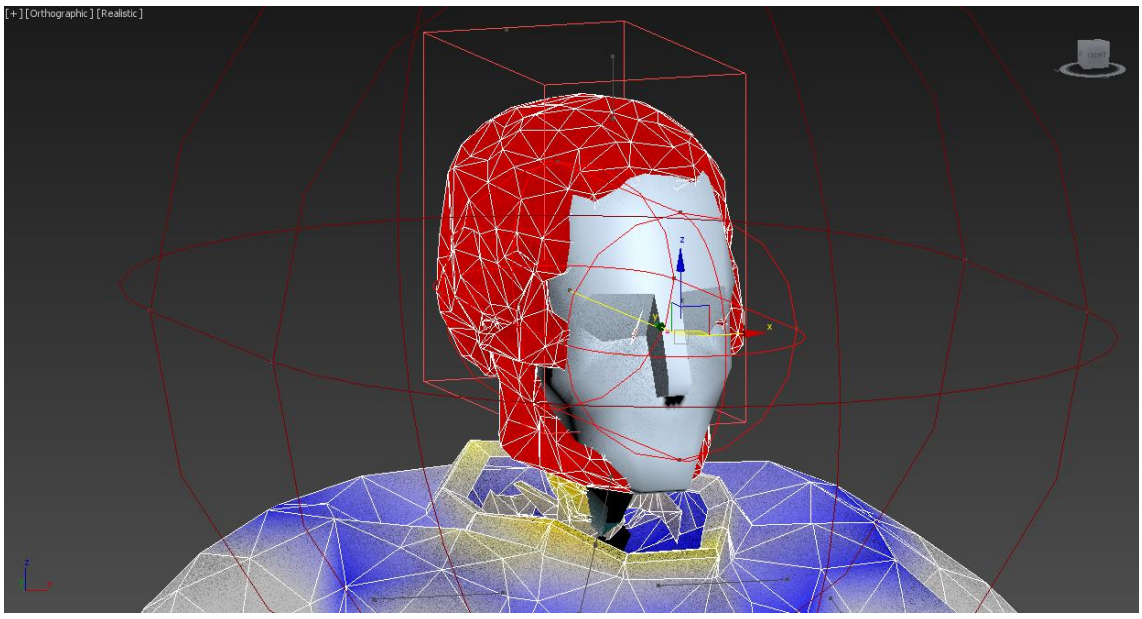
Mikäli malli on tarkoitus animoida ja sille halutaan mallin sisäisiä animaatioita, kuten liikkuvia jäseniä, on helpompaa luoda animointijärjestelmä, joka määrittää eri raajojen mahdolliset liikerajat ja jonka avulla mallin asentoa voi helposti muuttaa, kuin siirtää jokaista verteksiä tai kolmiopintaa manuaalisesti joka ruudussa haluttuun sijaintiin. Myös mallin geometria säilyttää paremmin suhteensa. Tällaisen järjestelmän luontia kutsutaan riggaamiseksi, ja sen lopputuloksena syntyvää järjestelmää ”luu”- tai Biped-järjestelmäksi.

Biped-järjestelmä

Biped-järjestelmä on 3ds Maxin erityisominaisuus, joka on suunnattu erityisesti humanoidi mallien riggausta varten. 3ds Max tuottaa valmiin Biped-nuken (kuva 19), jolle määritetään eri nivelten, sormien ja varpaiden määrä. Tämän jälkeen se pyritään asettelemaan ja skaalamaan mallin mukaisesti sen sisään. Paras asento luoda humanoidimalli on sellainen, missä raajat ovat mahdollisimman etäällä toisistaan. Kolmionnin saa taitekohdista parhaaksi mahdolliseksi, ja myöhemmin verteksipainotuksia aseteltaessa tapahtuma on mahdollisimman helppo. Kun biped-järjestelmä on aseteltu mallin asentoon, lisätään 3ds Maxin muunnospinoon skin-muunnos, joka pyrkii heti automaattisesti painottamaan eri jäsenten verteksit seuraamaan vastaavaa bipedin jäsentä. Valitettavasti tämä harvoin onnistuu täydellisesti, ja käyttäjän on korjailtava painotuksia (kuva 20). Painotusten tarkistaminen käy helpoiten koittamalla liikuttaa bipedin jäseniä mahdollisimman äärimmäisiin asentoihin ja seuraamalla, miten hyvin mallin pinta säilyttää alkuperäiset muotonsa. Kun geometria seuraa toivotulla tavalla järjestelmää, voidaan sanoa riggauksen olevan valmis ja voidaan siirtyä mallin animoimiseen.



Kuva 19. Pelin luurankovihollisten biped-järjestelmä.



Kuva 20. Biped-järjestelmän verteksikohtaisia painotuksia.

Biped-järjestelmän etu luujärjestelmään verrattuna ovat tarjoamat animaationluontityökalut ja eri animaatioiden siirrettävyys bipediltä toiselle. Biped tarjoaa lähes automatisoidut keinot luoda kävely-, juoksu- ja/tai hyppyanimaatioita vain valitsemalla bipedille ”askel” tila ja asettamalla parametrit. Nämä animaatiot tarjoavat hyvän pohjan mille tahansa humanoidille, ja niitä pystyy muuntamaan mieleisekseen esimerkiksi asettamalla uusia

tasoja aikajanaan ja manuaalisesti animoimalla lisäliikettä raajoihin, jolloin lopputuloksena eri tasojen animaatiot sulautuvat toisiinsa.

Luujärjestelmä

Luujärjestelmä perustuu mallille yksittäisistä komponenteista eli luista rakennettavaan hierarkkiseen rakennelmaan ja on huomattavasti joustavampi biped-järjestelmään verrattuna. Se mahdollistaa muidenkin kuin humanoidiluujärjestelmien luomisen. Joskus luu-järjestelmiä voidaan esimerkiksi käyttää biped-järjestelmien ohella, kun halutaan animoida tarkempia yksityiskohtia, kuten hiusten tai vaatteiden liikettä. Luut tukevat kahta erilaista hierarkkista käyttäytymistä, jotka tunnetaan suorana ja käänteisenä kinematiikkana. Luujärjestelmissä on tavallisesti yksi isäntäluu, josta haarautuvat muut luut, ja edellisen luu on aina isäkappale seuraavalle. Suorassa kinematiikassa lapsikappale seuraa aina isäkappaleensa liikettä, kun taas käänteisessä lapsi määrittää liikkeen. Jokaisen luun objektiavaruuden origo sijaitsee luun tyvessä, ja ne liikkuvat tällä lailla syntyvien nivelten mukaisesti. Kun liikuttaa lapsikappaletta, se heijastuu luun kääntymisenä isäntäkappaleeseen. Luujärjestelmä luodaan samantyyppisesti kuin biped-järjestelmän, tällä kertaa vain luu kerrallaan.

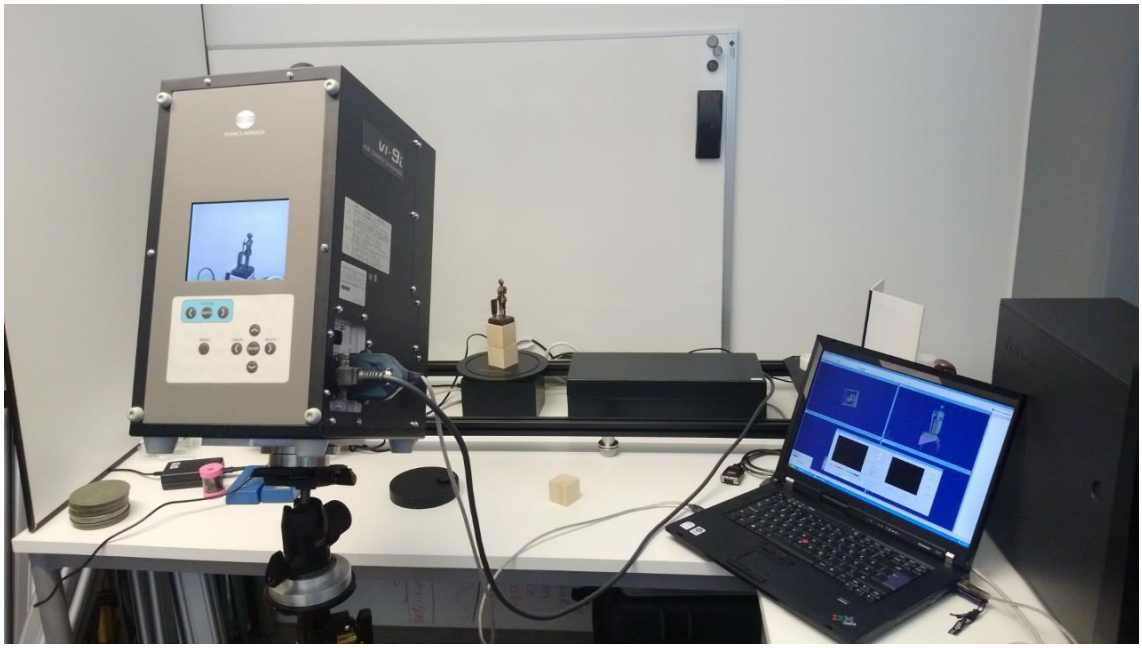
Kuvassa 21 on esitelty insinöörityön peliprojektia varten tehty hämähäkkimäinen luujärjestelmä. Järjestelmän ja sen animaatioiden luontia varten tutkittiin hämähäkkien liikeratoja.

Kuva 21. Luurankohämähäkin luujärjestelmä.

6 Lasermittaukset peliprojektia varten

6.1 Esineiden mittaus Konica Minolta -3D-digitointilaitteella

Joukko pienehköjä esineitä digitoitiin peliprojektia varten Konica Minoltan V9i-3D -digitointilaitteella (kuva 22). Toimintaperiaatteeltaan V9i on kolmioiva strukturoitua laservaloa hyödyntävä 3D-mittalaite, joka kykenee korkeisiin mittaustarkkuuksiin tasavärisissä vapaamuotoisissa pinnoissa. Kappaleet keilattiin pyörittäen niitä skannerin pyörityspöydällä. Kustakin kohteesta otettiin useita keilauksia 15–30 asteen välein kohteen geometriasta riippuen. Skannaukset yhdistettiin Konica Minolta Polygon Editin Tools (PET) -ohjelman automaattisilla toiminnoilla. Pyörityspöydän käyttö helpotti huomattavasti yksittäisten skannausten yhdistämistä, koska yhdistämisessä oli lähtötietoina skannausten välisen kulman muutos, pyöritysakelin sijainti ja kulma. Lisäksi kohteista otettiin tekstuurikuvat digitointilaitteen sisäisellä kameralla. [13.]



Kuva 22. Konica Minoltan V9i -3D-digitointilaitteella saa nopeasti 3D-malleja pienehköistä esineistä ja kappaleista. © FGI ja Aalto. Hannu Hyyppä.

Kolmioiduista ja yhdistetyistä skannauksista rakennettiin umpinaiset mallit Geomagic Studio -ohjelmistolla (http://www.geomagic.com/en/), joka on suunniteltu esineiden reverse engineering -työhön. Skannausten aukot täytettiin niiden reunojen tangettitietojen perusteella, ja kolmioverkon ongelmakohtia (kuten risteävät kolmiopinnat) korjattiin kulmien pehmennystyökalujen avulla.

Valmiit, tässä vaiheessa vielä erittäin tiheät kolmioverkkopintamallit teksturoitiin digitointilaitteen kameran kuvilla. Tilanteissa, joissa esineet olivat pinnaltaan erittäin kiiltäviä tai heijastavia, tätä teksturointia ei tehty, koska kuvissa näkyi paljon ympäristön heijastuksia ja kappaleen pinnan kiillot vaihtelivat kuvissa kappaleen kiertokulman mukaan. Teksturoinnin jälkeen kohteista laskettiin myös matalamman tarkkuuden kolmioverkkopintamallit. Tavoitteena oli ennen kaikkea kolmioiden määrän lasku riittävän suorituskyvyn säilyttämiseksi pelimoottoria käytettäessä, ja samalla mallien tiedostokoot luonnollisesti laskivat. Mallit tallennettiin .obj-muodossa. [14.] Kuvassa 23 on esitetty Konica Minoltan V9i (http://www.3dscanco.com/products/3d-scanners/3d-laser-scanners/konica-minolta) -mittauslaitteella matkamuistoteroittimesta syntynyt oikea haarniska hylätyn linnan käytäviä koristamaan.

Kuva 23. Koristehaarniskat linnan käytävällä.

6.2 Rakennetun ympäristön mittaus Leican maalaserkeilaimella

Grönqvistin talon 2000-luvun alkupuolella kunnostetun julkisivun on arveltu olevan yksi Helsingin keskustan monimuotoisimmista julkisivuista. Digitointi tehtiin rakennuksen edestä kadulta, noin silmän korkeudelta (kuva 24). Aikaa digitointiin kului joitakin tunteja. Julkisivusta mitattiin useita usean miljoonan pisteen pistepilviä. Syntyneet pistepilvet yhdistettiin tähysten avulla.

Kuva 24. Gröngvistin talon Esplanadin puoleinen osa digitoitiin Leican valmistamalla maalaserkeilaimella. © FGI ja Aalto. Hannu Hyyppä.

Rakennuksen julkisivun pistepilvestä valittiin jatkokäsittelyyn rakennuksen oven sisältävä alue (kuva 25). Pistepilvi kolmioitiin, kolmioverkkopinta siistittiin ja siitä laskettiin eritarkkuuksiset versiot Geomagic Studio -ohjelmalla (http://www.geomagic.com/). Tässä tapauksessa mallia ei teksturoitu valokuvilla. [13.]

Kuva 25. Grönqvistin talon julkisivumalli © Aalto, Virtanen, 2012.

Kolmioverkkopintamallien käsittelyä pelimoottorikäyttöön paremmin sopivaksi jatkettiin Blender-ohjelmassa. Matalan kolmiomäärän mallille rakennettiin tekstuurikartta (UV-teksturointi), ja tämän jälkeen sille luotiin normaalitekstuuri korkean kolmiomäärän mallista. Tällä tekniikalla matalan kolmiomäärän mallilla saavutettiin riittävä yksityiskohtataso renderöitäessä, kasvattamatta mallin kolmiomääriä liikaa.

Kuva 26. Gröngvistin talon sisäänkäynti pelissä.

Mallin jälkikäsittely jatkui 3ds Maxissa, jossa kolmioverkkoa siistittiin turhiasta polygoneista ja malliin jääneet reiät tilkittiin. Ovi yhdistettiin 3ds Maxissa mallinnetun mausoleumimallin sisäänkäynniksi (kuva 26).

Patsaan mittaus

Turun yliopiston kampuksella sijaitseva suurehko patsas mitattiin kahdella skannauksella Leican HDS6100-maalaserkeilaimella. Laitteisto oli sama, jota hyödynnettiin aiemmissa rakennusmittauksissa. Pistepilvistä tehtiin kolmioverkkopintamalli Geomagic Studio-ohjelmalla (http://www.geomagic.com/), vastaavalla tavalla kuin Grönqvistin talon julkisivun tapauksessa. Tämän jälkeen sekä korkean kolmiomäärän malli että matalan kolmiomäärän malli tallennettiin .stl–muodossa. Patsaan kolmioverkkopintamallin käsittelyä jatkettiin Blender-ohjelmassa, ja siitä toteutettiin vastaava normaalitekstuuroitu malli. Blender-ohjelmasta se siirrettiin vielä 3ds Max -ohjelmaan jälkikäsittelyä varten.

Kuvassa 27 kolme kansallista suurmiestä, ”Runeberg, Lönnrot, Snellman” on Harry Kivijärven suunnittelema mustagraniitista valmistettu muistomerkki, joka sijaitsee Turun yliopiston päärakennuksen edustalla Turun Yliopistonmäellä. Kuvassa 28 on esitetty sama patsas peliympäristöön vietynä.

Kuva 27. Turun yliopiston päärakennuksen edustalla sijaitseva muistomerkki © Aalto, Virtanen, 2012 & Hyyppä ja Ahlavuo 2013.

Kuva 28. Muistomerkki pelimaailmassa.

6.3 Kuvapohjainen rakennusmittaus - käsin tehty kuvapohjainen mittaus

Olavinlinnasta tuotettiin karkea, linnan tärkeimmät muurirakenteet sisältävä ulkopintamalli kuvapohjaisia mittausmenetelmiä hyödyntäen. Osasta linnan muuria laskettiin kolmiulotteinen pintamalli pitkällä polttovälillä kuvattujen panoraamakuvien pohjalta. Lisäksi linnasta mitattiin fotogrammetrisesti joukko muita pisteitä viistoilmakuvista iWitness-ohjelmalla. Tavoitteena oli mitata linnan muodon kannalta olennaisimmat nurkkapisteet isoista rakenteista. Ilmakuvista mitattujen pisteiden pohjalta rakennettiin yksinkertaiset kolmiopinnat. Aineistot yhdistettiin tornien huippujen ja muurien yhteisten pisteiden avulla. Vastaava yksinkertainen pintamalli rakennettiin myös panoraamakuvista mitatusta muurista sovittaen pintoja kolmioverkkopinnan päälle. Tuloksena saatiin yksinkertainen pintamalli koko linnasta (kuva 29). [13.]

Kuva 29. Olavinlinna pelissä.

Syntynyt pintamalli on teknisestä näkökulmasta erittäin kevyt ja sisältää vain hyvin alhaisen pistemäärän. Se kuitenkin sisältää koko linnan suurimpien rakenteiden peruspiirteet ja mahdollistaa linnoituksen kokonaisuuden hahmottamisen.

Kartoitusten luonti

Pelissä esiintyville malleille luotiin yleensä Photoshop -kuvankäsittelyohjelmaa ja 3ds Max -mallinnusohjelmaa käyttäen tekstuuri- ja normaalikartat. Joissakin tapauksissa luotiin myös heijastuskartat. Maaston puiden luonnissa käytettiin pakattua kartoitusta, jossa yhden bittikartan punainen sisälsi läpikuultavuuskartan, vihreä normaalikartan vihreän kanavan, sininen loistekartan ja alphakanava normaalikartan punaisen kanavan.

6.4 Automaattinen kuvapohjainen mittaus

Helsingin Sinebrychoffin puistossa sijaitsevasta tornista tehtiin osittainen ulkopinnan malli automaattista kuvapohjaista mittausohjelmaa hyödyntäen. Mallin tekemiseen käytettiin Autodesk 123D Catch -ohjelmaa (http://www.123dapp.com/catch), joka on erittäin

voimakkaasti yksinkertaistettu ja helppokäyttöinen automaattinen kuvapohjainen mittausohjelma. Sen käyttöä kuitenkin rajoittavat täysin automaattisen toiminnan huono säädettävyys ja lisenssiehdot. [16.]

Tornin itäpuolen julkisivusta otettiin noin 70 kuvan joukko digitaalisella järjestelmäkameralla (Nikon D5000). Kuvauksessa käytettiin laajakulmaista objektiivia, jolloin koko rakennus mahtui kuviin. Kuvattaessa kamera pidettiin käsisäädöillä, ja objektiivin tarkennus ja polttoväli lukittiin. Kuvajoukosta laskettiin teksturoitu kolmioverkkopintamalli Autodesk 123D Catch -ohjelmalla. Tulos siirrettiin Blender-ohjelmaan. Mallia siistittiin poistamalla siitä joitain kasvillisuuden aiheuttamia rakennukseen kuulumattomia kohteita ja poistamalla maatason osa mallista. Blenderistä malli vietiin 3ds Maxiin, jossa sille suoritettiin UVW-kartoitus ja lopulliset korjaukset ennen pelimoottoriin vientiä. [13.]

6.6 Ihmisen skannaus maalaserkeilaimella

Uutta Faro Focus -3D-maalaserkeilainta (http://www.faro.com/en-us/products/3d-surveying/faro-focus3d/overview) kokeiltiin ihmisen mittauksessa. Mittausasetelmassa hahmon etupuoli skannattiin suoraan ja selkäpuoli suurikokoisen peilin kautta. Hahmon ympärille asetettiin pallotähykset syntyvien mittausten yhdistämistä helpottamaan. Peiliä hyödyntämällä koko hahmo pystyttiin mittaamaan riittävän kattavasti yhdellä mittauksella, jolloin mittausaika pysyi noin 20–30 sekunnin mittaisena.

Syntynyt pistepilvi kolmioitiin, ja hahmon puolikkaat yhdistettiin Geomagic Studio -ohjelmassa (http://www.geomagic.com/). Yhdistämisessä käytettiin hahmon selkäpuolen skannauksessa hyvin erottuvia peilaustasoja, jolloin puolikkaat saatiin jo melko tarkasti yhteen. Tämän jälkeen tulos viimeisteltiin käsin pallotähyksiä ja hahmon piirteitä hyödyntäen. Mitattuun hahmoon jäi jonkin verran aukkoja mittausjärjestelyn vuoksi (toisen peilin käyttö olisi varmaankin parantanut tuloksia). Nämä syntyneet aukot täytettiin reunojen tangentiaalisuus huomioiden. Muuten mittauksen tarkkuus havaittiin melko hyväksi ja kohinan taso riittävän matalaksi esimerkiksi avatar-mallina käyttöä ajatellen. Sen jälkeen malli siirrettiin Blender-ohjelmaan ja siitä luotiin .obj-tiedosto. [14.]

Ihmismallia ei kuitenkaan hyödynnetty pelissä, koska sille ei onnistuttu löytämään oikean aikakauden vaatteita. Huomattiin kuitenkin, että manuaalisesti parantamalla mallin kolmioverkkoa olisi mahdollista saada toimivat animaatiot hahmolle ja sen luurankojärjestelmälle.

6.7 Lasermallien jälkikäsittely

Korkeapolygonisten mallien .obj-tiedostot siirrettiin 3ds Max 2014 -sovellukseen, jossa niille suoritettiin multires-toiminnolla polygonimäärän laskeminen mallista riippuen 1000–15 000 kolmioon pelimaailmaan sopivan mallin luomiseksi. Tämän jälkeen mallin pintaa vielä optimoitiin poistamalla turhia polygoneja manuaalisesti yhdistämällä verteksejä toisiinsa ja näin tuhoamalla pinnan muotoon vaikuttamattomia kolmioita. Seuraavaksi pinnan kohoumia voidaan vielä hienosäätää.

Kun mallin matalapolygoninen versio oli valmis, korjattiin vielä UVW-kartta paremmin tilansa käyttäväksi ja mahdollisimman ”laatoitusta” tukevaksi. Joissain pienten objektien tapauksissa käytettiin suoraan autogeneroitua UVW-karttaa, jotta pystyttiin hyödyntämään mittauskohteesta saatua alkuperäistä värikarttaa. Tämän jälkeen mallin sijainti origossa varmistettiin, samoin se, että mallin objektiavaruuden origo sijaitsi mallin keskipisteessä tai tietyissä tilanteissa mallin pohjan keskellä. Sen jälkeen luotiin diffuusio- eli tekstuurikartta mallille Adoben Photoshop-ohjelmaa käyttäen.

UVW-kartan korjauksen jälkeen luotiin 3ds Maxissa normaalikartta tuomalla alkuperäinen korkeapolygoninen malli samaan tiedostoon ja asettelemalla se samaan sijaintiin koordinaatistossa matalapolygonisen mallin kanssa, minkä jälkeen korkeapolygonisen mallin verteksien normaalit projisoitiin mallien ympärille muodostetun ”häkin” pinnasta lähtevien säteiden avulla matalapolygonisen mallin pinnalle ja tulostettiin tästä muodostuva normaalikartta. Toinen tapa, jota käytettiin, oli tuoda kummatkin mallit .obj-tiedostomuotoina xNormals-nimiseen normaalikartan generointiohjelmaan, jossa karttojen luotiin lähes automaattisesti, vain yksinkertaisesti parametreja säätelemällä. Lopuksi mallin sisältävä Osuma-laatikko (Bounding box) istutettiin uudelleen mallin ympärille oikeaan kulmaan.

7 Peliprojekti

7.1 Suunnitelma

Insinöörityön osana tehdyn peliprojektin lähtökohtana oli ideoida ja kehittää peli, jossa laserkeilatut mallit pääsisivät hyvin esiin. Parhaiten keilattu aineisto on edustettuna grafiikaltaan realismia tavoittelevassa pelissä, jossa kameran perspektiivi on lähellä kuvan sisältöä, jotta yksityiskohdat pääsevät paremmin esille. Tämä tietenkin johtaa graafisesti hieman raskaampaan lopputulokseen, koska tällöin esimerkiksi tekstuurien resoluutioiden on pystyttävä tarjoamaan tarkkuutta hyvin läheltä tapahtuvassa tarkastelussa. Suuret tekstuurit vaativat paljon videomuistia, ja tietokoneiden näytönohjaimissa sitä on tarjolla runsaimmin, joten oli luontevaa valita PC-pelin alustaksi. Koska PC tukee parhaiten myös eri tekniikoita, se mahdollisti parhaiten peliprojektin kulun tarkastelua, omien taitojen kehitystä ja pelinkehityksen tutkimista.

Projektin pelisuunnittelu alkoi sen pohtimisella, minkälaisessa ympäristössä lasermitatut mallit pääsisivät parhaiten oikeuksiinsa. Lisäksi määriteltiin pelin tyyli ja lajityyppi. Ensimmäiseksi päädyttiin kolmannesta persoonasta kuvattuun rooliseikkailupeliin, koska siinä malleja voi katsella läheltä ja useasta kulmasta kameraa liikuttamalla. Surrealistinen steampunk teemana houkutteli, joten peli sijoitettaisiin maailmaan, jossa kellokoneistot olisivat suuressa osassa. Tunnelmaa korostettiin visioimalla sumuinen maailma, jossa kellokoneistot liittyisivät koko maailmankaikkeutta pyörittävään mekaniikkaan, jonka oivaltaminen tulisi olemaan oleellista pelissä. Samalla tutkittiin, kuinka oleellinen osa pelin pelattavuutta on selkeä, nähtävä kausaliteetti, joten maailma sijaitsikin eräänlaisella unitasolla, johon viljeltiin epäjohdonmukaisuuksia kuten merenranta majakoineen linnan sisällä. Pelisuunnitelma kuitenkin eli projektin edistyessä.

Alkuaineistona oli fotogrammetriaa hyväksikäyttäen luotu Olavinlinnan malli, joka johti pelin hengen muuttumiseen hieman keskiaikaisemmaksi ja synkemmäksi. Lopulta pelimaailmaksi muotoutui kauhuseikkailu roolipelivivahtein.

7.2 Juoni

Soisella alueella sijaitseva linna kätkee salaisuuksia. Linnan talonmies on hoitanut rakennusta parhaansa mukaan isäntänsä poissa ollessa ja vaimonsa toimiessa majakanvartijana linnan sisällä sijaitsevassa majakassa (kuva 30), mutta pariskuntakin näyttää kadonneen jättäen jälkeensä vain toinen toistaan epämääräisempiä viestejä. Niiden mukaan linnan syövereistä on alkanut kantautua entistä pahaenteisempiä ääniä ja todennäköisin lähde ovat isännän eriskummalliset koneet (kuva 31). Pelaaja herää tähän vieraaseen maailmaan vanhan mökin kuilumaisesta kellarista vailla mitään muistikuvia menneisyydestä ja mielessään vain kummallinen tunne siitä, että kaikki ei ole kohdallaan.

Kuva 30. Linna kätkee sisäänsä meren rannan, josta vähän matkan päässä sijaitsevalla luodolla on majakka.

Kuva 31. Totuus maailmasta alkaa paljastua pelaajalle vähitellen.

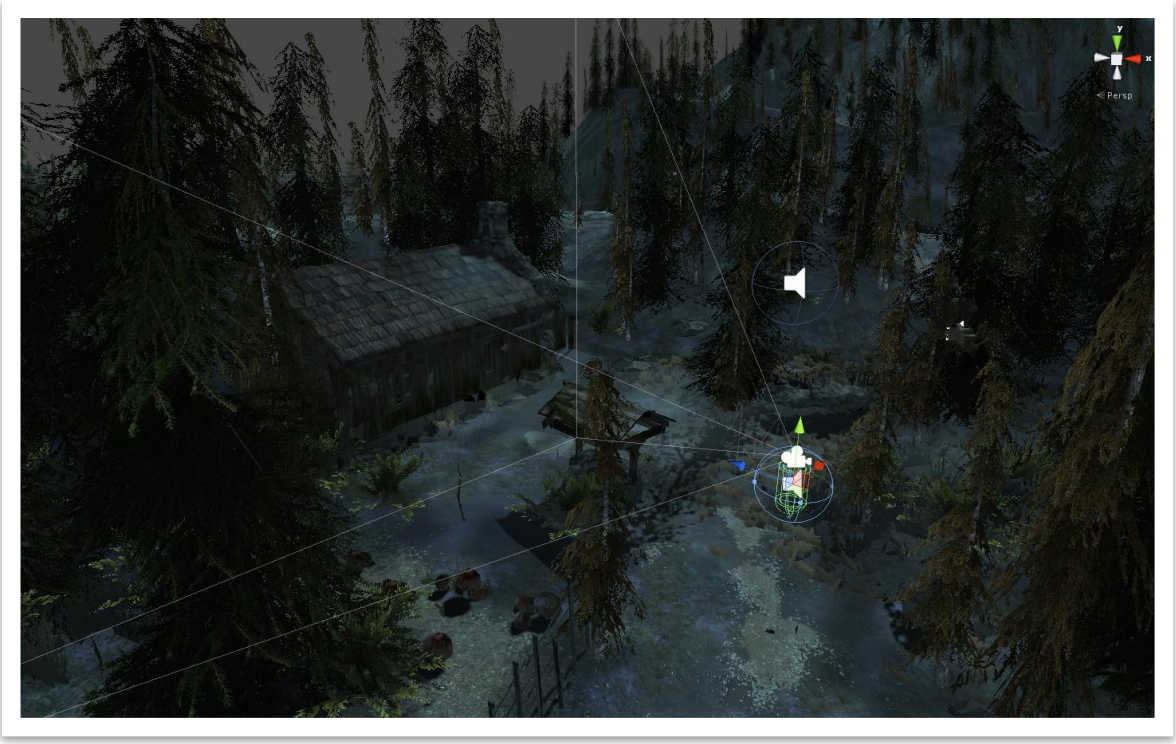
7.3 Pelimaailma

Pelimaailma sijoittuu hämärän rajamaille, unenomaiseen tilaan, jossa kausaliteetti rakoilee ja suuret kellokoneistot pitävät aikaa loputtomassa kierteessä.

Ympäristössä on pyritty tunnelmallisuuteen ja tavanomaisista peliympäristöistä poikkeavaan asetteluun. Metsät ja kasvusto ovat paikoittain hyvin tiheät. Kasvillisuus on osittain otettu Suomen luonnosta, kuten kuuset ja horsmat (kuva 32). Pelin pääalueena on usvainen suoalue, jossa suuren mäen päällä sijaitsee linna. Suoalueella on myös hieman matalamman mäen päällä juonen kannalta keskeinen hylätty hautausmaa (kuva 32) mausoleumeineen sekä parhaat päivänsä nähnyt mökki (kuva 34) ja lato. Suoalueen soista nousee valtavia hammasrattaita, ja metsissä vaeltaa luurankomunkkeja, sotureita ja mekaanisia hyönteismäisiä olentoja.

Kuva 32. Pelin metsäalueet ovat hyvin tiheäkasvuisia ja sisältävät viitteitä Suomen luontoon.

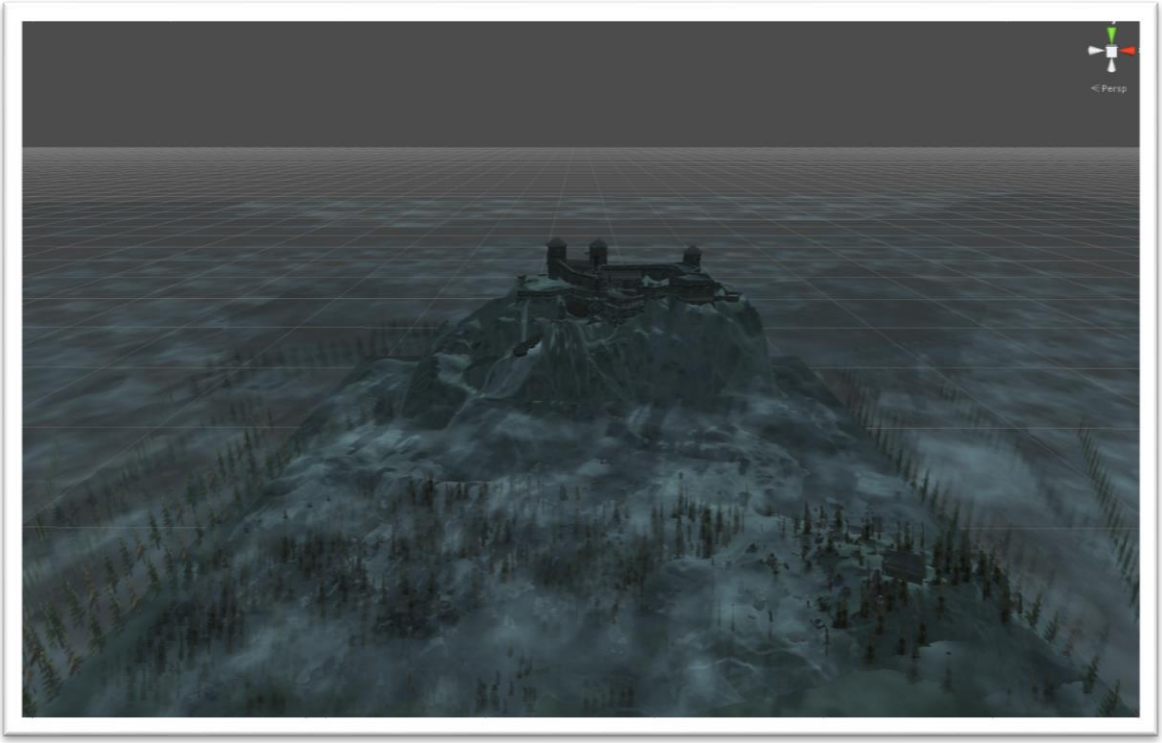
Kuva 33. Hylätty hautausmaa suolla sijaitsevalla mäellä.



Kuva 34. Pelin näkymä Unityn editorissa, jossa pelaaja-järjestelmä katsoo kohti rakennusta.

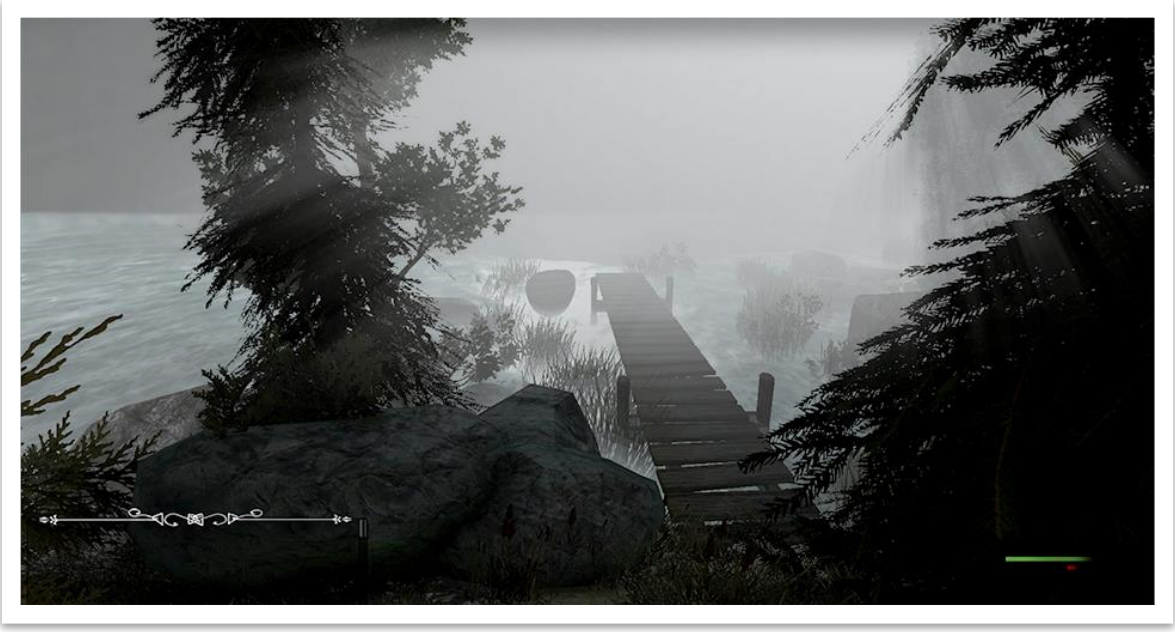
Pääalueelta pääsee sivualueille, joita ovat ränsistyneen mökin kellari, linnan maakellari (kuva 35), mausoleumi ja linna (kuva 36).

Kuva 35. Linnan maakellarissa sijaitsee useampia salakäytäviä.



Kuva 36. Suoalue linnoineen. Pelin suurin yksittäinen alue.

Linna on jakautunut kolmeen eri alueeseen. Linnan oleskelutilat ovat loputon sokkelo. Pelaajaa siirrellään huoneista samannäköisiin huoneisiin täsmälleen samassa asennossa sattumanvaraisesti. Oleskelutiloissa on myös paljon lintupääkalloisia luurankovihollisia, jotka on suhteellisen helppo päihittää yksitellen, vaikkakin massoina ne ovat erittäin vaarallisia. Oleskelutiloista on pääsy linnan sisäiselle meren rannalle (kuva 37), jossa pelaajaa odottaa venekyyti majakan rantaan. Oleskelutiloissa on myös salakäytävä linnan alla sijaitsevaan luolastoon, johon pelaaja pääsee pelin myöhäisemmässä vaiheessa.



Kuva 37. Linnan sisällä sijaitsevalle majakalle pääsee vain veneellä.

Pelialueelta toiselle pääsee kulkemaan vapaasti, mikäli pelaaja on hankkinut pääsyn kyseiselle alueelle. Esimerkiksi linnan ovi on lukossa, eikä sinne pääse ilman avainta. Pääsy rannalle, josta pääsee veneellä linnan majakalle, on osittain sattumanvarainen tapahtuma. Linnan käytävillä täytyy vaeltaa jonkin aikaa löytääkseen huoneen, josta siirtymä tapahtuu. Majakkatapahtuma on kertaluonteinen.

7.4 Ohjelmointi

Pelin ohjelmoinnissa hyödynnettiin sekä C#:a että Unityscriptiä. Näiden kahden välillä pelin lopullisessa suoritusnopeudessa ei ole eroa, kunhan käyttää tyypitettyä Unityscrip-

tiä. Tavallisesti voitaisiin sanoa, että paras tapa on käyttää yhtä kieltä, mutta koska kyseessä ei ole kaksi aivan samoin toimivaa kieltä, jotkin asiat oli mielekkäämpää toteuttaa toisella kielellä kuin toiset. Myös niiden erillinen kääntöjärjestys auttoi koodin jäsentämisessä. Koska C# kääntyy ennen Unityscriptiä, sillä pyrittiin tekemään staattiset, yleiset ja riippumattomat luokat. Komponentteihin, joita kutsutaan toistuvasti, kuten vihollisten tekoäly ja kentällä sijaitsevien objektien animaatiot, optimoitiin olemaan kuluttamatta laskentaa, kun ne sijaitsevat liian kaukana pelaajasta tai eivät sillä hetkellä tarjoa mitään lisää pelikokemukseen.

Projektissa kiinnitettiin myös huomiota objektien luomiseen. New-sanan käyttöä vältettiin ja eri peliobjekteja pyrittiin uudelleenkäyttämään mahdollisimman paljon, jotta vältyttäisiin turhilta, erittäin raskailta System.GC.Collect()-kutsuilta. Edellä mainitun Garbage-Collector-kutsun tarkoituksena on automaattisesti huoltaa muistia tuhoamalla objektit, joihin ei viitata enää. Tämä on yksi JIT:n huonoimpia puolia peliohjelmoinnin kannalta, sillä se voi aiheuttaa merkittävääkin nykimistä peleissä. Tätä kutsua kiertämään on kehitetty ObjectPool eli objektialtaaksi kutsuttu toimintamalli. Kierrätysaltaan periaate on, että tuhoamisen sijaan eri objektit työnnetään talteen altaaseen inaktiivisiksi ja tarpeen vaatiessa uudelleen alustetaan vastaamaan uutta instanssia luokasta. Vain altaan ollessa tyhjä turvaudutaan objektien todelliseen luontiin. Allas on yksinkertaistettuna tavallisesti laajennettava lista, jonka ympärille on rakennettu kierrätystä tukevia toimintoja. Omaa varsinaista objektien kierrätysallasta ei kuitenkaan luotu hallinnoimaan tapahtumaa, koska katsoin, että siihen ei ollut tarvetta, sillä pelissä oli hyvin vähän ajonaikaista instantioitumista ja objektien tuhoutumista.

Unity Pron käyttäjä saa käyttöönsä myös helsinkiläisen Umbra Software Umbra 3 -työkalun. Umbra 3 luo esivalmistellun ”occlusion culling”- eli näkyvyyskarsinta-järjestelmän, joka perustuu binäärisen puumallin. Se jakaa pelimaailman soluihin ja niiden sisäisiin soluihin luoden näin hierarkkisen puun, jota ajon aikana tarkastelemalla pystytään optimoimaan ruudulle piirrettäväksi lähetettävä data kameran sijainnin mukaan. Frustrumkarsinta auttaa poissulkemaan frustumin ulkopuolelle jäävät alueet piirrosta, mutta sen sisäpuolella syvyyssuunnassa eri objektien taakse jäävät objektit joudutaan kuitenkin jossain määrin käsittelemään, vaikka ne lopuksi jäävätkin niiden eteen tulevien mallien taakse piiloon, ja juuri tämän huomioon ottamiseen näkyvyyskarsinta erikoistuu. Unityssä tämä tarkoittaa Umbran kohdalla, että frustumin sisään mutta piiloon jäävien objektien Renderer-komponentti deaktivoidaan (disable) ennen piirtoliukuhihnaa. Umbra 3 käyttää kahta erillistä puuta ajon aikana muuttuville ja muuttumattomille objekteille.

Umbra on täysin prosessoripuolella tapahtuva ja monisäikeistetty eikä se sisällä muistin dynaamisia allokaatioita, joten se ei vaikuta esimerkiksi CarbageCollectorin toimintaan negatiivisesti.

Umbra 3:n käyttö jäi projektin ulkopuolelle, mutta se on yksi niistä ominaisuuksista, jotka otetaan käyttöön seuraavassa, Pro-versiolla tuotetussa versiossa. Sen sijaan projektissa käytettiin mallien yhdistämistä toisiinsa siten, että lähekkäiset, todennäköisesti samaan aikaan kuvassa näkyvät kopiot samasta mallista muunnettiin yhdeksi malliksi, jolloin saatiin tehokkaasti pienennettyä piirtokutsujen määrää.

7.5 Ohjelman rakenne

Peliprojektissa käytettiin ”mustalaatikko”-periaatteista, pelaajakeskistä rakennetta, jossa pelaajan interaktioyritykset laukaisevat eri peliobjektien komponentteja. Interaktiosillan luomiseen käytettiin lähinnä erilaisia kollisiotestejä ja säteiden ampumista. Peliobjektit jäsenneltiin sijaitsemaan tyyppinsä mukaisissa avaruuksissa, mikä helpottaa samantyyppisten objektien käsittelyä ja hallinnointia. Eri objekteille luotiin myös eri kerroksia (Layer), jotta saatiin optimoitua eri peliobjektien välistä keskustelua. Kerrokset lähinnä varmistivat sen, että objektien toiminnot tutkivat vain interaktioon kykenevät objektit kaikkien pelin objektien sijaan. Esimerkkinä tästä on vihollisten havainnointi ja hyökkäys. Havainnoidessaan säteillään ympäristöä vihollisten ei tarvitse välittää esimerkiksi sumusta, vaan säteet voidaan laittaa hylkäämään objektit, jotka sijaitsevat tai eivät sijaitse halutulla tasolla bittimaskeja käyttäen. Hyökkäysten ei tarvitse välittää mistään muusta kuin pelaajaobjektista.

Unityn eri pelinosiot, kuten kentät ja menut, voidaan jakaa erillisiin ”Sceneihin” eli näkymiin tai tasoihin. Tällöin välimuistiin latautuu vain tietty alue pelistä. Projektissa käytettiin seuraavanlaista näkymärakennetta.

Logo

Erillinen näkymä, joka esittää vain oman ”Palikka”-logon (kuva 38) ja josta poistutaan päävalikkoon painamalla mitä tahansa näppäintä.

Kuva 38. Palikka Logo. Insinöörityön tekijän käyttämä tunnus peleissään.

Päävalikko

Päämenu sisältää valikon ääni- ja grafiikka-asetuksille, hahmon luonnin ja tallennettujen pelien lataamisen (kuva 39).

Kuva 39. Pelin päävalikko.

Hahmon luonti

Hahmoa luotaessa hahmolle valitaan etunimi ja sukunimi ja sille jaetaan eri ominaisuuksiin viisi kykypistettä. Eri ominaisuuksia ovat voima, nopeus, kestävyys ja akrobatia.

Voima määrittää, kuinka suurta vahinkoa pelaaja pystyy aiheuttamaan hyökkäyksillään ja kuinka painavia esineitä nostamaan ja liikuttamaan.

Nopeus määrittää, kuinka nopeasti pelaaja pystyy liikkumaan.

Kestävyys kertoo, kuinka paljon vahinkoa pelaaja pystyy sietämään, ennen kuin kuolee.

Akrobatia määrittää, kuinka korkealle pelaaja pystyy hyppäämään.

Edellä mainittuja ominaisuuksia pystyy pelissä kasvattamaan erilaisilla esineillä, kuten aseilla tai ruualla. Alkuperäisenä ajatuksena oli, että ominaisuudet kasvaisivat mitä enemmän niitä käyttäisi, mutta ominaisuudet jäivät hieman enemmän taka-alalle, kuin oli suunniteltu, ja siksi oli luontevampaa rajoittaa niiden kasvuakin. Tulevissa versioissa niistä ehkä otetaan enemmän irti.

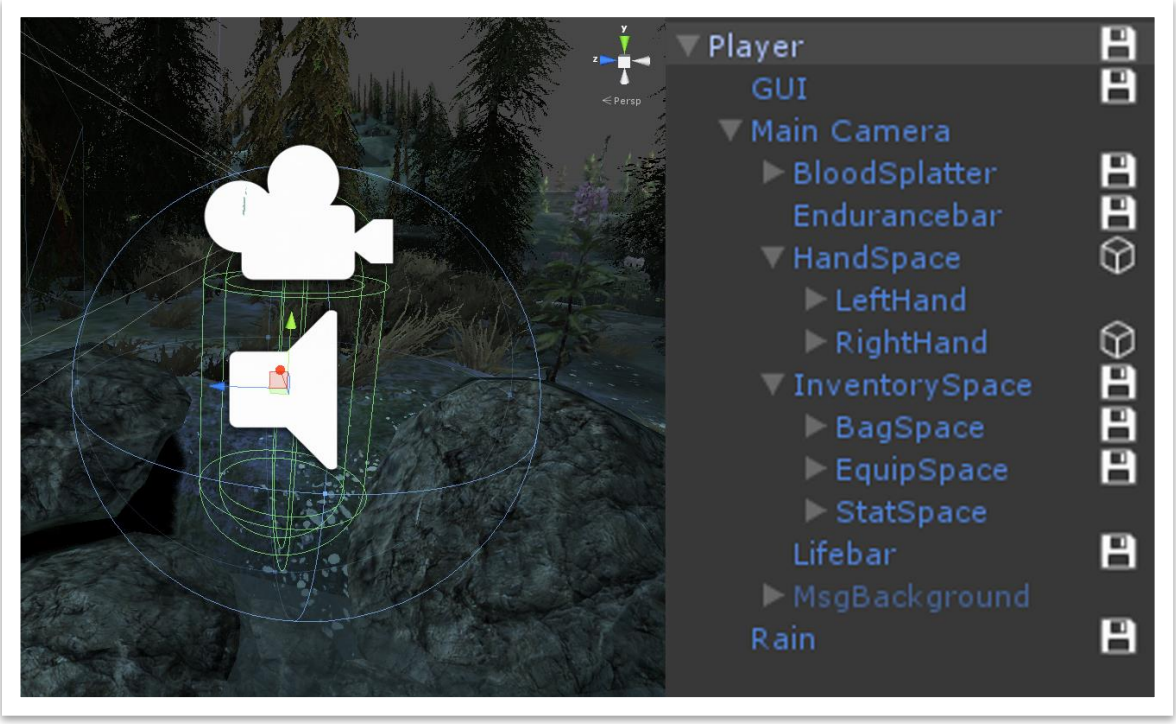
Hahmo

Suurin osa pelin logiikasta sijaitsee pelaajahahmossa. Pelaajahahmon hierarkkinen rakenne on seuraava:

Pelaaja-objekti sisältää liikkumiseen ja muiden peliobjektien väliseen interaktioon tarvittavat komentorivit ja horisontaalisen näkökentän hallinnan. Seuraavana hierarkiassa tulee GUI-peliobjekti, joka sisältää GUI-komponentin hallinnoimaan pelimaailman ulkopuolisia asioita: valikoita, inventaariota ja vasemman laidan viestilaatikon. Rain on peliobjekti, joka hallinnoi sadepartikkeliefektiä ja päättää, milloin sadetta esiintyy. Sade tulee lokaalissa koordinaatistossa laskennallisista syistä. Kun ”sadepilvi” pysyy jatkuvasti pelaajan yllä, ei tarvitse luoda koko pelialueen kokoista, partikkelimäärältään moninkertaista säärintamaa, joka tuhoaisi pelin FPS:n. MainCamera sisältää skybox-materiaalin ja vertikaalisen näkökentän hallinnan ja toimii pelaajan näkökenttänä. Kamera-avaruudesta sijaitsevat BloodSplatter,

Endurance- ja Lifebar, HandSpace, InventarySpace ja Deaktivoitu MsgBackground, joka toimii taustagrafiikkana varsinaisille viesteille pelissä. BloodSplatter on graafinen elementti, joka tulee näkyviin pelaajaa vahingoitettaessa ja on nimensä mukaisesti ”veriläiskiä kameranruudulla”. Endurance- ja Lifebar ovat GUI:n graafiset elementit indikoivat pelaajan iskujen voimakkuutta ja kestävyyttä. HandSpace sisältää pelaajan käsiobjektit, joista vasen lähinnä kuljettaa tehtävien kannalta oleellisia esineitä. Oikea käsi on olennainen taisteluissa. Oikea käsi hallinnoi myös hyökkäyksiä, käytössä olevaa asetta sekä näiden animaatioita.

InventorySpace on GUI-alue, jossa pelaajan inventaario, hahmon varustus ja ominaisuudet ovat näkyvillä. Pelaaja-objektin hierarkia on esitetty kuvassa 40.



Kuva 40. Pelaajahahmon hierarkkinen rakenne.

Fysiikka

Pelissä käytettiin hyödyksi Unityn fysiikkamoottoria. Pelin tehtävissä ja ongelmien ratkaisuissa pyrittiinkin hyödyntämään sitä mahdollisuuksien mukaan. Esimerkiksi pelin alussa pelaajan herätessä maakellarista hänen täytyy koota sieltä löytyvistä esineistä itselleen torni päästäkseen pois. Fysiikka ei kuitenkaan ole käytössä universaalisti kaikessa, vaan ainoastaan irtoesineissä, vihollisissa ja pelaajassa. Esineitä pystyy

kantamaan, heittämään ja työntämään, mikäli niiden massa on tarpeeksi alhainen suhteessa pelaajan voimaan.

Viholliset

Koska peliä kehittäessä käytössä oli vain ilmainen Unity ja koska kyseessä oli yhden hengen projekti, vihollisten tekoäly jäi jotakuinkin hyönteismäiseksi, eli eri ärsykkeisiin vastaamiseksi ja yksinkertaiseksi liikeratojen tunnistukseksi resurssien puutteen vuoksi. Nyt liikkuminen tapahtuu laskennallisesti raskaalla säteiden ampumisella. Säteitä ammutaan 120 asteen alueelle eli ihmisen tarkan oletusnäkökentän mukaisesti, vihollisen näkökyvyn etäisyydelle, 5 asteen välein ja katsotaan, kohtaako se mitään, mihin vihollisen tulisi reagoida. Mikäli säteet eivät havaitse mitään merkitsevää, vihollinen jatkaa vaelteluaan, kun pelaaja on lähettyvillä. Jos vihollinen kuitenkin näkee pelaajan, se asettuu jahtaamistilaan, kunnes se on hyökkäysetäisyydellä, jolloin tila vaihtuu hyökkäykseksi ja vihollinen hyökkää (kuva 41).

Kuva 41. Epäkuolleet luurankosoturit vaeltavat soisen metsän reuna alueilla.

Vihollinen reagoi myös esteisiin yrittämällä korjata liikerataansa. Pelaaja voi jäädä taistelemaan tai pyrkiä karkaamaan tilanteesta pääsemällä tarpeeksi etäälle vihollisesta, jolloin se palaa odotustilaan (kuva 42).

Kuva 42. Mekaaninen hexapodi on pelin suurin ja vaarallisin vihollinen.

Pelissä on myös hiipimiskohtauksia, joissa liian läheinen ja nopea liike havahduttaa viholliset pelaajan läsnäolosta ja saavat ne jahtaamistilaan. Näissä tilanteissa on painotettu hiipimistä, joten vihollisten havaitessa pelaajan on enää vähän tehtävissä.

7.6 Maailman luonti

Maailman luonnissa käytettiin Unityn maastonluontityökaluja maastonpohjan luomiseen ja pienkasvillisuuden ja puuston asetteluun. Sisätilat rakennettiin 3ds Maxissa modulaarisista huoneista. Maaston kuusistoa varten käytettiin pakattua kartoitustekniikkaa, jossa bittikartan eri kanaviin sisällytettiin erikartoituksia. Punainen kanava sisälsi läpikuultavuuskartan, vihreä taas normaalikartan vihreän kanavan, sininen kiiltokartan ja alphakanava normaalikartan punaisen kanavan (kuva 43).

Kuva 43. Suoalueen kuusissa on käytetty pakattua kartoitusta.

Sumu

Pelin ulkoalueilla on sumua, joka on yhdistelmä useampaa eri menetelmää. Ensinnäkin raskas sumu on rakennettu pyörivistä neliöpolygoneista, joissa on läpinäkyvät tekstuurit. Sumupolygonit häviävät taustaan sulautuen, kun pelaajan y-koordinaatti saapuu liian lähelle sumun y-koordinaattia. Tätä sumua on tukemassa partikkeliefektisumu, joka on ilmassa vapaammin leijailevaa sumua, ja horisontin sekoittuminen sumuväriin. Jälkimmäinen on myös yleisesti käytetty optimointimenetelmä, joka tavallisesti vain sekoitetaan horisontin väriin.

Lumi

Pelissä testattiin myös lumen luomista eri pinnoille varjostinohjelmoinnilla. Kuvassa 44 on lumivarjostin, jossa on käytetty verteksivarjostinta siirtämään mallin pinnan kulmapisteitä lumisateen osumasuuntaan, mikä luo lumelle hieman volyymiä, ja pikselivarjostinta muuntamaan lumisateen osumakohdista pinnan tekselien väriä lumen tekstuurin ja värin mukaiseksi. Varjostin toimi hyvin, mutta se jätettiin lopullisesta peliversiosta pois.

```
Shader "Custom/SnowShader" {
    Properties {
    _MainTex ("Base (RGB)", 2D) = "white" {}
    _SnowTex ("Snow Texture", 2D) = "white" {}
    _Bump("Bump",2D) = "bump" {}
    _Snow("Snow Level", Range(-0.5,0.5)) = 0
    _SnowVolume("Snow Volume", Range(1,2)) = 1
    _SnowColor ("Snow Color", Color) = (1.0,1.0,1.0,1.0)
    _SnowDirection ("Snow Direction", Vector) = (0,1,0)
    _SnowDepth ("Snow Depth", Range(0,0.2)) = 0.1

    }

    SubShader {

        Tags { "RenderType"="Opaque" }
        LOD 200

        CGPROGRAM

        #pragma surface surf Lambert vertex:vert

        sampler2D _MainTex;
        sampler2D _SnowTex;
        sampler2D _Bump;
        float _Snow;
        float _SnowVolume;
        float _SnowDepth;
        float4 _SnowColor;
        float4 _SnowDirection;

        struct Input {

            float2 uv_MainTex;
            float2 uv_Bump;
            float3 worldNormal;
            INTERNAL_DATA
        };
        //Verteksivarjostin
        void vert(inout appdata_full v) {

            //muunna normaali maailmankoordinaatistoon
            float4 sn = mul(UNITY_MATRIX_IT_MV, _SnowDirection);
            //muunna verteksin sijaintia lumen määrän mukaan
            v.vertex.xyz += (sn.xyz + v.normal ) * _SnowDepth
                        * _Snow* int(clamp(dot(v.normal, sn.xyz)/lerp(1,-1,_Snow*0.6),0,1));

        }
        //Pikselivarjostin
        void surf (Input IN, inout SurfaceOutput o) {

            half4 mainText = tex2D (_MainTex, IN.uv_MainTex);
            half4 snowText = tex2D (_SnowTex, IN.uv_MainTex);
            //hae lumelle normaalikartta
            o.Normal = UnpackNormal(tex2D(_Bump,IN.uv_Bump));

            //Laske pinnan normaalin ja lumisateen suunnan pistetulo
            //ja käytä lumen volyymia korkeimpana mahdollisena arvona.
            float snow=clamp(dot(WorldNormalVector(IN, o.Normal),
                        _SnowDirection.xyz)/lerp(1,-1,_Snow),0,_SnowVolume);
            //painota pikselin väriä osuvan lumen määrän mukaan lumen tekstuurilla ja värillä.
            o.Albedo = _SnowColor.rgb *snowText* snow * snow+mainText.rgb+mainText.rgb * (1-s);
            o.Alpha = 1;
        }
        ENDCG
    }
    FallBack "Diffuse"
}
```

Kuva 44. Lumivarjostin.

Valaistus

Maaston ja puiden varjojen luontiin käytettiin Unityn mukana tullutta valokartoitustyökalua, Beästiä. Koska reaaliaikaisten varjojen käyttö on laskennallisesti hyvin raskasta, Beäst luo ja laskee pelimaailman objekteille valokartoitukset halutuin parametrein ja valonlähtein.

Pelimoottoreiden valon esilaskenta on yksi merkittävistä syistä, joiden vuoksi eri mallien UVW-karttojen tulisi mahtua koordinaattien 0,0 ja 1,1 väliin eli neliön muotoiseen tilaan, joka usein on mallinnusohjelmissa valmiiksi korostettuna. Tämä siksi, koska valokartoitukset halutaan laskea pelimoottorin puolella automaattisesti, jotta valonlähteen siirtämisestä johtuvat muutokset varjojen sijainneissa saataisiin päivitettyä mahdollisimman kivuttomasti. Koska kyseessä on automatisoitu toiminto, ja kaikki kartoitusbittikarttojen tekselien UVW-koordinaatit sijaitsevat edellä mainitun alueen sisäpuolella UVW-kartta pois lukien (vaikka ylitsemenevät alueet todellisuudessa katkaistaan myös pysymään kyseisellä alueella tavallisesti tekstuurin toistettavuuden vuoksi), täytyy myös valaistusbittikartan kattaa vain 0,0:n ja 1,1:n välinen neliönmuotoinen alue. Jos UVW-kartta ulottuu neliön ulkopuolelle, tapahtuu kartoitusten kohdalla laatoitus efekti, joka toimii sen huomioivien tekstuurien kanssa hyvin eli ylimenevä alue katkeaa ruudukkomaisesti myös UVW-yksikön sisään. Koska valaistus ei ole laatoittuva, vaan pinnan eri kohdilla on erilaiset valoarvot, tämä ei tietenkään toimi. Unityssä on kuitenkin otettu ongelma huomioon ja malleja siihen tuotaessa voi määrittää myös mallille automaattisesti generoitavan, toisen UVW-kartan erityisesti valokartoitusta varten. [16.]

Piirtopolut

Unityn kuvanpiirrolle voi määrittää erilaisia piirtopolkuja PlayerSettings-valikosta tai vaihtoehtoisesti suoraan kameralle. Piirtopolut määrittävät lähinnä, miten valaistusta ja varjoja tullaan käsittelemään. Unity tukee kolmea eri piirtopolkua: kulmapistevalaistua (vertex lit), suoraa piirtoa (Forward) ja laskennallista valaistuspiirtoa (Deferred Lightning). Laskennallinen valaistuspolku on ainoastaan Pro-version ominaisuus. [16.]

Kulmapisteen valaistuspiirto (Vertex Lit)

Kulmapisteen valaistuspiirto on näistä kolmesta kaikkein laskennallisesti kevyin ja parhaiten tuettu laitteistopuolella. Kulmapisteen valaistuspiirto tehdään yhdellä kertaa siten, että kaikista valonlähteistä osuva valo lasketaan mallin kulmapisteiden kohdalta ja väliin jäävän pinnan valaistus interpoloituu näiden arvojen mukaisesti. [16.]

Koska valo lasketaan vain kulmapisteiden kohdalta, tämä piirtopolku ei tue suurinta osaa pikselitasolla tapahtuvista efekteistä, kuten varjoja, normaakartoituksia tai tarkkoja valaistuskarttoja.

Suorapiirto (Forward)

Suorapiirrossa jokaiselle objektille suoritetaan yksi tai useampi laskentakerta, riippuen malliin vaikuttavista valoista. Myös valoja itsessään käsitellään poikkeavasti toisistaan riippuen niiden intensiteetistä ja asetuksista. Esimerkiksi tietty määrä kirkaimmista mallin pintaan osuvista valoista käsitellään pikselitasolla. Unityssä neljään pistevalonlähteeseen asti niitä käsitellään kulmapistetasolla ja loput käsitellään ”Palloharmonisesti” (Spherical Harmonics), joka on hyvin nopeaa karkeaa laskentaa pintaan osuvalle valolle. Suorassa piirrossa, mikäli valonlähde on asetettu piirtotilaltaan epätärkeäksi (Not Important), se tullaan aina käsittelemään joko kulmapistekohtaisesti tai palloharmonisesti. Valot, jotka taas on asetettu tärkeiksi (Important), käsitellään aina pikselitasolla. [16.]

Laskennallinen valaistus (Deferred lightning)

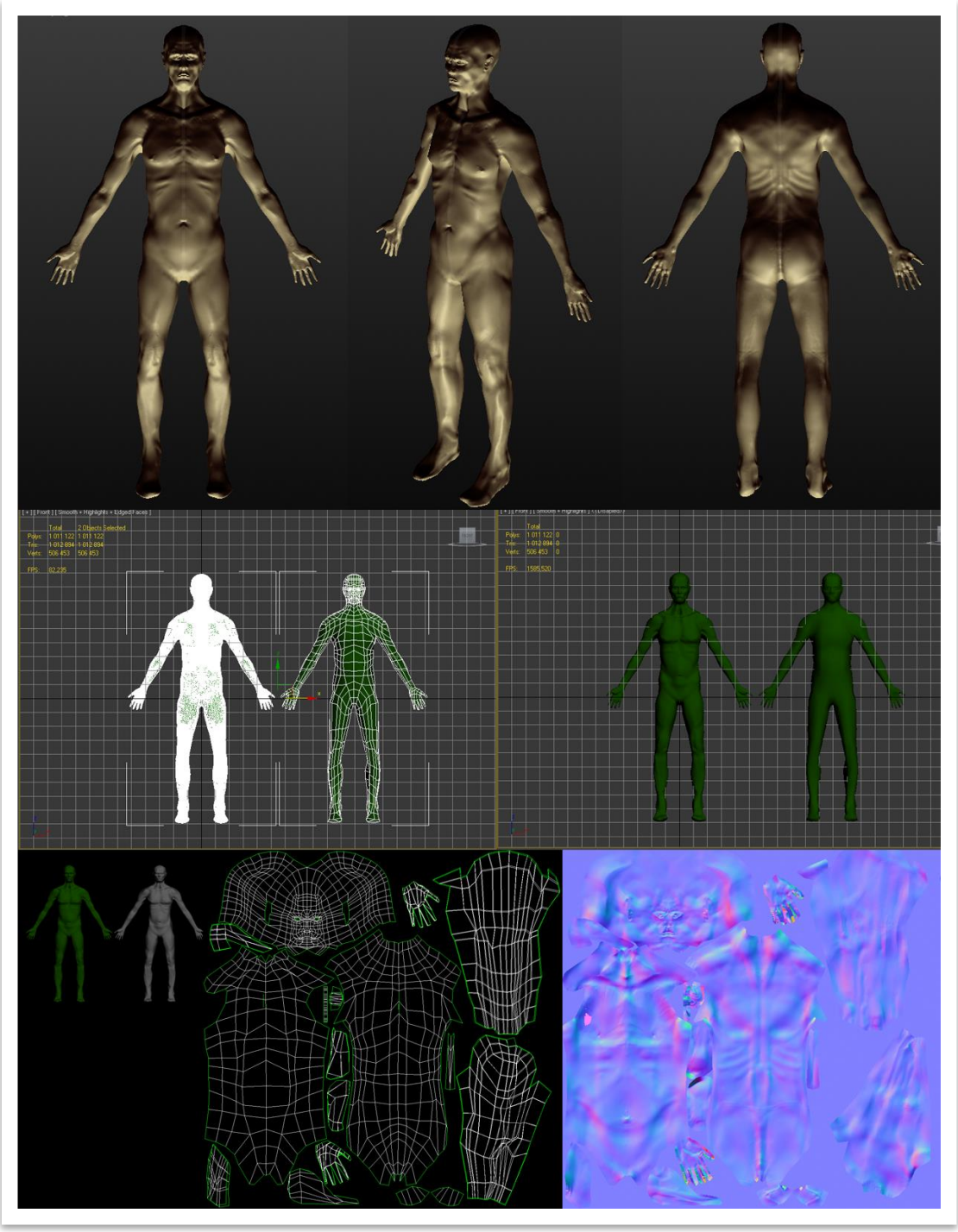
Laskennallinen valaistus on laskennallisesti raskain vaihtoehto: siinä ei ole ylärajaa mallin pintaan vaikuttavien valonlähteiden määrälle, ja ne kaikki käsitellään pikselitasolla. Laskennallisessa valaistuksessa on myös se etu että laskennan määrä on suhteellinen pikselien määrään, joihin valo yltää. Tällöin ei ole merkitystä, kuinka montaa eri objektia valo valaisee ja laskennan määrää voidaan pitää kurissa pitämässä valot pieninä. Haittapuolina laskennallisessa valaistuksessa on anti-aliasioinnin tuen puute ja osittain läpinäkyvät mallit, joihin joudutaan käyttämään suoraa piirtopolkua. [16]

Äänet

Pelin äänimaailma koostuu pääosin tausta-ambienssista, liikuteltavien peliobjektien elementtiäänistä ja vihollisten ääniefekteistä. Jokaisella vihollisella on ääniefektit seuraaville tiloille: hyökkäämiselle, askelille, jahtaamiselle, osumiselle ja kuolemiselle. Ääniefekteihin raakamateriaali on haettu internetistä ilmaislevitykseen tarkoitetuilta sivuilta, kuten SoundBible (http://soundbible.com/). Tämän jälkeen materiaali on editoitu halutunlaiseksi Audacity-audioeditoriohjelmalla (http://audacity.sourceforge.net/). Käytännössä on pyritty siihen, että vain pitkäkestoiset äänet, kuten tausta-ambienssi, on pidetty .mp3-muodossa tilan säästämiseksi, mutta pienet ääniefektit, kuten oven narina tai vihollisten äänet ovat .wav-formaatissa. Pakattu .mp3 hidastaa toistoa, koska se joudutaan ensin purkamaan, kun taas pakkaamattomana .wav on laskennallisesti tehokkaampi vaihtoehto äänille mutta vieden kuitenkin merkittävästi enemmän muistia. Ambientti- taustamusiikin teki opiskelutoverini Jani Miettinen, mistä suuri kiitos hänelle.

7.8 Projektin eteneminen

Projekti käynnistyi tammikuussa 2013 mallintamalla peliin ensimmäinen malli, androgyynin humanoidi (kuva 45) jonka oli tarkoitus toimia hahmomallipohjana peliin tuleville humanoideille ja samalla syventää omia animointitaitojani. Mallinnus alkoi luomalla nelipolygoneista koostuva pelimalli, joka siirrettiin Sculptris-muovausohjelmaan, jossa mallinnettiin pinnan yksityiskohdat. Näin syntyi korkeapolygoninen mallin. Malli vietiin sen jälkeen 3ds Maxiin, jossa normaalikartta luotiin manuaalisesti. Tuolloin animointi vei paljon aikaa ja oli hidasta toteuttaa, mutta se nopeutui jokaisen riggauksen jälkeen merkittävästi, kunnes projektin loppupuolella kyseinen vaihe ei kestänyt kuin murto-osan alkuajoista. Hahmoa ei kuitenkaan käytetty lopullisessa pelissä pohjana muille kuin muumioidulle linnan entisen talonmiehen vaimolle.



Kuva 45. Androgyyni malli pelin hahmojen pohjamalliksi. Hahmolle luotiin normaali- ja tekstuurikartat ja luujärjestelmä animaatioineen.

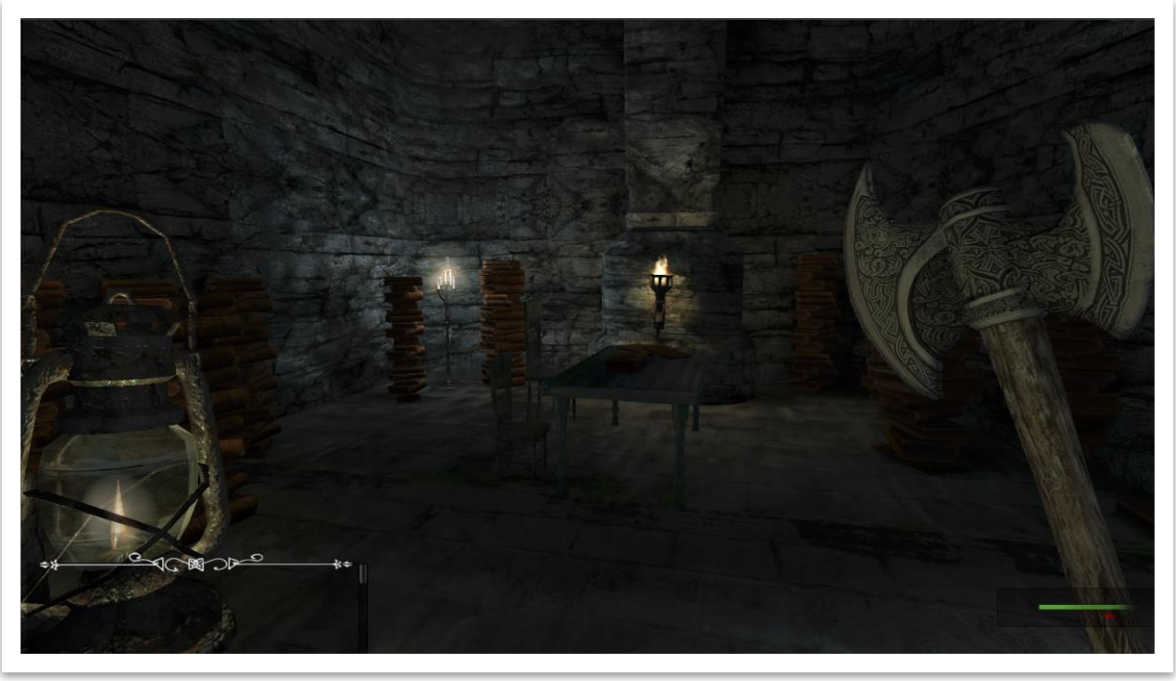
Edellä mainittu tapahtui jo ennen varsinaista pelisuunnitelman kehittelyä ja ensimmäistä varsinaista projektiin liittyvää tapaamista vain karkean peli-idean pohjalta. Hahmon valmistuttua alkoi itse pelimaailman tunnelman, juonen ja ympäristön suunnittelu. Referenssiksi kerättiin kuva-atlaksia ja tutkittiin mahdollisia rakennus- ja esinekohteita Suomen rajojen sisältä, joita voisi keilata peliin rekvisiitaksi. Myös niiden pohjalta luotiin kuva-atlas.

Alkuperäinen peli-idea oli suunniteltu toimimaan kolmannesta persoonasta kuvattuna, lajityyppinään rooliseikkailu, mutta juuri animointitaitojen silloinen puute sai pohtimaan mahdollista toisenlaista lähestymistapaa, jotain, missä animaatioita ei tarvisi niin paljoa, ja ajatus ensimmäisestä persoonasta kuvatusta pelistä syntyi (kuva 46). Ennen suunnanmuutosta oli kuitenkin jo nähty melkoisesti vaivaa esimerkiksi älykkään kameran luomisessa, ja se jätettiin tulevaisuuden versioita varten, mikäli peliin haluttaisiin lisätä kolmannen persoonan näkymä. Päätös pelaajan perspektiivin muuttumisesta lisäsi tempoa huomattavasti, ja se osoittautuikin erittäin hyväksi ratkaisuksi.

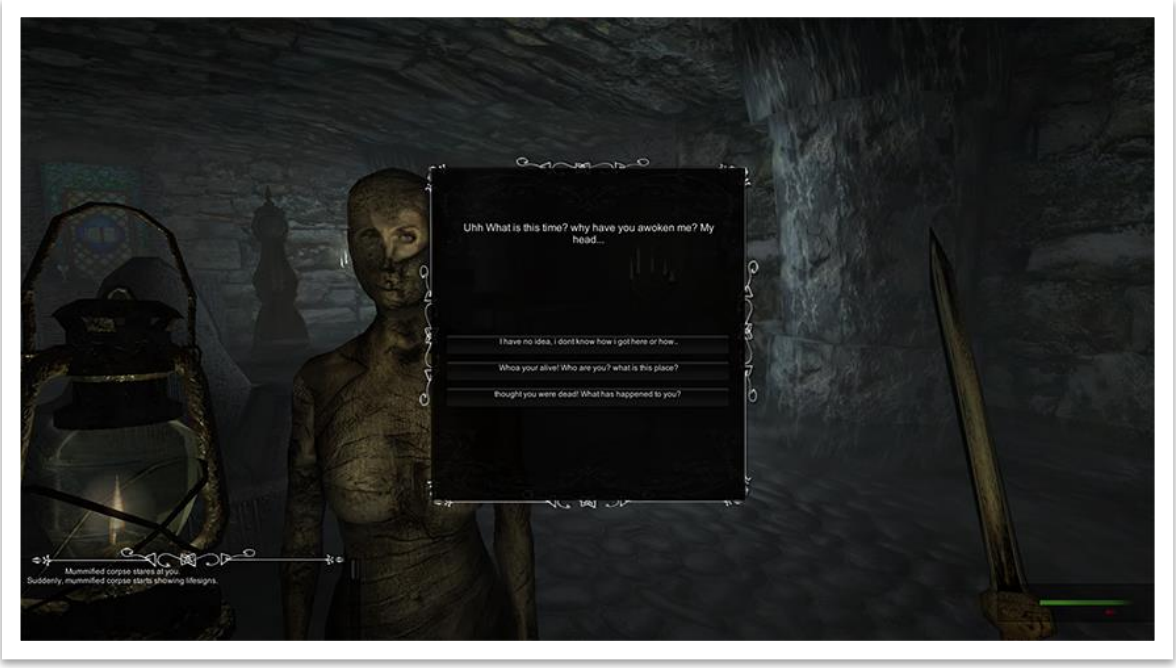
Ensimmäisiä komponentteja, joita peliin ohjelmoitiin, olivat vihollisen tekoäly, josta luotiin useampi versio, muunnelma Unityn CharacterMotor-luokasta ja käyttäjän syötettä varten FPSInputControlle-luokka. Koska tarkoitus oli luoda peliin epätodellista tunnelmaa, pyrittiin erilaisia ”linssejä” kamerassa. Aluksi käytössä olikin ihmisen näkökenttää huomattavasti laajempaa näkökulmaa.

Kun pelin maailmasta oli ensimmäinen vedos valmiina, alkoi taistelumekanismin ja dialogijärjestelmän suunnittelu. Taistelumekanismi sisälsi kaksi erilaista hyökkäystä, piston ja lyönnin. Näiden iskujen aiheuttaman vahingon määrää pystyi nostamaan lataamalla niitä pitämällä iskua vastaavaa hiiren näppäintä pohjassa tarpeeksi kauan. Tämä oli tarkoitus pitää vain alkuasetelmana taistelujärjestelmille ja myöhemmin tarkoituksena oli siirtyä vapaammin ohjattavaan eri aseiden käsittelyyn.

Dialogijärjestelmä (kuva 47) toteutettiin aluksi staattisena luokkana joka toimi eräänlaisena kortisto-ohjelmana. Eri henkilöillä oli oma, uniikki tunnusluku jolla haettiin oikeat dialogit, joilla puolestaan oli uniikit indeksoivat arvot. Tämä järjestelmä oli kömpelö, ja se muuttuikin ajan myötä enemmän Unityn sisäistä arkkitehtuuria hyödyntäväksi eli komponenttipohjaiseksi. Jokainen hahmo muutettiin sisältämään oma dialogikomponentti, joka salli editorissa eri puheiden muuttamisen.



Kuva 46. Linnan kirjasto ensimmäisen persoonan näkökulmasta.



Kuva 47. Muumiotu majakanpitäjä ja dialogisysteemi.

Kun taistelumekanismin ja dialogijärjestelmän ensimmäiset versiot olivat koeajossa, aloitettiin inventaario-järjestelmän suunnittelu (kuva 48). Inventaarion GUI-elementtejä harkittiin pitkään, mutta lopulta päädyttiin täysin kolmiulotteiseen, kaksiulotteisuuden illuusiolla varustettuna. Tämä siksi, että haluttaessa pystyttäisiin käyttämään samoja peliob-

jekteja inventaariossa kuin itse pelissä, ilman erillisten kaksiulotteisten kuvaikonien luontia. Kolmiulotteinen inventaario myös mahdollistaa monimuotoisempien animaatioiden käytön tarvittaessa ja jatkokehityksenä tuotettavan pyöriteltävän ”mannekiinin”, jossa senhetkinen puettu varustus näkyy. Inventaariota lähdettiin tekemään tunnettujen pelien inventaariojärjestelmiä mukaillen, pienin poikkeuksin. Inventaarion tavaratila määräytyy pelaajan voiman ja mahdollisten lisäesineiden mukaan. Tavaratilaa kuvaavat pienet hammasrattaat, joiden sisään mahtuu aina yksi eri esine. Esineitä pystyy liikuttamaan eri rattaasta toiseen, ja tiputtamaan pelikentälle. Kun vie hiiren niiden yläpuolelle ja pitää hiirtä pohjassa, tuo graafinen käyttöliittymä esiin osoitetun esineen Item-komponenttiin tallennettuja, tarkempia tietoja, kuten nimen, varustetyypin, arvon ja mahdollisen pidemmän kuvauksen.

Kuva 48. Pelaajan inventaarionäkymä. Näkymä on jaettu kolmeen eri osaan: hahmon perustietoihin, varustenäkymään ja tavaratilanäkymään.

Peliin oli tarkoitus myös ottaa mukaan tallennusmahdollisuus. Koska kyseessä olisi muuten ollut hyvinkin aikaa vievä toteutus, päädyttiin käyttämään Save Game Manager -nimistä liitännäistä. Save Game Manager osoittautui suunnitelluksi aikaisemmille Unity-versioille, joten sen käyttöönotto ei ollut täysin mutkatonta, vaan lähdekoodia täytyi päivittää itse, koska virallisia päivityksiä ei ollut tiedossa.

Peliin haluttiin myös jonkinlaista sään vaihtelua, ja koska kyseessä oli synkkä-tunnelmainen, surrealistinen peli, päädyin tekemään satunnaisesti eri ulkoalueille mentäessä ilmaantuvan sateen. Sateen simulointiin käytettiin Unityn tarjoamaa partikkelijärjestelmää pohjana (kuva 49).

Kuva 49. Pelin sade toteutettiin Unityn partikkelijärjestelmää hyväksi käyttäen.

Pelin testaus toteutettiin muutama henkilön avulla parissa eri kohtaa kehitystä. Testaussessiot paljastivat lähinnä visuaalisia virheitä ja joitakin fysiikkaan ja vihollisten tekoälyyn liittyviä ongelmia. Testaus toimi sanallisen palautteen periaatteella, eikä sitä varsinaisesti dokumentoitu. Testausta aiotaan painottaa tulevaisuudessa huomattavasti enemmän.

8 Laserkeilauksen sovellutuksia peliympäristöihin

Lasermittauksella on kiistaton etu nykymaailmaan sijoittuvien pelien maailmojen mallintamisessa. Peli voidaan sijoittaa olemassa oleviin paikkoihin, joissa vieläpä kaikki on aivan kuin todellisen maailman vastaavassa sijainnissa. Autopelit ovat erittäin hyvä esimerkki tästä. Ralliradat ja autot ovat suoraan mitattavissa, eivätkä ne ainoastaan nopeuta pelin valmistumista vaan antavat sille merkittävää lisäarvoa. Kukapa ei haluaisi

kokeilla samoja kilparatoja joissa maailmanluokan mestarit ajavat. Mikäli lasermittauksella tehdään nykymaailmasta kopio pelimaailmaan, voidaan pelimaailmaa käyttää monessa sovelluksessa, ja sen jälkeen pelimaailma ei ole ainoastaan pelaamista varten, vaan toimii myös erilaisten sovellusten tarjoajana.

Esimerkiksi lisättyä todellisuutta voidaan esittää pelimaailmassa. Tätä voidaan hyödyntää muun muuassa kaupunkisuunnittelussa. Vastaavasti lisätyn todellisuuden avulla voidaan pelimaailmaan rakentaa paikkatietopalveluita. Tästä on esimerkkinä Geodeettisen laitoksen tekemä reaaliaikaisesti päivittyvä aikataulunäyttö, joka näyttää Tapiolan keskuksen Merituulentien pysäkiltä seuraavaksi lähtevät linja-autovuorot. Kaupat voivat myös käyttää pelimaailmaa tuotteittensa mainostamisessa, ja mahdollisesti nuoremmat asiakkaat mieluummin seuraisivat kauppojen tarjouksia, jos ne liittyivät tällaiseen pelimaailmaan. [27.]

Pelimoottorit luovat hyvän alustan reaaliaikaiselle mallin päivittämiselle tuoreella informaatiolla tarjoten samalla myös tarkasteluympäristön ja haluta lisättyä todellisuutta [1, s. 15].

Geokätkentää voi pelimaailman ja virtuaalitodellisuuden kanssa toteuttaa myös siten, että varsinainen kätkö löytyy pelimaailmasta, kun älypuhelimen antama paikannus vastaa kätkön koordinaatteja. Kätköt voivat olla tällöin myös palkintoja, lahjoja ja mainoksia käyttäjille. Kätköjen kokonaisuus voi myös muodostaa kokonaisuudessaan seikkailupelin. Todellisuutta vastaavalla pelimaailmalla voidaan yhdistää virtuaalinen ja fyysinen maailma uusilla tavoilla. [27.]

Pelimaailmaan on mahdollista tuoda ympäröivästä maailmasta myös attribuuttitietoja, kuten BIM-tietoja, jotka ovat tärkeitä rakennetun ympäristön mallintamisessa. Pelimaailmaa on mahdollista käyttää myös dokumentoinnissa. Esimerkkinä voisi mainita talonrakennusprosessin. Useilla kaupungeilla on antaa laserkeilauksen tiedot rakentajille. Tällöin käyttäjät voisivat optimoida rakennuksen sijainnin tontilla suhteessa olemassa olevaan puustoon, korkeusmalliin ja näkymiin. Rakentamisen suunnitelmamalli voidaan viedä pelimoottoriin yhdessä ympäristömallin kanssa. Rakennuksen aikana tapahtuvia puutteita voisi seurata muun muassa valituslistojen avulla. Pelin ja pelimoottorin etu valituslistoissa olisi kuitenkin, että kutakin valitusta kohden löytyisi kameralla otettu kuva viasta ja vika olisi paikannettu oikeaan paikkaan rakennuksessa. Käyttäjät voisivat kukin

laittaa valituksensa reaaliajassa samaan malliin, joka olisi siten reaaliaikainen dokumentaatio puutelistalle, jota voisi tarkastella varsin helposti virtuaalisesti käymättä aina paikan päällä. [27.]

9 Yhteenveto

Insinöörityössä aikana perehdyttiin syvällisemmin jokaiseen pelinkehityksen osa-alueeseen. Työn aikana havaittiin piirtoliukuhihnan tuntemuksen tärkeys ja erilaisten optimointikeinojen hyödyt pelin sulavuuden parantamisessa. 3ds Max-, Unity-, Photoshop- ja Audacity-ohjelmien tuntemus syveni huomattavasti. Peliprojektin aikana huomattiin pelilajin ja -tyypin olevan yhden hengen tuotantotiimille melkoinen haaste ja havaittiin, että vastaavanlaisissa projekteissa olisi hyvä olla ainakin lisämallintaja resurssien tasaisemmin jakamiseksi.

Teollisuuskeilaimella tuotettiin insinöörityössä ritariteroittimesta, kuluneesta kirjasta ja kotkapatsaasta malli. Maalaserkeilausta hyödynnettiin rakennuksen julkisivun, henkilön ja muistomerkin pistepilven tekemiseen. Käsin tehtyä kuvapohjaista tarkkaa mittaustekniikkaa hyödyntäen mallinnettiin Olavinlinna ja Helsingin Sinebrychoffin puistossa sijaitseva torni. Korkeapolygoniset mallit siirrettiin 3ds Maxiin, jossa niille tehtiin polygonimäärän vähentäminen ja jälkikäsittely pelimaailmaan sopivan mallin luomiseksi. Lopputuloksena syntyi pelikelpoisia malleja.

Valmiit mallit siirrettiin Unity-pelimoottoriin ja peliprojektiin onnistuneesti. Lisää malleja mallinnettiin suoraan 3ds Maxissa peliä varten. Pelin ohjelmoinnissa hyödynnettiin C#:a, Unityscriptiä ja Shaderlab (Cg) -ohjelmointikieltä. Peliprojektissa ei käytetty kolmannen osapuolen malleja tai komentosarjoja vaan ne tehtiin itse (poikkeuksena Unityn mukana tulleet komentosarjat ja tallennuslisäosa). Peliprojektista itsestään saatiin toimiva ensimmäinen versio valmiiksi, mutta tulevaisuudessa on tarkoitus lisätä parempi tekoäly ja reitinhaku vihollisille ja myös hioa pelattavuutta ja juonen kuljetusta.

Projektin pohjalta havaittiin, että lasermitattujen mallien hyödyntäminen suoraan peleissä ei ole vielä suotavaa niiden pinnan kolmioinnin ja epäoptimaalisten UVW-karttojen takia. Yksinkertaiset ja helpot muodot on nopeampi mallintaa manuaalisesti ja lopputuloksesta syntyy siistimpi ja optimaalisempi. Realistisuus kuitenkin syntyy pienistä yksityiskohdista, ja vaikka niitä voidaan jäljitellä, se on aikaa vievää eikä ilman huomattavia

taitoja täysin todenmukaista. On kuitenkin olemassa pinnalta ja muodoltaan monimutkaisia kohteita, joiden mallintaminen on työläs prosessi. Tällöin lähes automaattisesti luodun, mittatarkan mallin edut verrattuna manuaalisesti luotuun, täydellisen kolmipinnan malliin ovat merkittävät etenkin staattisten eli liikkumattomien mallien tapauksessa.

Työssä huomattiin, että lasermittausmenetelmillä tuotetusta korkeapolygonisesta mallista voidaan tuottaa pienipolygoninen, peliympäristöön soveltuva malli ja hyödyntää korkeapolygonista mallia normaalikartan luonnissa. Ongelmakohdaksi nousi tietyissä tilanteissa keilausohjelmistojen automatisoitu UVW-kartoitus, joka ei vastannut laadultaan pelimalleille sopivaa karttaa.

Keilaustapahtuman yhteydessä kerättyjen valokuvien käyttö mallien tekstuurina on merkittävästi raskaampi vaihtoehto kuin manuaalisesti luotujen, toistuvuutta hyödyntävien tekstuurien. Suurien mallien kohdalla pelissä, joka on läheltä kuvattu, jouduttaisiin turvautumaan niin suuriin resoluutioihin, että niiden käyttö ei olisi mielekästä. Valokuvapohjaisissa tekstuureissa on kuitenkin liikkumattomien valonlähteiden ja staattisten objektien yhdistämisessä se erityispiirre, että niiden valo- ja varjokartat ovat hyvin realistisia. Koska valot ja varjot tulevat todellisesta maailmasta ilman erillistä laskentaa, niiden monimutkaisuudelle ja tarkkuudelle ei ole rajoja.

Lasermittauksella on myös kiistaton etu nykymaailmaan sijoittuvien pelien maailmojen mallintamisessa: peli voidaan sijoittaa todellisiin paikkoihin, joissa kaikki on aivan kuin todellisen maailman vastaavassa sijainnissa.

Mittausteknologioiden kehitys laitteiston rakentamisesta aina mallin saattamiseen pelimoottoriin asti on hyvä esimerkki laserkeilausteknologian läpimurrosta informaation tuotannossa ja sen mahdollisuuksia kuluttaja- ja kartoitus- sekä yristysten välisillä (business to business, B2B) markkinoilla. Laserkeilauksen kansainväliset markkinat kasvavatkin jopa 15 prosenttia vuodessa.

Lähteet

1. Kukko, Antero, Jaakkola, Anttoni & Hyyppä, Juha. 2012. GL suunnannäyttäjänä kaupunkien 3D-mallinnuksessa. Geodeettisen laitoksen kolmiulotteinen malli Tapiolan keskuksesta auttaa yhdistämään virtuaalista ja fyysistä maailmaa. Positio 1/2012, s. 13–15.

2. Centre of Excellence In Laser Scanning Research. Verkkodokumentti. Finnish Geodetic Institute. < http://www.fgi.fi/coelasr/ >. Luettu 24.2.2014.

3. Pfeifle, Sam. 2012 Oh, so now laser scanning is video game technology. Verkkodokumentti. < https://www.sparpointgroup.com/Blogs/Head-in-the-Point-Clouds/Oh,-so-now-laser-scanning-is-video-game-technology-/ >. Luettu 12.12.2013.

4. Finney,C Kenneth. 2013. 3D Game Programming All In One. Third Edition. Boston,USA: Cengage Learning.

5. Hyyppä, Hannu & Ahlavuo, Marika. 2012. Smart City – Kilpailukykyinen ja energisoiva kaupunki. Maankäyttö 1/2012, s. 10–13.

6. Cronnvall, Timo, Käknäs, Pasi & Turkka Tommi. 2012. Laserkeilauksen käyttö liiketunneleiden kunnossapidon hallinnassa. Tutkimus. Helsinki: Liikennevirasto.

7. Heiska, Nina. 2010. Maalaserkeilaimet ovat kehittyneet geodeettisiksi mittauslaitteiksi. Maankäyttö 4/2010, s. 14–17.

8. Vaaja, Matti. 2012. Laserkeilauksella yksityiskohtaista tietoa jokiympäristöistä. Positio 4/2012, s.13–15.

9. Hyyppä, Juha, Zhu, Lingli, Liu, Zhengjun, Kaartinen & Harri. Jaakkola, Anttoni. 2013. 3D City Modeling and Visualization for Smart Phone Applications. Hershey, Pennsylvania, USA: IGI Global.

10. Koski, Jarkko. 2001. Laserkeilaus – Uusi ulottuvuus paikkatiedon keräämiseen. Maankäyttö 1/2001, s. 24–26.

11. Hyyppä, Juha. & Hyyppä, Hannu. 2007. Laserkeilaus. Geodeettinen laitos.

12. Kari, Veera. 2011. Laserkeilaus ja pistepilven käsittely ydinvoimarakentamisessa. Opinnäytetyö. Tampereen ammattikorkeakoulu.

13. Virtanen, Juho-Pekka, Turppa, Tuomas & Hyyppä Hannu. 2014. Unity ja laserkeilaus. Työpaperi. Aalto yliopisto.

14. Virtanen, Juho-Pekka, Laserkeilausmallinnus 2013. Työpaperi. Aalto yliopisto.

15. Ruohonen, Sanna. 2007. FARO LS 880 -laserkeilain vapaan keilainaseman menetelmässä. Opinnäytetyö. Tampereen ammattikorkeakoulu.

16. Unity Documentation. Verkkodokumentti. Unity Technologies. < docs.Unity.com/Documentation/> Luettu 11.8.2013.

17. Witters, Koen. deWitters-gameloop. Verkkodokumentti. <http://www.koonsolo.com/news/dewitters-gameloop/> Luettu 6.7.2013.

18. Puhakka, Antti. 2008. 3D-grafiikka. Helsinki: Talentum Media ja Antti Puhakka.

19. Flyktman, Reima. Muisti. Verkkodokumentti. < http://www.kuvakenno.fi/tietokone/muisti.html > Luettu 22.12.2013.

20. OpenGL dokumentaatio. Verkkodokumentti. Khronos Group. <https://www.opengl.org/documentation> . Luettu 11.12.2013.

21. Making better textures for games, 'power of two' and proper image dimensions. Verkkodokumentti. Katsbits. <http://www.katsbits.com/tutorials/textures/make-better-textures-correct-size-and-power-of-two.php>. Luettu 11.12.2013.

22. Marucchi-Foino, Romain. 2012. Game and Graphics Programming for iOS and Android with OpenGL ES 2.0.2012. West-Sussex: United Kingdom: John Wiley & Sons.

23. Perspective and orthographic projection matrix. Verkkodokumentti. Scratchapixel. <http://www.scratchapixel.com/lessons/3d-advanced-lessons/perspective-and-orthographic-projection-matrix/opengl-perspective-projection-matrix >. Luettu 11.11.2013.

24. Dunn, Fletcher & Parberry, Ian. 2002. 3D Math Primer for graphics and Game Development, Plano, Texas: Wordware Publishing.

25. Z-buffering. Verkkodokumentti. Princenton University <https://www.princeton.edu/~achaney/tmve/wiki100k/docs/Z-buffering.html >. Luettu 12.1.2014.

26. Vertex Shaders. Verkkodokumentti. Nvidia Corporation. <http://www.nvidia.com/object/feature_vertexshader.html>. Luettu 22.11.2013.

27. Hyyppä, Juha. 2014. Kaukokartoitus ja fotogrammetria. Geodeettinen laitos. Haastattelu 17.2.2014.