PELIMOOTTORI VR -SOVELLUKSEN ALUSTANA

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Riihimäen kampus, Tietotekniikan koulutusohjelma

Kevät, 2017

Sofia Toivola

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietotekniikan ko.
Riihimäki

Tekijä Sofia Toivola **Vuosi** 2017

Työn nimi Pelimoottori VR -sovelluksen alustana

Työn ohjaaja/t Petri Kuittinen

TIIVISTELMÄ

Työn tavoitteena oli tehdä selvitys virtuaalitodellisuudesta ja pelimoottoreista, jotka tukevat virtuaalitodellisuutta. Samalla tutustuttiin eri virtuaalitodellisuuslaitteistoihin. Käytännön osuuden tavoitteena oli tuottaa erilaisia sovellustestauksia Gear VR -laitteistolle Unity- ja Unreal -pelimoottoreilla. Työ tuotettiin Deemec Oy:lle. Deemec on aiemmin tuottanut renderöityjä 360-kuvia. Sovellustestausten tarkoituksena oli selvittää, mitä kaikkea voidaan pyörittää moottoreilla, esimerkiksi vuorovaikutusten ja mallien monimutkaisuuden osalta.

Kaikki toivotut sovellustestaukset saatiin toimimaan Unity-pelimoottorilla. Myös optimointi saatiin toimimaan hyvin annetuissa raja-arvoissa. Unreal-moottorilla ei saatu yhtä hyviä optimointituloksia, eikä näin ollen lähdetty tekemään monimutkaisempia sovellustestauksia.

Gear VR -laitteisto asettaa tietyt vaatimukset sisällölle, jota halutaan tuottaa. Tietokoneen ympärillä pyörivän virtuaalilaitteiston avulla saadaan luotua graafisesti hienompaa sisältöä, sillä laskentatehoa on huomattavasti enemmän. Mobiili VR -sovelluksia luotaessa on hyvä ymmärtää raja-arvot, joiden sisällä pyöritään. Tämä vaikuttaa esimerkiksi siihen, miten monimutkaisia malleja voidaan pyörittää sovelluksessa, tai kuinka paljon dynaamisia objekteja, animaatioita tai efektejä voi olla näkymässä.

Avainsanat Virtuaalitodellisuus, Pelimoottori, Unity, Unreal

Sivut 50 sivua, joista liitteitä 0 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Information Technology
Riihimäki

Author Sofia Toivola **Year** 2017

Subject Game engine as a platform for VR software development

Supervisors Petri Kuittinen

ABSTRACT

The aim of the thesis was to study virtual reality and game engines that support it. The goal of the practical part was to produce various application tests for Gear VR with Unity and Unreal. The work was commissioned by Deemec Oy. Deemec has produced rendered 360 images before. The purpose of the application tests was to find out what kind of applications could be made for Gear VR, interaction- and complexity wise.

All planned application tests were successfully conducted with the Unity engine. Optimization was also achieved by given limit values. The Unreal engine did not acquire sufficient optimization results and therefore more complex applications were not made.

Gear VR sets certain requirements for the content you want to produce. Using virtual reality PC-hardware, graphically finer content can be created since then computing power is much greater. When creating mobile VR applications, it is good to understand the limits of, for example, how complex models can be implemented into application, or how many dynamic objects, animations or effects can be seen.

Keywords Virtual Reality, Game engine, Unity, Unreal

Pages 50 pages

SISÄLLYS

1 JOHDANTO

1.1 Teoriaosuus

Teoriaosuus sisältää vertailua eri virtuaalitodellisuutta tukevien pelimoottoreiden kanssa sekä tutustumista ylipäänsä virtuaalitodellisuuteen ja virtuaalilaseihin. Siinä määritellään myös projektissa käytettävät tekniikat sekä aikataulu.

1.2 Käytännön osuus

Käytännön toteutuksen tavoitteena on tuottaa prototyyppitason virtuaalitodellisuussovellus tai sovelluksia, joilla voidaan esitellä virtuaalitodellisuusympäristöjä. Eri testiversiot toteutetaan Unity- sekä Unreal -pelimoottoreilla. Sovellukset tulevat sisältämään 3D-malleja ja yksinkertaista vuorovaikutusta maailmassa. Samalla vertaillaan Unityn ja Unrealin hyviä ja huonoja puolia.

Virtuaalitodellisuudelle sovellusten kehittäminen on minulle uusi aihe, johon aion tehdä perusteellisen selvityksen. Lopputuotteessa käytetään Gear VR -laitteistoa, joten tähän tarvitaan tuen pystyttäminen Unitylle sekä Unrealille. Projektiin pitää pystyä tuottamaan 3D-malleja 3ds Maxista. Ympäristössä on myös vuorovaikutusta, kuten liikkumista ympäristössä sekä UI-elementtejä, joilla voidaan käynnistää videoita tai animaatioita. Liikkumisen ja valintojen ohjaaminen tapahtuu Android-peliohjaimen ja katseen suunnan tarkastelun avulla.

Tarkemmin kuvattuna käytännön toteutuksessa ympäristöön tuotuja malleja ja esineitä voi katsoa, mutta niiden läpi ei voi mennä. Esineisiin ei myöskään voi tarttua, eikä niitä voi työntää ympäriinsä. Ympäristössä voi liikkua vapaasti fysiikan lakien mukaan, mutta törmäystarkastelu ja etäisyyden mittaaminen esineistä määrittää, kuinka lähelle niitä pääsee, jotta vältetään kameran leikkautuminen esineen kanssa. Visuaalinen ilme ei tule olemaan AAA-studion tasoa, lähdetään ennemmin prototyyppitason sovelluksesta.

2 VIRTUAALITODELLISUUS JA PELIMOOTTORIT

2.1 Virtuaalitodellisuus

Virtuaalitodellisuus on yksinkertaisimmillaan ihmisen aistimuksien muokkaamista niin, että ihminen kokee toisen, virtuaalisen todellisuuden (Virtual Reality Society n.d.). Keskityn tässä tarkastelemaan virtuaalitodellisuutta erityisesti tietokoneella toteutettujen sovellusten näkökulmasta, vaikka myös esim. teemapuistot tai panorama-maalaukset sopivat virtuaalitodellisuuden määritelmään.

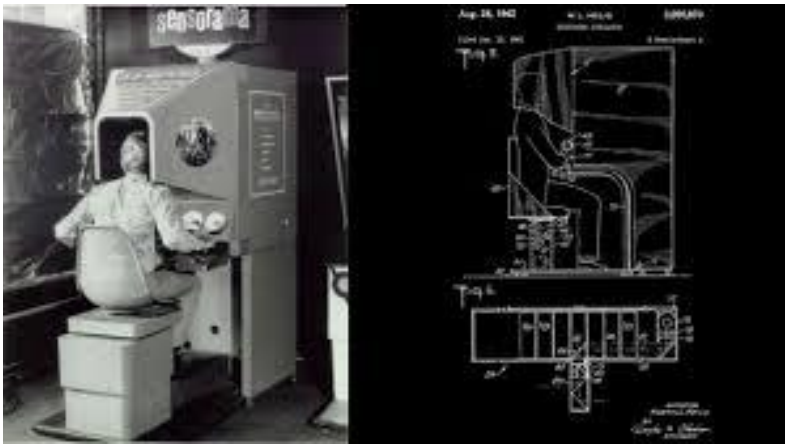
Virtuaalitodellisuudella nykypäinä tarkoitetaan tietokoneella tehtyä 3D-ympäristöä tai 360-kameralla tuotettua kuvaa, jota katsotaan virtuaalilasien kautta (Virtual Reality Society n.d.). Lasien kanssa päätä kääntämällä ihminen pystyy katsomaan maailmaa ympärillään ja kuulemaan äänentoistosta ympäristöön liitettyjä ääniä. Tärkeimpiä muokattavia aistikokemuksia virtuaalitodellisuuden kannalta ovat siis tällä hetkellä näkö- ja kuuloaisti. Virtuaalitodellisuuden tavoitteena on tuottaa käyttäjälle immersio, eli subjektiivinen kokemus jossa ihminen 'unohtaa' todellisen maailman ympäriltään ja kokeekin olevansa sisällä virtuaalisessa todellisuudessa (Virtual Reality Society n.d.).

Uusimpiin virtuaalitodellisuuslaitteistoihin on myös tuotu mukaan pään kääntymisen tarkastelun lisäksi liikkeen tarkastelu sijainnin suhteen. Tämä on toteutettu sekä tarkastelemalla virtuaalilasien eli pään liikettä ja sijaintia sekä käsissä pidettävien ohjainten sijaintia. Tällä ominaisuudella maailmassa voi myös mm. liikkua ilman ohjainta kävelemällä, nähdä kätensä liikkeet virtuaalimaailmassa ja tarkastella maailmaa ilman mekaanista suurennusta/pienennystä (zoom) kumartumalla eteen tai taaksepäin. Nämä ominaisuudet mahdollistavat entistä vahvemman immersion, sillä vuorovaikutus virtuaalisen maailman kanssa on muuttunut entistä samankaltaisemmaksi todellisuuden kanssa.

Virtuaalitodellisuutta käytetään nykyään esimerkiksi peleissä, elokuvissa ja erilaisissa simulaatioissa. Simulaatiosovelluksia löytyy sekä viihde-, markkinointi- että opetuskäyttöön.

2.1.1 Historia

Virtuaalitodellisuus itsessään ei ole uusi keksintö. Ensimmäiset lentosimulaatiolaitteistot löytyvät jo 30-luvulta. Vuonna 1962 patentoitiin Sensorama (kuva 1), arcade-pelikonetta muistuttava videoiden katselulaite, joka loi virtuaalitodellisuuden äänien, stereoskooppisen 3D-näytön, tuulettimien, hajun ja tärisevän tuolin avulla. (Virtual Reality Society n.d.).



Kuva 1. Sensorama (Proteus Vr labs LTD n.d.).

Sensoraman lisäksi 60-luvulla konseptoitiin monia muita virtuaalitodellisuuslaitteita. Nämä olivat suurimmaksi osaksi virtuaalilaseja muistuttavia, Head Mounted Display (HMD) -laitteita. Näistä konsepteista ensimmäinen, oli vuonna 1960 kehitetty Telesphere mask, jolla pystyi katsomaan tavallista filmiä ilman liikkeen tarkastelua. Lasit sisälsivät stereoskooppisen 3D-kuvan. Seuraava konsepti oli nimeltään Headsight, nykyisiä virtuaalilaseja muistuttava HMD-laite, joka kehitettiin vuonna 1961. Tämä laite sai kuvansa ulkoisesta kamerasta ja se olisi mahdollistanut esimerkiksi armeijan käytössä vaarallisten paikkojen turvallisen tutkailun dynaamisesti päätä liikuttelemalla. (Virtual Reality Society n.d.)

Vuonna 1965 Ivan Sutherland loi 'Ultimate Display' -konseptin. Tämän konseptin tavoitteena oli tuottaa lähes täydellinen immersio, ja sen tärkeimpiin spekseihin kuului virtuaalisen maailman tuottaminen tietokoneella, sen ylläpitäminen reaaliajassa, käyttäjän realistinen vuorovaikutus ympäristön kanssa ja tätä vastaava 3D-ääni. Tämän pohjalta Sutherland kehittikin Sword of Damocles HMD-laitteen (kuva 2). Se oli yhteydessä tietokoneeseen, jolla kuva tuotettiin. Laite oli kuitenkin niin painava, että se piti kytkeä kattoon kiinni, mikä teki siitä kömpelön käyttää. (Virtual Reality Society n.d.)

Kuva 2. Sword of Damocles (Proteus Vr labs LTD n.d.).

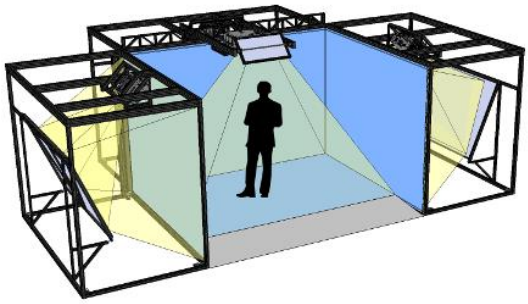
Vuonna 1987 kehitettiin monia virtuaalitodellisuuslaitteistoja, kuten Dataglove ja EyePhone HMD -laite. Tällöin alettiin myös puhua virtuaalitodellisuus-termistä. (Virtual Reality Society n.d.)

Virtuaalitodellisuus alkoi näkyä suurelle yleisölle 90-luvulla. Silloin tuotettiin Arcade-pelejä VR-laseille, joissa latenssia oli noin 50 ms, stereoskooppisella 3D-näkymällä. Osa peleistä oli monen pelaajan pelejä. Myös SEGA ilmoitti valmistavansa omat VR-lasinsa, vuonna 1993. Laseissa oli pään sijainnin tarkastelu ja stereoääni. Lasit jäivät kuitenkin prototyypeiksi. Sen sijaan Nintendo sai julkaistua vuonna 1995 Nintendo Virtual Boy -konsolin (kuva 3). Tämä ei kuitenkaan yltänyt myyntimenestykseksi. Syinä saattoivat olla värien vähyys, ohjelmiston tuen vähyys ja käyttömukavuuden puute. Samana vuonna julkaistiin iGlasses (kuva 4). (Virtual Reality Society n.d; Experimental Game Design – Postgaming 2016.)

Kuva 3. Nintendo Virtual Boy (tomodachi.wikia.com n.d.).

Kuva 4. iGlasses-virtuaalitodellisuuslasit (Whelan 2014).

90-luvulla luotiin myös virtuaalilaseista poikkeava CAVE-ympäristö (kuva 5). CAVE-järjestelmässä noin 3-6 seinälle projisoidaan kuvaa, pienessä huoneessa käyttäjän ympärille. Järjestelmässä ei siis tarvitse pitää yllään laseja, mutta käytön tekee hankalaksi projektoreiden ja sopivan huoneen tarve. (Billinghurst 2016.)



Kuva 5. CAVE-ympäristö (Visbox n.d.).

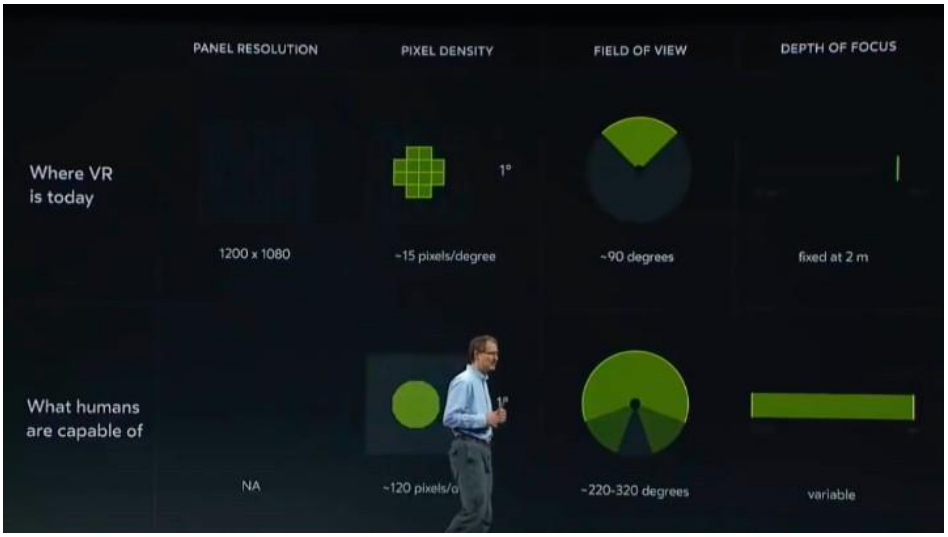
2.1.2 Vaatimukset

Jotta virtuaalitodellisuus olisi mahdollisimman todentuntuinen, eikä aiheuttaisi pahoinvointia, tulisi laitteiston vastata ihmisen näkökyvyn ja kuvien tulkintakyvyn asettamiin vaatimuksiin. Todellisuudessa emme erota yksittäisiä pikseleitä tai koe varsinaista viivettä katsoessamme ympäristöä ympärillämme, mihin virtuaalitodellisuuslasien kehittäjät myös pyrkivät.

Ihmisen näkökyky ei ole kuitenkaan täysin yksiselitteinen, eivätkä kaikki ihmiset näe juuri samalla tavalla. Ihmisen tarkan näön alue on hyvin pieni ja tätä tarkkaa pistettä ympäröiviä alueita aivot paikkailevat niiden oppimien mallien mukaisesti. Ilmiötä voi tarkastella erilaisten optisten illuusioiden avulla. Näin ollen myös kuvien tulkinta, joka tapahtuu aivoissa, on yksilökohtaista. Todellisuus muodostetaan subjektiivisesti kunkin aivoissa. (Abrash 2015.)

Näiden faktojen takia on vaikea antaa täysin tarkkoja lukuarvoja, joita virtuaalitodellisuuden tulisi noudattaa tai saavuttaa sekä laitteisto että sovellustasolla. Osalle ihmisistä aiheutuu helpommin pahoinvointia, toiset pystyvät viettämään kauemmin aikaa virtuaalitodellisuudessa.

Kuten kuvasta 6 nähdään, virtuaalitodellisuudella voidaan tuottaa tällä hetkellä noin 15 pikseliä per aste, 90:nen asteen katselukulmalla ja fokusointipiste on noin kahdessa metrissä. Yleisesti ottaen ihminen kuitenkin kykenee noin 120 pikseliin per aste, 220:nen asteen katselukulmalla ja fokusointipiste on muuttuvalla asteikolla. (Langley 2016.)



Kuva 6. Abrash pitämässä puhetta ihmisen näkökyvyn ja virtuaalitodellisuuden yhteydestä (Langley 2016).

Nykypäivän virtuaalitodellisuuslaseissa käyttäjä näkee vielä pikselit. Jotta saataisiin samanlainen kokemus ja pikselitiheys kuin katsottaessa tavallista monitoria, pitäisi resoluution olla 5kx5k per silmä (Abrash 2015).

Katsottaessa liikkuvaa näkymää tai liikettä sisältävää näkymää, on ruudunpäivitystaajuuden korkeus todella tärkeä. Ihmisen silmä havaitsee liikkeen erittäin tarkasti erityisesti silmän tarkan näön pisteen ulkopuolella. Jos ruudunpäivitystaajuus ei ole tarpeeksi korkea, voi liike tuntua ”sekavalta” (disorienting), viiveiseltä ja aiheuttaa pahoinvointia. Tarpeeksi korkea ruudunpäivitys tuottaa vahvemman immersion. Yleisesti ottaen minimi ruudunpäivitystaajuudelle nykypäivän virtuaalitodellisuuslaitteilla on 60 Hz. (Wiltshire 2017.)

Sovellusta kehitettäessä pitäisi itse sovelluksen frame rate (kuvataajuus) saada vastaamaan ruudunpäivityksen lukua, esimerkiksi Gear VR -laitteistolle kehitettäessä sovellusta, ruudunpäivitys on 60 Hz joten fps-lukema pitää saada pidettyä 60 fps:ssä (Oculus Documentation n.d.)

Viiveet tekevät virtuaalitodellisuudesta helposti pahoinvointia tuottavan kokemuksen, sillä aivot eivät ole tottuneet kokemaan viiveitä. Enimmillään latenssia saisi olla ideaaliolosuhteissa 20 ms tai vähemmän. Jo 50 ms viiveen huomaa, vaikka sovellus tuntuukin vielä responsiiviselta. (OculusRift Blog n.d.)

2.1.3 Laitteet

Yleisesti ottaen laitteet voidaan jakaa kahteen kategoriaan, langattomiin ja langallisiin virtuaalilaseihin. Langattomien virtuaalilasien etuna on niiden helppo liikuteltavuus, sillä tietokonetta tai pelikonsolia ei tarvitse kantaa mukanaan käyttääkseen sovellusta. Suurin osa langattomista virtuaalilaseista käyttää hyödykseen älypuhelinta (mobiili VR). Haittapuolena on

kuitenkin useimmissa tapauksissa suoritustehon vähyys ja pienempi virkistystaajuus (taulukko 1). Tämä vaikuttaa negatiiviseesti immersioon ja luo rajoitteita mallien ja ympäristöjen monimutkaisuudelle. Myös akunkesto on tärkeä huomiokohde. Gear VR, Google DayDream sekä Google Cardboard ovat esimerkiksi mobiili VR -laitteistoja.

Vaikka pikseliterävyys onkin korkeampi joissain mobiili VR -laitteistoissa verrattuna langallisiin laitteistoihin (taulukko 1), ei tämä suoraan tarkoita parempaa kuvanlaatua. Tietokoneisiin liitettävissä VR-laseissa on liitettynä kyseisen tietokoneen graafinen laskentateho, mikä tekee kuvanlaadusta yleensä paremman. (Shanklin 2016.)

Langallisten lasien etuna on parempi kuvanlaatu, tehokkuus ja lähes poikkeuksetta korkeampi virkistystaajuus (taulukko 1). Haittapuolena on tehokkaan koneen tarve. Vaikka langalliset virtuaalilasit eivät tarvitse erillistä akkua, ne vievät tilaa ja ovat raskaampia liikutella. Esimerkiksi markkinointikäytössä tämä voi olla hankala elementti. Htc Vive, Oculus Rift sekä Sony PS VR ovat langallisia VR -laitteistoja. Vive ja Oculus Rift pyörivät PC-ympäristöissä ja Sony PS VR Playstation 4:lla.

Lisäksi omana virtuaalitodellisuuslaitteistokategoriana voitaisiin pitää langattomia virtuaalilaseja, jotka pyörivät oman käyttöjärjestelmänsä ympärillä mobiililaitteen sijaan. Tällaisia laitteita ovat esimerkiksi Microsoft Hololens ja tulevaisuuden Oculus Rift (Prasuethsut 2016). Hololens on enemmän keskittynyt AR-laitteistoksi, eikä sitä ole vielä varsinaisesti julkaistu kuluttajille.

Taulukko 1. VR laitteistojen vertailutaulukko (Shanklin 2016).

LAITE	PIKSELIÄ PER SILMÄ, TYYPPI	RUUDUNPÄIVITYSTAAJUUS	NÄKÖKENTTÄ ASTEINA	RAUTAVAATIMUKSET	LANGALLISUUS	SIJAINNIN TARKASTELU
HTC VIVE	1080x1200 OLED	90 Hz	110	PC	HDMI + USB + virta + 2 virtaa (majakoille)	Kyllä
OCULUS RIFT	1080x1200 OLED	90 Hz	110 <	PC	HDMI + 2/3 USB	Kyllä
SONY PS VR	960x1080 RGB OLED	90 -120 Hz	100	PS4	HDMI + USB	Kyllä
GEAR VR	1280x1440 AMOLED	60 Hz	96 - 101	Samsung Galaxy Phone 6<	Langaton	Ei
DAYDREAM	1280x1440 - 960x1080 AMOLED	60 Hz		Pixel/Pixel XL Phone	Langaton	Ei

2.1.4 Nykyhetken käyttökohteet ja tulevaisuus

Virtuaalitodellisuutta käytetään tunnetuimmin peleissä. Virtuaalitodellisuus on kuitenkin käytössä myös monissa muissa kohteissa.

Armeijan sovelluksia ovat mm. lento, taistelutilanne ja ajoneuvo -simulaatiot (kuva 7). Virtuaalitodellisuutta käytetään myös lääkintäjoukkojen opettamiseen. Sovelluksia löytyy myös post-traumaattisesta stressihäiriöstä (PTSD) kärsivien sotilaiden ja veteraanien hoitoon. (Virtual Reality Society n.d.)

Kuva 7. Laskuvarjohyppy-simulaatio virtuaalitodellisuudessa (Virtual Reality Society n.d.).

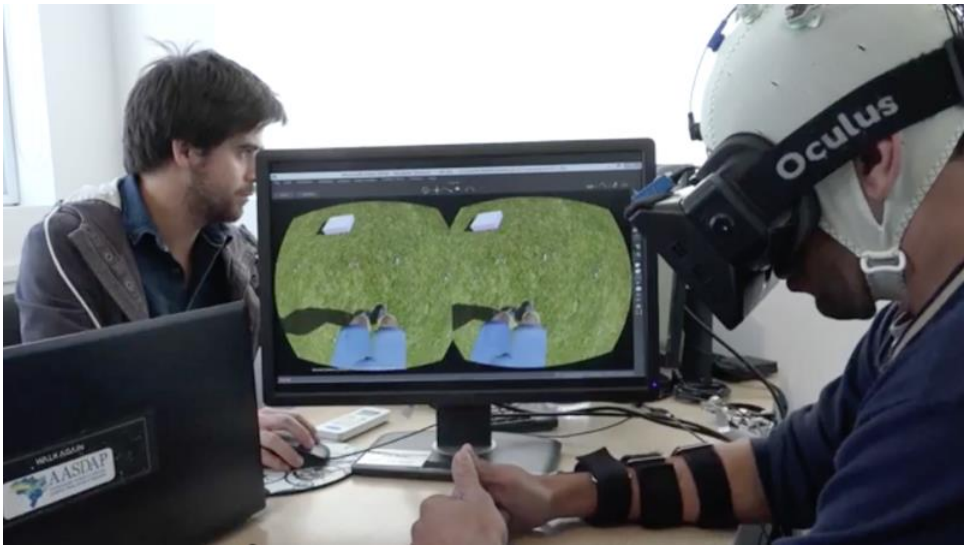
Virtuaalilaseja käytetään myös opetuksessa ja tämä onkin yksi tulevaisuuden käyttökohde. Virtuaalitodellisuus palvelee monia erilaisia oppimistyylejä, sillä se antaa mahdollisuudet sekä visuaaliseen, auditiiviseen että kineettiseen oppimiseen. Opiskelija voi sekä nähdä että kuulla opetuskohteesta, mutta myös kokeilla opittua turvallisessa ympäristössä. Se antaa mahdollisuuden kokemusoppimiseen aivan uudella tavalla, mikä voisi olla esimerkiksi sairaanhoitopuolella todella hyödyllistä.

Sairaanhoitoalalla vaarallisia hoitotilanteita voitaisiin harjoitella turvallisesti, ilman vaaraa oikean potilaan vahingoittamisesta. Harjoitustilannetta voisi tarvittaessa toistaa useita kertoja ja omista virheistään voisi oppia eettisesti. Koska kalliit simulaatiolaitteet, tilat sekä nuket olisivat mallinnetut virtuaalitodellisuuteen, ei näitäkään tarvitsisi erikseen hankkia, pelkästään virtuaalitodellisuuslaitteisto ja tähän simulaatio-ohjelmat. Ennen monen käyttäjän yhtäaikaisia järjestelmiä huonona puolena olisi ryhmäoppimisen puuttuminen, vaikkakin toisen oppilaan suorittamaa harjoitusta voitaisiin seurata joko yhteisesti luokan kesken tai yksityisesti pelkästään esimerkiksi opettajalle näkyvästä ruudusta.

Virtuaalitodellisuutta on kokeiltu ja käytetty myös terveydenhuoltoalan sovellutuksena. Sitä on esimerkiksi käytetty leikkauspotilailla rauhoittavan/nukuttavan korvikkeena (kuva 8). Sen on huomattu vähentävän kipua ja stressiä operaation aikana. Tässä on pidettävä kuitenkin huolta, että katseltava virtuaalitodellisuussovellus tai -video on luonteeltaan rauhoittavaa ja sykettä alentavaa. (Marchant 2017.)

Kuva 8. Virtuaalitodellisuuslasit leikkauspotilaan käytössä (Neal 2014).

Virtuaalitodellisuutta on käytetty halvautuneiden potilaiden kuntouttamiseen (kuva 9). Ulkoisen mekaanisen luurangon ja virtuaalitodellisuuden kanssa ollaan saatu palautettua mm. motorisia kykyjä selkäydinvauriosta kärsivillä potilailla. (Dvorsky 2016.)



Kuva 9. Virtuaalitodellisuuslasit kuntoutuskäytössä (Dvorsky 2016).

On epävarmaa, tuleeko virtuaalitodellisuus lyömään itseään läpi kuluttajamarkkinoilla lähivuosina. Pelaajia, kuten myös muita käyttäjäryhmiä, markkinoilta karkottaa laitteistojen hinta sekä peli- ja sovellushittien vähyys. Ei-pelaajat joutuvat tyytymään useimmiten langattomiin ratkaisuihin, sillä suurin osa langallisista virtuaalilaseista vaatii todella tehokkaan koneen.

Sen sijaan yrityksissä virtuaalitodellisuuden markkinointikäyttö voisi olla suurikin ilmiö. Valtaviakin suunnittelutuotteita, kuten talojen arkkitehtuuria tai suuria mekaanisia osia, pystyttään esittelemään esimerkiksi asiakastapaamisissa tai messutapahtumissa. Tuotteita ei tarvitse näin ollen tuoda paikalle tai näiden ei tarvitse vielä edes olla valmiita ja silti toimintaa pystytään esittelemään monipuolisesti. Myös erilaiset asuntojen, lomakohteiden ja hotellien 360-esittelyvideot voivat edistää markkinointia.

Muita kohteita ovat esimerkiksi virtuaalimuseot ja näyttelyt, elokuvat ja teatteriesitykset, teemapuistot sekä historiallisten kohteiden esittelyt. (Virtual Reality Society n.d.)

2.2 **Pelimoottorit**

Pelimoottorilla tuotetaan ensisijaisesti pelejä, mutta pelimoottoreita voidaan käyttää myös toisenlaisten sovellusten, kuten simulaatioiden tai animaatioiden tuottamiseen. Suurimpaan osaan pelimoottoreista tuotetaan joko ulkoisesti tai moottorin sisällä 2D- ja/tai 3D-grafiikkaa. Skripteillä ohjataan pelin tai sovelluksen toimintoja. Moottoriin on useimmiten tehty työkaluja tai koodiluokkia, jotka tekevät kehityksestä helpompaa. Pelimoottorien kautta peli pystytään myös kokoamaan (build/export) tietyille, pelimoottorin tukemille alustoille. Näin pystytään tuottamaan sama peli esimerkiksi sekä PC- että mobiilipelaajalle, tai vaikka virtuaalilasien käyttäjille.

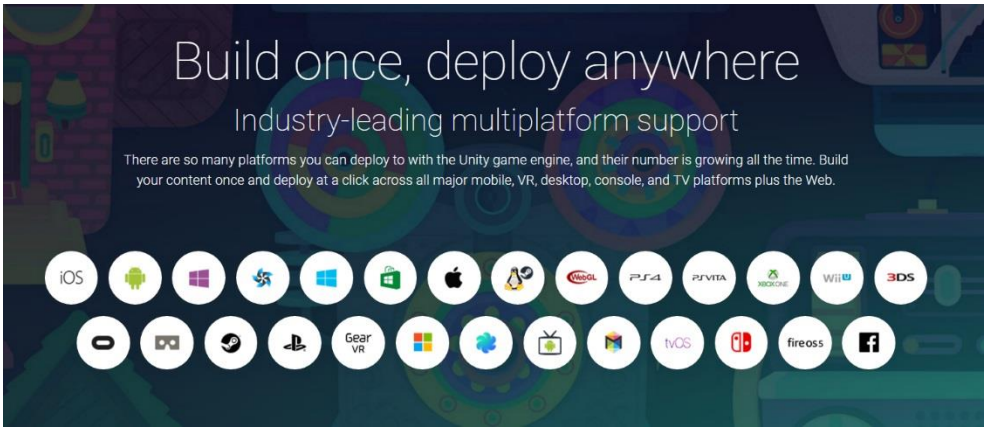
Yleisesti ottaen pelimoottorin käyttäminen virtuaalitodellisuuden sisällöntuotannon alustana mahdollistaa monimutkaisempien vuorovaikutusten tekemisen maailman kanssa. 360-videota katsottaessa on mahdollista päätä kääntämällä nähdä ympäröivä maailma, mutta pelimoottorilla tuotettuun maailmaan voidaan yhdistää esimerkiksi liikkumista tai muita monimutkaisempia vuorovaikutuksia, kuten esineiden siirtelyn paikasta toiseen.

Kaksi yleisintä pelimoottoria joita käytetään virtuaalitodellisuuden luomiseen ovat Unity ja Unreal Engine. Lisäksi virtuaalitodellisuutta tukevia moottoreita ovat Crytek CryEngine, Autodesk Stingray, MaxPlay ja Amazon Lumberyard. (Oculus Documentation n.d.)

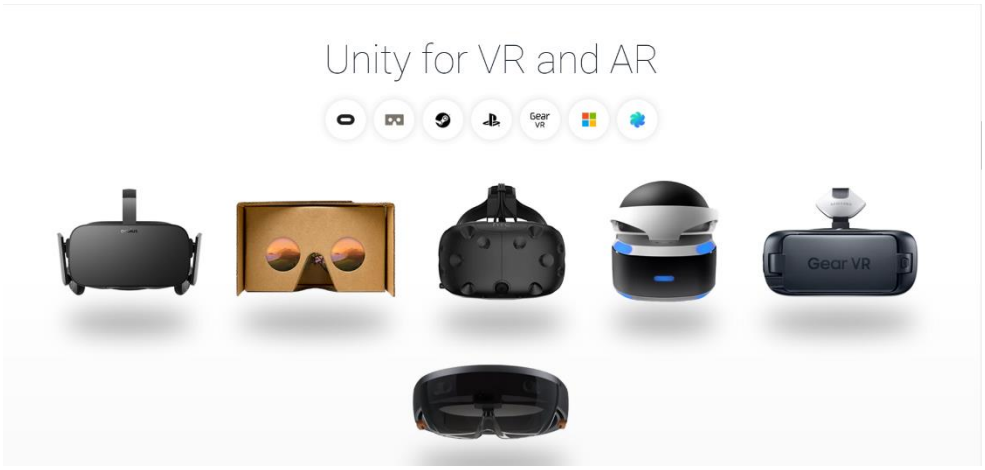
2.2.1 Lyhyet esittelyt moottoreista

Unity on Unity Technologiesin kehittämä pelimoottori, jolla voidaan tehdä sekä 2D- että 3D-pelejä ja sovelluksia. Unity on skriptipohjainen moottori, johon on mahdollista tuottaa toiminnollisuuksia C#- tai JavaScript-kielillä (taulukko 3). Unity on monialustainen moottori ja se tukeekin laajasti erilaisia alustoja, mm. web, mobiili, PC, VR, konsoli sekä TV -sovellutuksia (kuva 10; kuva 11). Unityn lähdekoodi ei ole vapaata (taulukko 3). Tämä voi olla joissain projekteissa rajoittava tekijä. (Unity Technologies n.d.)

Unityn kilpailuvaltteina ovat suuri käyttäjäkunta - erityisesti mobiilipuolella Unity on erittäin suosittu - Asset Store, monialustaisuus sekä helppokäyttöisyys. Koska käyttäjäkunta on niin suuri, löytyy usein ongelmiin äkkiä vastaus Unityn foorumeilta. Myös sisältöä löytyy monipuolisesti Asset Storesta.



Kuva 10. Unityn alustatuki (Unity Technologies n.d.).



Kuva 11. Unityn tuki virtuaalilaitteistoille (Unity Technologies n.d.).

Unreal Engine on Epic Games:in kehittämä moottori. Kuten Unityllä, myös Unreal Engine 4:lla pystyy tekemään 2D- tai 3D-pelejä. UE4 on kuitenkin sisäänrakennettuna enemmän mahdollisuuksia 3D-mallien kompleksim-

paan editointiin, siinä missä Unityä käyttäessä näihin operaatioihin tarvitaan ulkoinen 3D-mallinnusohjelma. UE4 käyttää skriptauskielenään C++ ja Blueprint-työkalua (taulukko 3), jolla pelitapahtumia voidaan tuottaa ilman ohjelmointitaitoa. UE4 ei tue yhtä montaa alustaa kuin Unity (kuva 12; taulukko 2). Unrealin lähdekoodi on avointa, joten moottoria voi laajentaa haluamakseen (taulukko 3). (Epic Games n.d.)

Kuva 12. Unrealin alustatuki (Epic Games n.d.).

Cryengine on Crytekin kehittämä moottori. Toiminnallisuudet tuotetaan siihen C++ ohjelmointikielellä, kuten Unreal Enginessä (taulukko 3). Cryengine ei tue yhtä monia alustoja kuin Unity tai Unreal (kuva 13; taulukko 2). (Crytek n.d.).

Kuva 13. CryEnginen tukemat alustat (Crytek n.d.).

Amazon Lumberyard on Cryenginen pohjalta kehitetty moottori. Koodi tuotetaan siihen C++ ohjelmointikielellä ja Flow Graph -visuaalisella skriptaustyökalulla (taulukko 3). Lumberyardin lähdekoodi on vapaa (taulukko 3). (Amazon Web Services n.d.)

Taulukko 2. Vertailua virtuaalitodellisuuden tuen kanssa (Unity Technologies n.d.; Epic Games n.d.; Cryengine Documentation 2016; Amazon Web Services n.d.; Autodesk Stingray Help n.d.; OSVR Developer Portal n.d.).

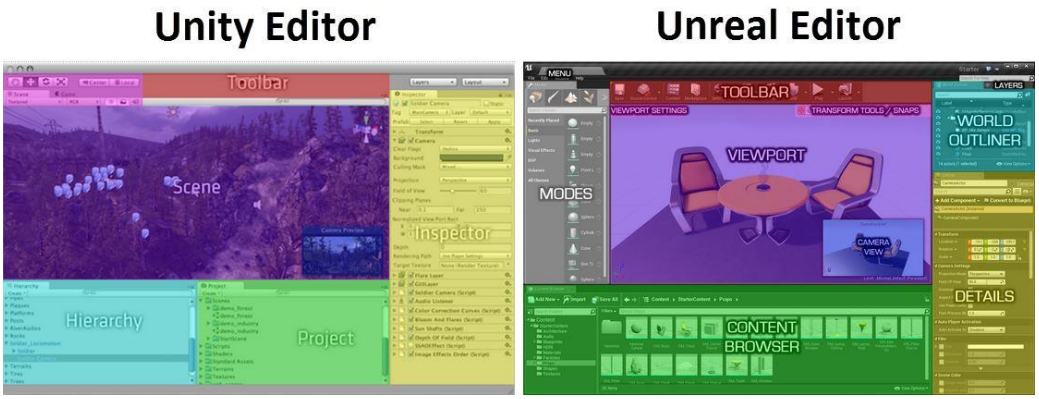
	UNITY	UNREAL	CRYENGINE	LUMBERYARD	STINGRAY
GEAR VR	x	x			x
OCULUS RIFT	x	x	x	x	x
HTC VIVE	x	x	x	x	x
SONY PS VR	x	x		x	
DAYDREAM	x	x			x
CARDBOARD	x	x			x
HOLOLENS	x				
OSVR	Plugin (asset store / github)	Plugin (github)	x		

Taulukko 3. Ohjelmointikielten ja lähdekoodin vertailutaulukko (Unity Technologies n.d.; Epic Games n.d.; Cryengine Documentation 2016; Amazon Web Services n.d.; Autodesk Stingray Help n.d.).

	OHJELMOINTIKIELI	LÄHDE
UNITY	C#, Javascript	Suljettu
UNREAL ENGINE	C++, Blueprint	Vapaa
CRYENGINE	C++	Suljettu (avoinna Githubissa)
AMAZON LUMBERYARD	C++, Flow Graph	Vapaa
AUTODESK STINGRAY	Flow, Lua, (C / C++)	Vapaa

2.2.2 Lyhyt Unity – Unreal -vertailu

Kuten kuvasta 14 nähdään, moottorien käyttöliittymät muistuttavat paljon toisiaan. Samalla värillä varjostetut näkymät vastaavat toiminnollisuuksiltaan toisiaan.



Kuva 14. Käyttöliittymien vertailu (Unreal Engine Documentation n.d.).

Kuvassa 15 on listattu Unityn ja Unrealin sanaston eroavaisuuksia.

Category	Unity	UE4
Gameplay Types	Component	Component
	GameObject	Actor , Pawn
	Prefab	Blueprint Class
Editor UI	Hierarchy Panel	World Outliner
	Inspector	Details Panel
	Project Browser	Content Browser
	Scene View	Viewport
Meshes	Mesh	Static Mesh
	Skinned Mesh	Skeletal Mesh
Materials	Shader	Material , Material Editor
	Material	Material Instance
Effects	Particle Effect	Effect, Particle, Cascade
	Shuriken	Cascade
Game UI	UI	UMG (Unreal Motion Graphics)
Animation	Animation	Skeletal Animation System
	Mecanim	Persona , Animation Blueprint
2D	Sprite Editor	Paper2D
Programming	C#	C++
	Script	Blueprint
Physics	Raycast	Line Trace, Shape Trace
	Rigid Body	Collision, Physics
Runtime Platforms	iOS Player, Web Player	Platforms

Kuva 15. Sanaston eroavaisuuksia (Unreal Engine Documentation n.d.).

Kuvassa 16 on lyhyt koodiesimerkki, jolla lasketaan Count-muuttujaan aina yksi lisää ruutua päivitettäessä. Yleisesti C# mielletään helppolukuisemmaksi kieleksi, kuin C++. Kuvassa 17 näkyy sama esimerkki toteutettuna Blueprint-luokkana.

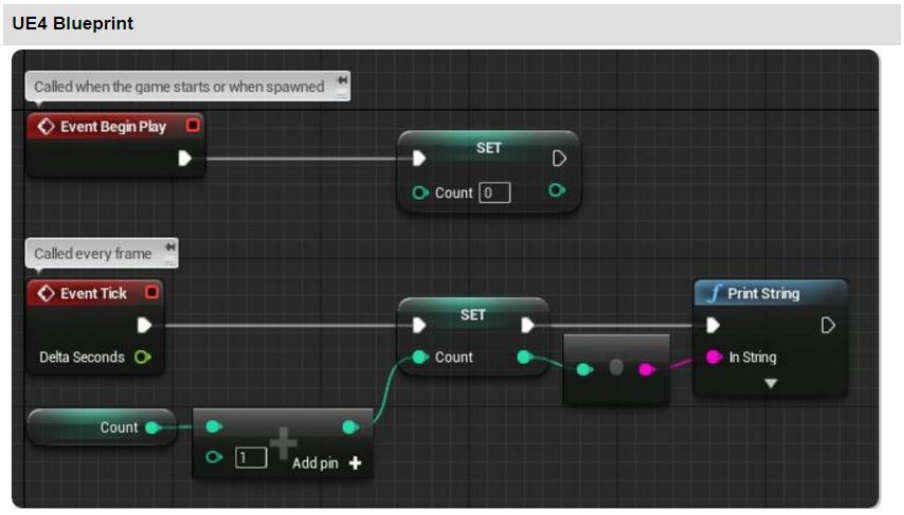
Unity C#

```
using UnityEngine;
using System.Collections;

public class MyComponent : MonoBehaviour
{
    int Count;

    // Use this for initialization.
    void Start ()
    {
        Count = 0;
    }

    // Update is called once per frame.
    void Update ()
    {

        Count = Count + 1;
        Debug.Log(Count);
    }
}
```

UE4 C++

```
#pragma once
#include "GameFramework/Actor.h"
#include "MyActor.generated.h"

UCLASS()
class AMyActor : public AActor
{
    GENERATED_BODY()
    int Count;

    // Sets default values for this actor's properties.
    AMyActor()
    {
        // Allows Tick() to be called
        PrimaryActorTick.bCanEverTick = true;
    }

    // Called when the game starts or when spawned.
    void BeginPlay()
    {
        Super::BeginPlay();
        Count = 0;
    }

    // Called every frame.
    void Tick(float DeltaSeconds)
    {
        Super::Tick(DeltaSeconds);
        Count = Count + 1;
        GLog->Log(FString::FromInt(Count));
    }
};
```

Kuva 16. Lyhyt koodiesimerkki (Unreal Engine Documentation n.d.).



Kuva 17. Blueprint-esimerkki (Unreal Engine Documentation n.d.).

3 SOVELLUKSEN SUUNNITTELU JA AIKATAULUTUS

3.1 Toteutukseen valitut tekniikat

Toteutuksessa päätettiin käyttää moottorina sekä Unityä että Unrealia. Nämä ovat suosituimmat moottorit virtuaalitodellisuuden kehityksessä. Näin saadaan myös tutkittua kummankin moottorin toiminnollisuuksia ja näitä voidaan vertailla paremmin.

Muina valintaperusteluina oli mm. monialustaisuus, kummallakin moottorilla on tuki moniin muihin eri virtuaalilaitteistoihin, mikä tekee uusien sovellusversioiden tekemisen eri laitteistoille todella helpoksi. Tuettuja alustoja on huomattavasti enemmän kuin esimerkiksi verrattuna CryEngineen. Myös renderöintitaso on korkea kummassakin moottorissa ja niissä on sekä ominaisuuksia että tehokkuutta, joita tarvitaan tehtäviin sovellustestauksiin.

Virtuaalitodellisuuslaitteisto päätettiin pitää tässä toteutuksessa Gear VR -laitteistossa sen helpon liikuteltavuuden vuoksi. Vaikkei kuvataajuus ja tehokkuus olekaan yhtä hyvä kuin kaikilla kilpailevilla tuotteilla, sen helppous esimerkiksi messuympäristössä voittaa parempilaatuisen, mutta tietokoneen ja tilan vaativan kokonaisuuden.

3.2 Aikataulu

Opinnäytetyön tulisi olla valmis huhtikuun 2017 loppuun mennessä. Taulukosta 4 nähdään karkea arvio, kuinka paljon aikaa mikäkin osuus työstä tulisi viemään. Jos kaikki menee arvion mukaan, opinnäytetyö tulee valmistumaan arvioituun aikaan. Opinnäytettä kirjoitetaan osittain yhtäaikaisesti työtä tehdessä.

Taulukko 4. Aikataulu.

TEHTÄVÄ	*AIKA*
Liikkuminen ympäristössä	0,5<1 viikko
Mallien tuominen ympäristöön	0,5<1 viikko
Optimoinnin toteuttaminen	1 viikko
Vuorovaikutus ympäristössä	0,5<1 viikko
Animaatiot ja videot	0,5<1 viikko
Opinnäytetyön kirjoittaminen	4-5 viikkoa

3.3 **Lyhyesti tilaajasta**

Työn tilaajana toimii Deemec Oy. Deemec on vuonna 2011 perustettu insinööritoimisto, jonka tärkein palvelu on mekaniikkasuunnittelu. Deemecillä on toimistoja sekä Suomessa Hyvinkäällä että Romaniassa Brasov'issa. Deemec työllistää yli 30 ammattilaista. Muita Deemecin palveluja mekaniikkasuunnittelun lisäksi ovat sähkö- ja automaatiosuunnittelu, virtauslaskenta sekä 3D-visualisointi. 3D-visualisoinnissa toteutetaan sekä kuvarenderöintejä että animaatiovideoita. Uusimpana palveluna on virtuaaliympäristöjen toteuttaminen. (Deemec Oy n.d.)

Deemec Oy:llä ei ole aiempia pelimoottorilla pyöriviä virtuaalitodellisuusympäristöjä. Toiveena olikin tuottaa ympäristöjä, joissa pääsisi liikkumaan vapaasti peliohjaimella. Deemec on aiemmin toteuttanut 3D-kuvia Gear VR -laiteteelle. Kyseiset kuvat on tuotettu 3ds Maxin avulla.

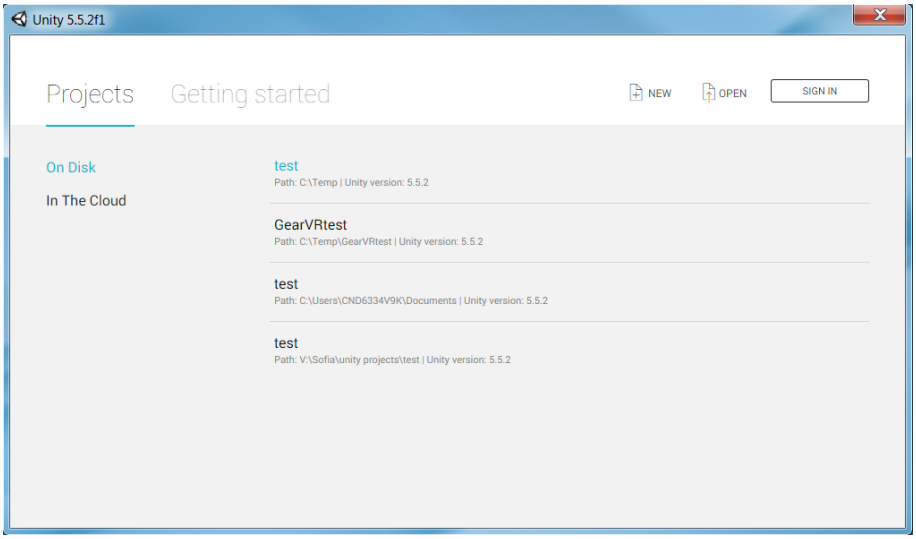
4 GEAR VR -SOVELLUS

4.1 Virtuaalitodellisuusprojektin luominen

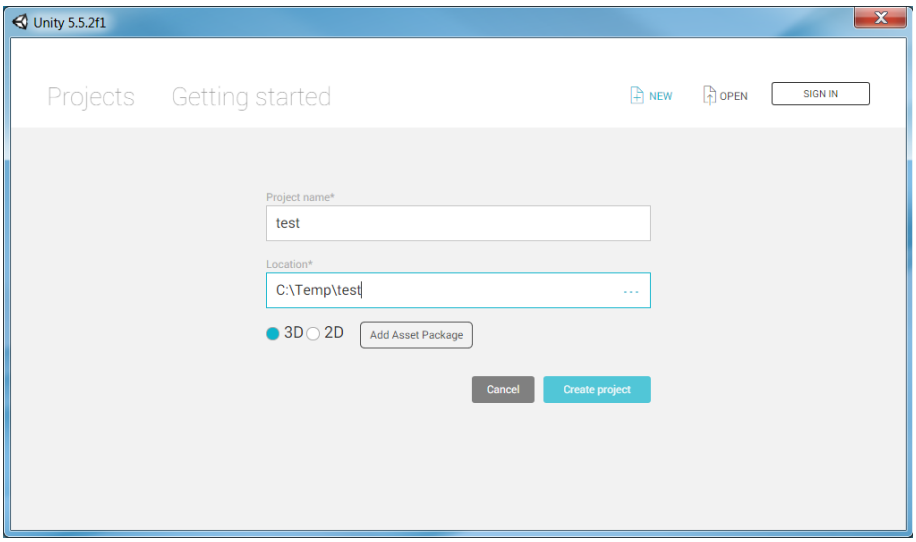
Kappaleessa kerrotaan, miten virtuaalitodellisuuden kehitysympäristö luodaan eri pelimoottoreilla.

4.1.1 Unity

Avataan Unity. Valitaan Projects -ikkunasta NEW. Nimetään projekti kuvan 19 mukaisesti testiksi. Valitaan projektille sopiva sijainti ja ruksitaan 3D-vaihtoehto. Tämän jälkeen luodaan projekti Create Project -nappia painamalla. Myöhemmin avattaessa Unityn luodun projektin saa auki Projects -ikkunasta valitsemalla halutun projektin (kuva 18).

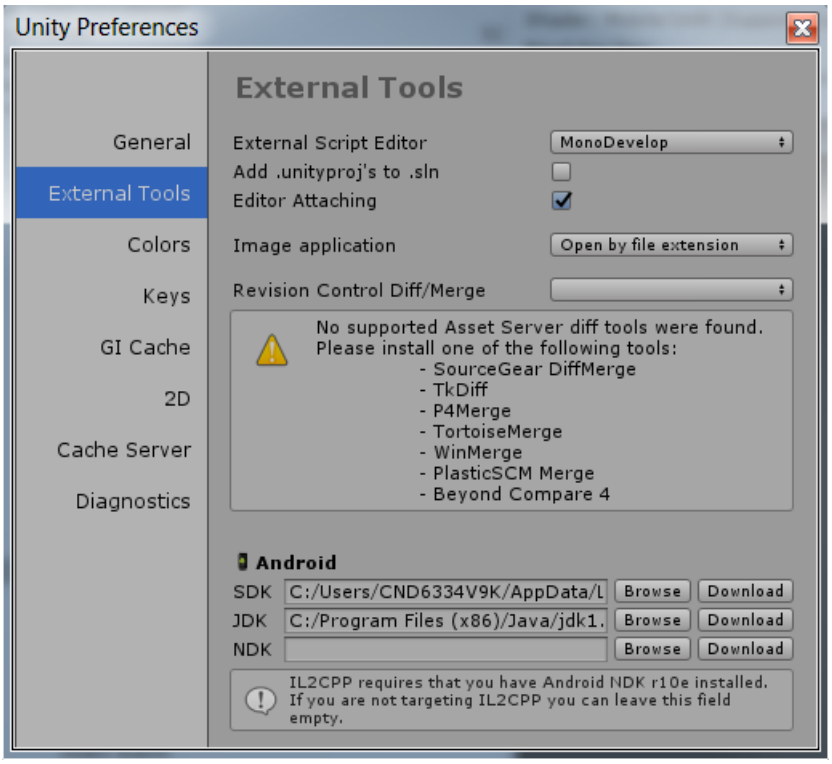


Kuva 18. Projects-ikkuna.



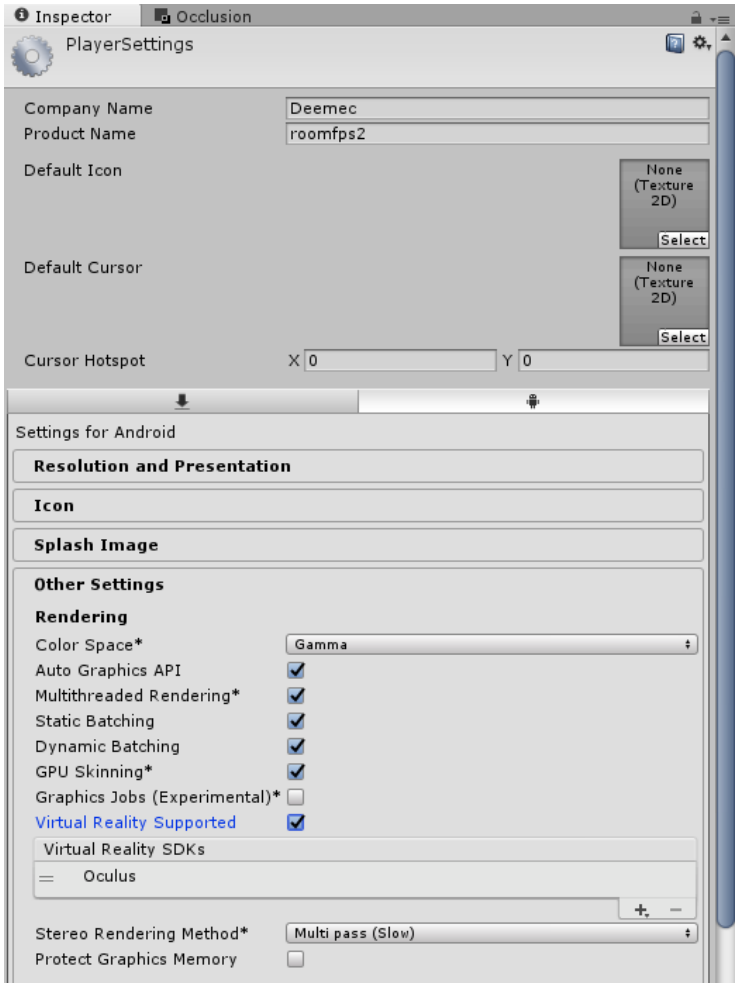
Kuva 19. Projektin luominen.

Jotta sovelluksia voitaisiin tuottaa android-puhelimelle, tarvitaan Unityn tapauksessa android SDK ja Java JDK. Näiden polut lisätään Edit > Preferences > External Tools -ikkunan kautta, kuten kuvassa 20 nähdään.



Kuva 20. External Tools -ikkuna ja lisätyt SDK- ja JDK -polut.

Seuraavaksi avataan PlayerSettings-asetukset Edit > Project Settings > Player -kautta. Avataan Other Settings -sivu ja valitaan Virtual Reality Supported -optio käyttöön, kuvan 21 mukaisesti.

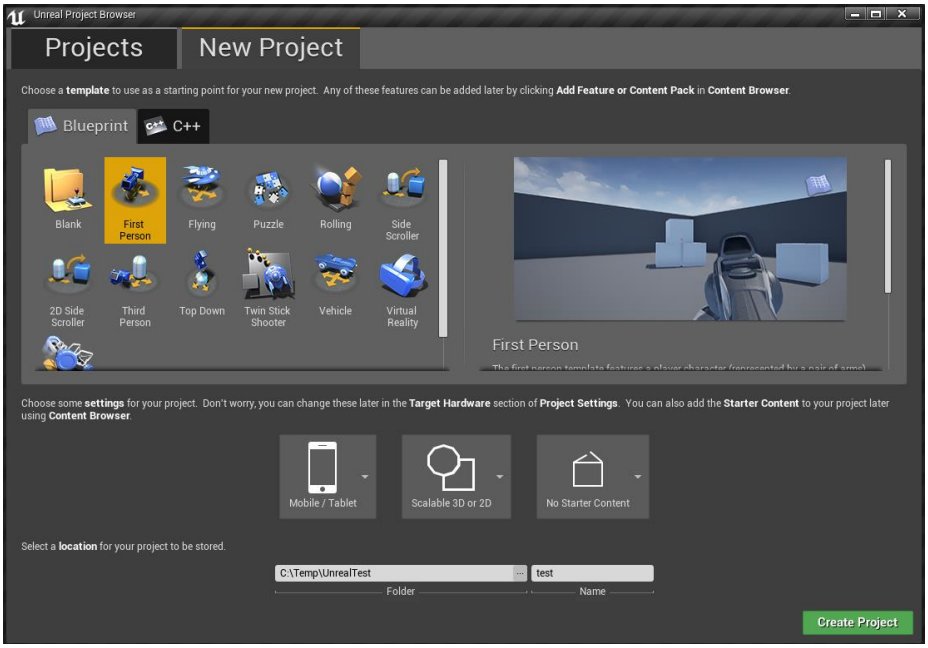


Kuva 21. Player Settings.

Lopuksi tehdään Assets-kansion alle kansiot Plugins > Android > assets. Sinne lisätään oculus signature -tiedosto, jonka saa generoitua oculuksen sivuilta android-puhelimelle. Player-asetuksissa projektille annetaan sopiva nimi, Android Bundle Identifier -kenttään syötetään sopiva nimitieto ja minimum API level asetetaan esimerkiksi 21:teen.

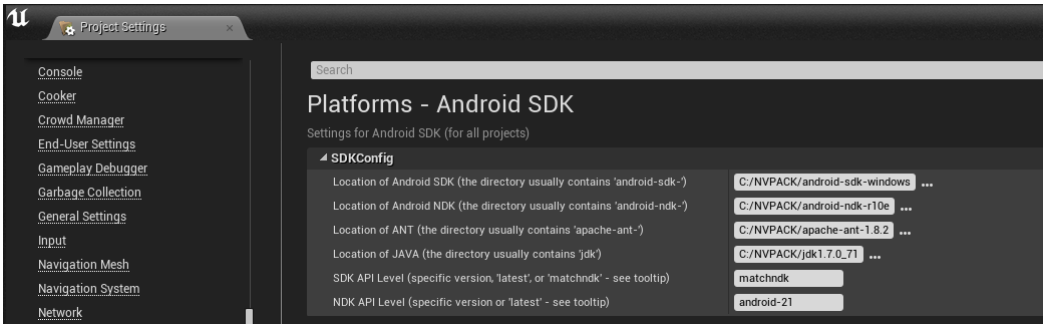
4.1.2 Unreal

Avataan Unreal Engine Epic Games Launcher:in kautta. Tämän jälkeen luodaan uusi projekti kuvan 22 mukaisesti.



Kuva 22. Unreal-projektin luominen.

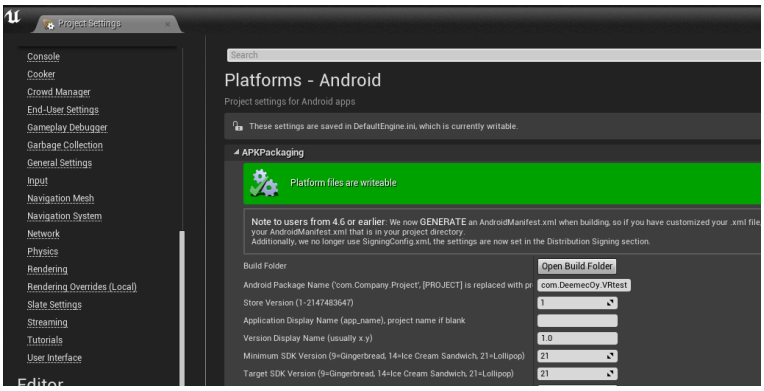
Jotta projektit voitaisiin koota androidille, Unrealin tapauksessa on helpointa käyttää CodeWorksforAndroid-asennustyökalua, niin kaikki tarvittavat paketit saa asennettua kerralla. Kuten kuvassa 23 näkyy, lisätään CodeWorksin asennussijaintiin lisätyt android sdk, android ndk, apache-ant ja Java jdk -pakettien polut Unreal asetuksiin. Asetukset löytyivät edit > Project Settings -sijainnista.



Kuva 23. Android SDK -sijaintiasetukset.

Lisätään seuraavaksi tuki Gear VR:lle Edit > Plugins > Virtual Reality -ikkunan kautta, kuten kuvassa 24 näkyy.

Kuva 24. Gear VR -tuen lisääminen Unreal-projektiin.



Kuva 25. Platforms-asetukset.

Lopuksi lisätään luodun projektin test > Build > Android alle kansio assets ja tähän kansioon lisätään oculus signature -tiedosto. Platforms-asetuksissa lisätään Android Package Name -kenttään sopivat tiedot ja minimum API level asetetaan esimerkiksi 21:teen, kuten kuvassa 25 näkyy.

4.2 **Liikkuminen ympäristössä peliohjaimella**

Kappaleessa kerrotaan, miten liikkuminen toteutetaan eri pelimoottoreilla.

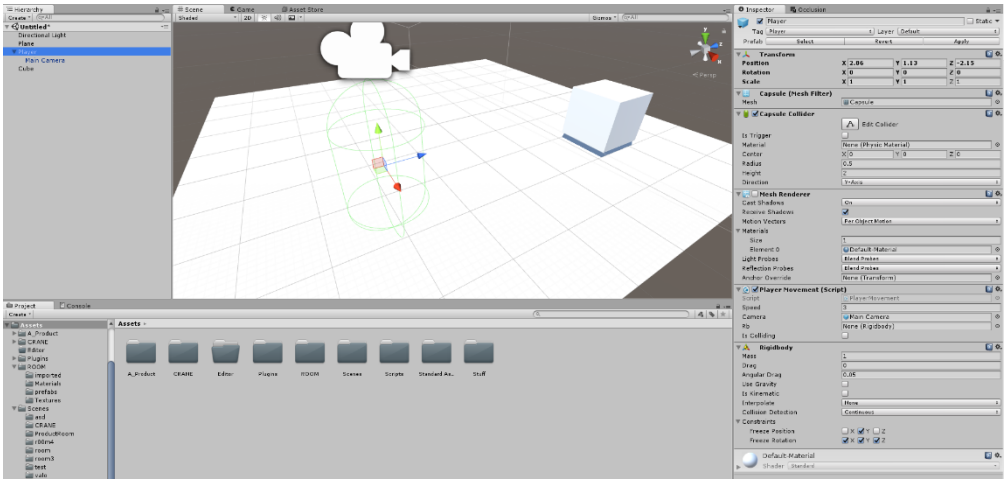
4.2.1 Unity

Luodaan pelaajan kehoksi kapseli. Otetaan heti Mesh Renderer -komponentti pois käytöstä ja nimetään kapseli pelaajaksi (Player). Asetetaan kamera pelaajaobjektin kehon alle lapsiobjektiksi ja säädetään kameran korkeus halutuksi. Annetaan Player-objektille vielä Rigidbody-komponentti. Tämän jälkeen luodaan PlayerMovement-skripti (skripti 1) ja asetetaan se pelaajan kehon komponentiksi. Kuvassa 26 nähdään haluttu Player-malli.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerMovement : MonoBehaviour {

    float horizontal;
    float vertical;
    public float speed;
    public GameObject camera
    public Rigidbody rb;
    public bool isColliding = false;

    void Start() {
        rb = GetComponent<Rigidbody>();
    }

    void Update () {
        horizontal = Input.GetAxis ("Horizontal");
        vertical = Input.GetAxis ("Vertical");

        rotate ();
```

```
        doMove ();

    }

    void rotate(){
        var x = Input.GetAxis ("Horizontal_right") * Time.deltaTime * 30f;
        transform.Rotate (0, x, 0);
    }


    void doMove(){
        Vector3 directionHorizontal = camera.transform.right * Input.GetAxis-
Raw("Horizontal");
        Vector3 directionVertical = camera.transform.forward * Input.GetAxis-
Raw("Vertical");
        Vector3 direction = directionVertical + directionHorizontal;
        direction.y = 0;

        if (isColliding) {
            if(horizontal == 0 && vertical == 0){
                rb.velocity = Vector3.zero;
            } else {
                rb.velocity = direction * speed;
            }

        } else if(!isColliding) {
            rb.velocity = Vector3.zero;
            if (horizontal == 0 && vertical == 0) {
                //DO NOTHING
            } else {
                transform.Translate(direction * Time.del-
taTime * speed, Space.World);
            }

        }

    }

    void OnCollisionStay(Collision col) {
        if (col.gameObject.tag == "col_Obj") {
            isColliding = true;
        }
    }

    void OnCollisionExit(Collision col) {
        if (col.gameObject.tag == "col_Obj") {
            isColliding = false;
        }
    }

}
```

Skripti 1. Player Movement -Skripti.

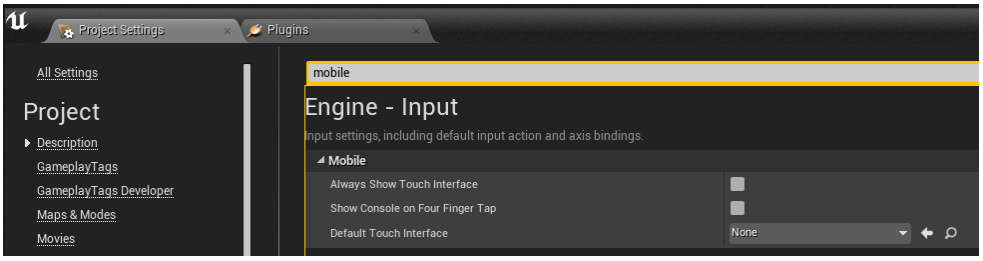


Kuva 26. Player-objektin komponentit.

Käydään tarkistamassa Unityn Input Managerista, että Ohjaimen mappaukset vastaavat koodissa käytettyjä mappauksia. Nämä löytyvät kohteesta Edit > Project Settings > Input. Törmättäville kohteille annetaan tagtiedoksi col_Obj-merkkijono.

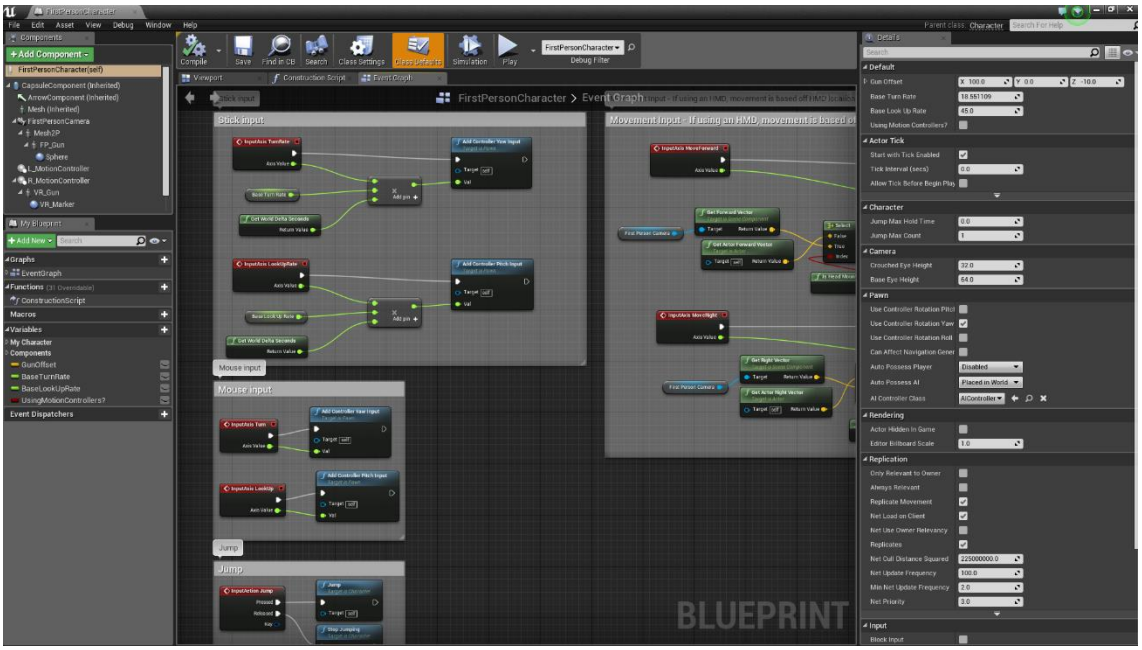
4.2.2 Unreal

Koska projektipohjaksi valittiin First Person Shooter -pelipohja, ei Unrealissa tarvitse lähteä skriptaamaan omaa liikkumiskoodia. First Person Blueprint -ympäristössä poistetaan pelaajan ase käytöstä. Otetaan myös mobiilipeleissä käytetyt virtuaaliset peliohjaimet pois käytöstä asetuksien kautta (kuva 27). Unrealissa liikkuminen ja pyöriminen ovat valmiiksi mapattu oikein peliohjaimelle, eikä näitä asetuksia tarvitse muuttaa.



Kuva 27. Mobiili-input -asetukset.

Liikkumisen ja pyörimisen nopeutta säädetään vähän pienemmäksi (kuva 28), sillä virtuaalitodellisuudessa liian nopea liikkumistahti aiheuttaa pahaa oloa pelaajalle. Myös kameraa nostetaan vähän korkeammalle, jottei pelaaja näytä ryömivän lattiatasossa (kuva 28).



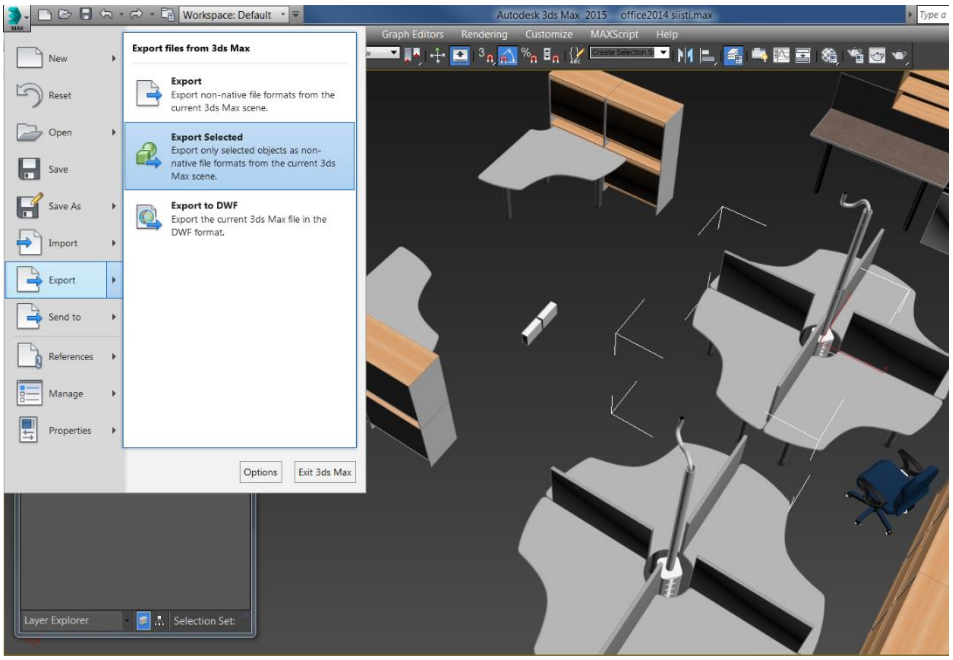
Kuva 28. First Person Character -Blueprint muokkausnäkymä.

4.3 3D-mallien tuominen projektiin FBX-formaatissa 3ds Maxista

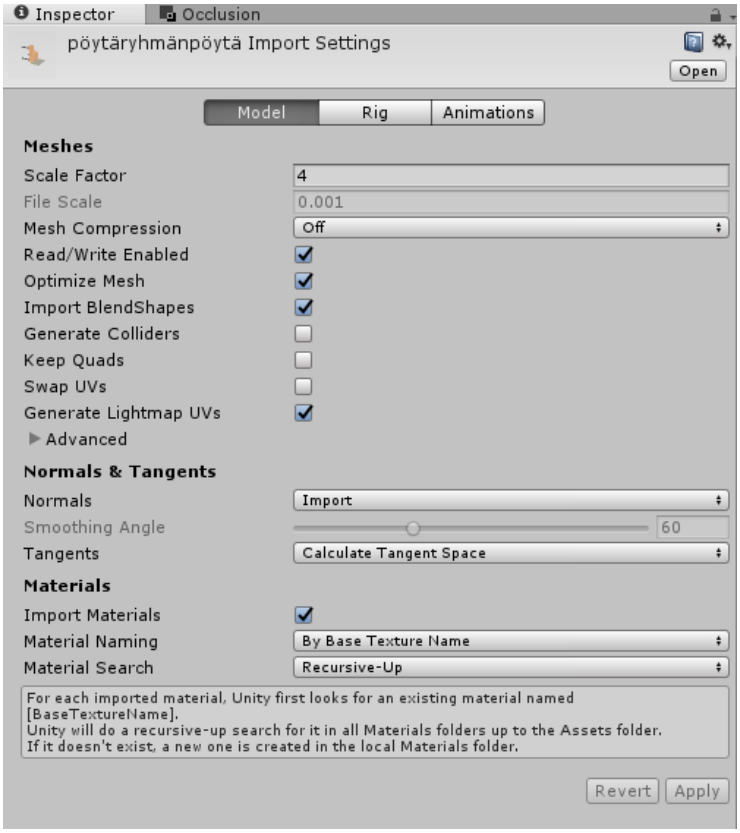
Kappaleessa kerrotaan, miten 3D-malleja tuodaan eri pelimoottoreihin.

4.3.1 Mallien tuominen Unity-projektiin

Halutut kohteet valitaan 3ds Maxissa. Tämän jälkeen otetaan valikosta Export > Export selected, kuten kuvassa 29 nähdään. Halutuksi formaatiksi valitaan esimerkiksi .fbx ja tiedosto nimetään. FBX Export -ikkunassa voi vielä muokata asetuksia, jonka jälkeen valitaan ok. Exportatut mallit tuodaan Unityyn Assests > Imported -kansioon. Tämän jälkeen Unityn käyttöliittymässä objekteja voidaan valita ja näiden asetuksia muuttaa Import-asetuksissa, kuten kuvassa 30 näkyy. Import-asetuksissa valitaan mm. haluttu objektin skaala ja halutut optimointioptiot. Mallien pivot-piste on keskellä kohdetta automaattisesti. Objekti on myös valmiiksi yksi kappale, jota voi myös tarkastella osatasolla.



Kuva 29. Valittu objekti ja Export Selected -valinta.



Kuva 30. Import-asetukset.

4.3.2 Mallien tuominen Unreal-projektiin

Mallien tuominen Unreal-projektiin on hieman monimutkaisempaa, kuin Unityssä. Jos pivot-piste tahdotaan keskelle objektia, on objekti siirrettävä ensin Maxissa keskipisteeseen (0,0,0). Jos objekti myös tahdotaan Unreal-projektiin yhtenä kappaleena, ei osina, on objektin kappaleet (mesh) ensin yhdistettävä (attach) toisiinsa. Näiden operaatioiden jälkeen kohteet valitaan ja exportataan .fbx-muotoon, aivan kuten Unityyn tuodessakin. Unrealissa Import-asetukset avautuvat omaan ikkunaan. Esimerkiksi skaala ja muut optiot valitaan tässä. Jälkeenpäin esimerkiksi objektin skaalausasetuksia pääsee muuttamaan tuplaklikkaamalla objektia Resurssienhallinnassa.

Jos pivot-piste unohdettiin Maxissa siirtää, voidaan se tehdä Unreal-projektissa. Objekti tuodaan editorin ikkunaan ja tämän jälkeen klikataan objektin päällä hiiren oikealla näppäimellä, valitaan Pivot > set object pivot here, jolloin piste siirtyy ainakin lähemmäs. Pistettä voidaan siirtää myös hiiren keskinäppäimellä klikkaamalla ja siirtämällä haluttuun paikkaan.

Objektit voidaan myös yhdistää palasista yhdeksi kappaleeksi Merge Actors -työkalulla. Työkalu löytyy Window > Developer Tools > Merge Actors -valikoista. Halutut palaset tuodaan editoriin ja nämä valitaan. Tämän jäl-

keen valitaan työkaluikkunasta Merge Actors, jolloin palaset yhdistyvät toisiinsa. Näin saadaan aikaiseksi kokonainen objekti, jota on tietyissä tapauksissa helpompi siirrellä, kuin valtavaa määrää pieniä palasia.

4.4 **Optimointi**

Työtä tehdessä tuli yllätyksenä, kuinka paljon sovellusta joudutaan optimoimaan. Tässä kappaleessa luetellaan tapoja, joilla optimoida virtuaalitodellisuussovellusta. Optimointitapoja on hyvin paljon, eikä tässä listassa ole esitelty kaikkia mahdollisia. Kaikkia esiteltyjä tapoja ei käytetty jokaisessa sovellustestauksessa.

4.4.1 Tavoitearvot

Yleisesti ottaen kummallakin moottorilla tuotetun sovelluksen on pyörittävä 60 fps mobiililaitteella, eli jokaisen kuvan laskentaan ja piirtämiseen saa kulua vain noin 16 ms. Taulukossa 5 näkyvät Mobiili ja PC -sovellusten tavoitearvojen erot. Nämä tavoitearvot pätevät sekä Unity- että Unreal -sovelluksille.

Taulukko 5. Tavoitearvojen erot (Oculus Documentation n.d.).

	MOBIILI	**PC**
FPS	60 fps	90 fps
PIIRTOKÄSKYJÄ PER KUVA	50-100	500-1000
TRIS TAI VERTS PER KUVA	50,000-100,000	1-2 miljoonaa

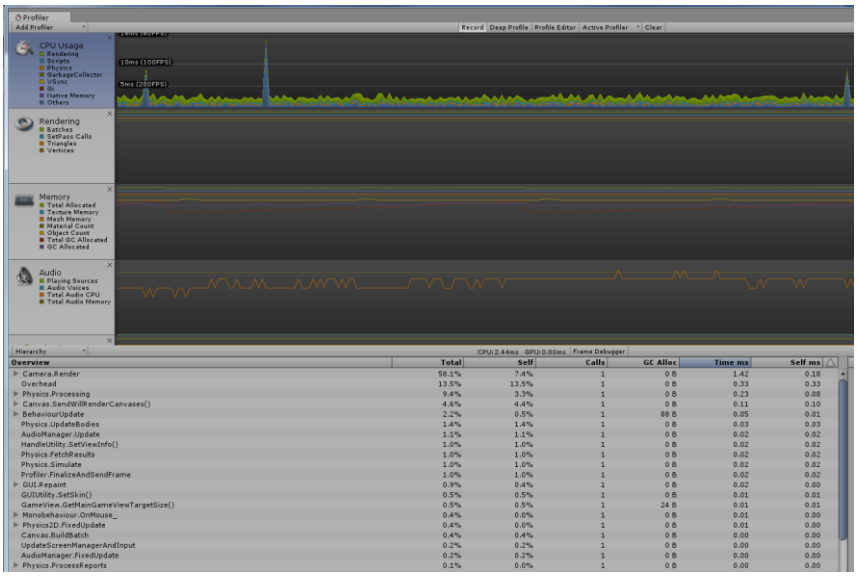
4.4.2 Profiler

Optimointia ei ole koskaan järkevä lähteä tekemään sokkona. Siksi on luotu erilaisia työkaluja, joilla voidaan selvittää, missä kunkin sovelluksen pullonkaula on. Yleisesti ottaen on hyvä selvittää, onko tukos CPU- vai GPU-puolella. CPU vastaa siitä, mitä ja miten piirretään sekä lähettää käskyjä GPU:lle. GPU piirtää kuvan CPU:lta tulleiden ohjeiden mukaisesti. Esimerkiksi CPU-optimointeja ovat piirtokäskyjen vähentämiset ja GPU-optimointi voisi olla shaderien yksinkertaistaminen.

Kuvassa 31 on Unityn esimerkkinäkymä, jota profiloimme. Kyseistä ympäristöä on optimoitu.

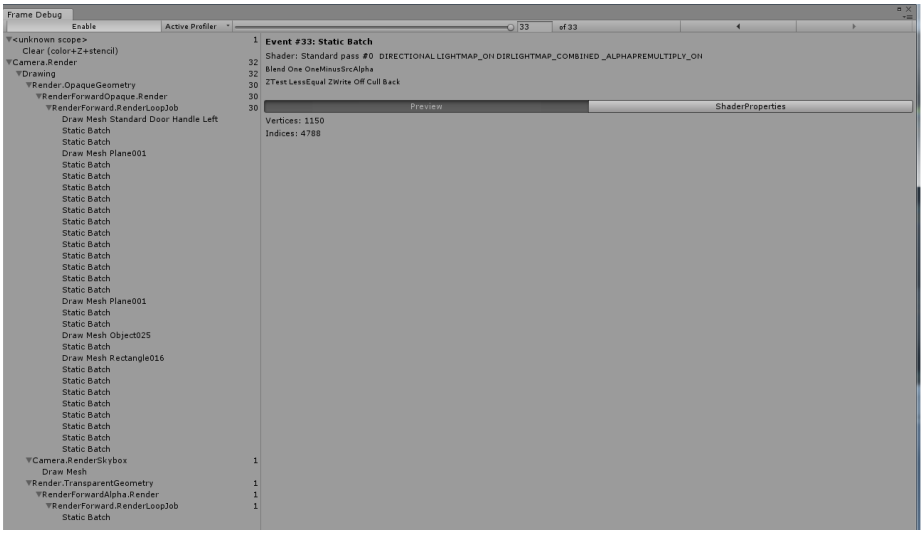
Kuva 31. Unityn esimerkkinäkymä.

Unityn sisäänrakennetun Profiler-työkalun (kuva 32) saa auki Window > Profiler kautta. Sen avulla voidaan tarkastella paljonko mikäkin sovelluksen komponentti vie aikaa, reaaliajassa. Ylhäällä on graafinen näkymä, josta näkee korkeat piikit. Alhaalla on näkymä, josta voi tarkemmin tutkia tiettyjen komponenttien prosentuaalisia sekä ajallisia kulutustasoja. Kyseisessä esimerkissä renderöinti vie eniten aikaa.



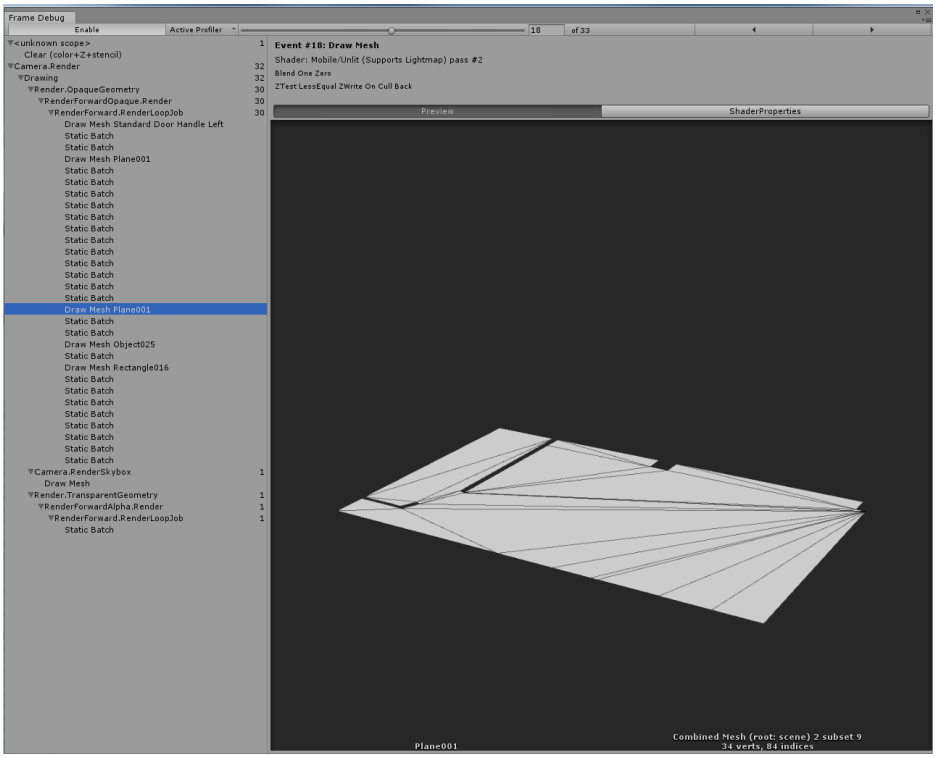
Kuva 32. Unity Profiler -ikkuna.

Unityn toinen työkalu, Frame Debugger (kuva 33) on erittäin hyödyllinen piirtokäskyjen määrän tarkkailemiseen ja selvittämään, miten objektit piirretään ruudulle. Sen saa auki Window > Frame Debugger -polusta. Kuvassa 33 oikealla pulella olevasta listauksesta saa katsottua piirtokäskyjen määrän, joka on tässä tapauksessa 32.



Kuva 33. Frame Debugger -ikkuna.

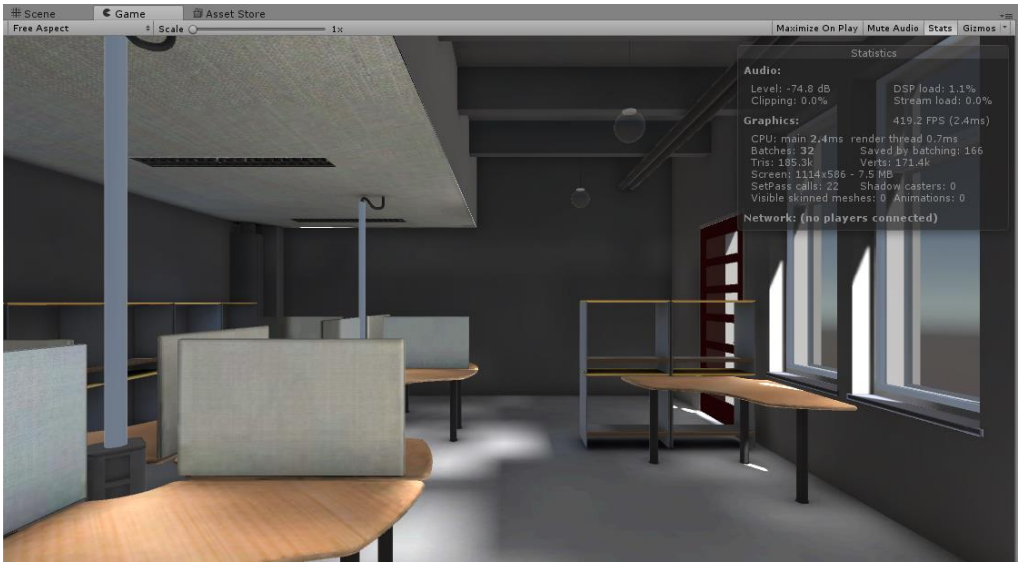
Klikkaamalla listan eri objekteja, saadaan katsottua mitä sillä kerralla piirretään ja miten. Kuvassa 34 on esimerkki piirrettävästä Plane-objektista (lattia). Samalla Game-näkymässä esikatsellaan piirtoprosessia kuvan 35 mukaisesti.



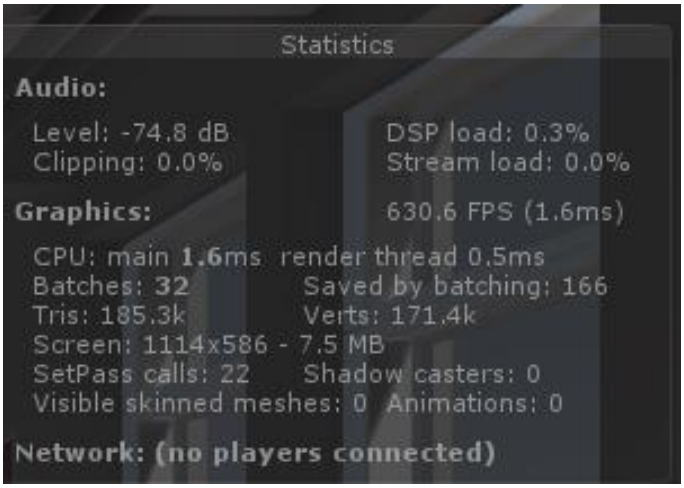
Kuva 34. Lattian piirtämisen esikatselu.

Kuva 35. Lattia on piirretty, mutta esimerkiksi pöytien pinnat ovat vielä piirtämättä.

Unityn Statistics -paneelin saa auki valitsemalla Game-näkymän ja tämän jälkeen klikkaamalla Stats, kuten kuvassa 36 näkyy. Statistics-paneelista näkee helposti tärkeimmät tiedot suoritustehosta suoraan pelinäkymästä (kuva 37). Tärkeitä tietoja ovat piirtojen (batches) määrä, kuinka kauan aikaa CPU ja render thread käyttävät sekä tris- ja verts -lukumäärät.

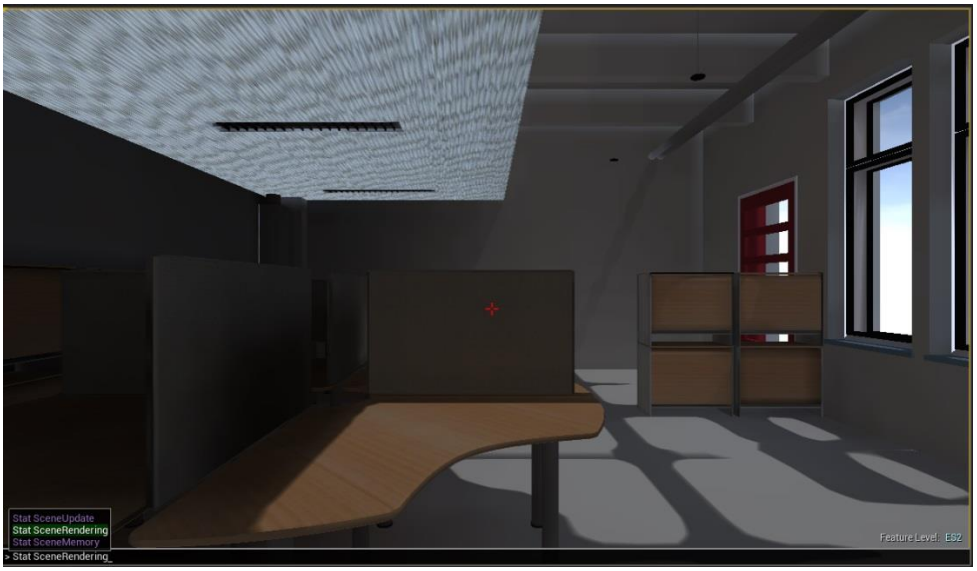


Kuva 36. Statistics-paneelin aukaisu.



Kuva 37. Paneelin tiedot suurennettuna.

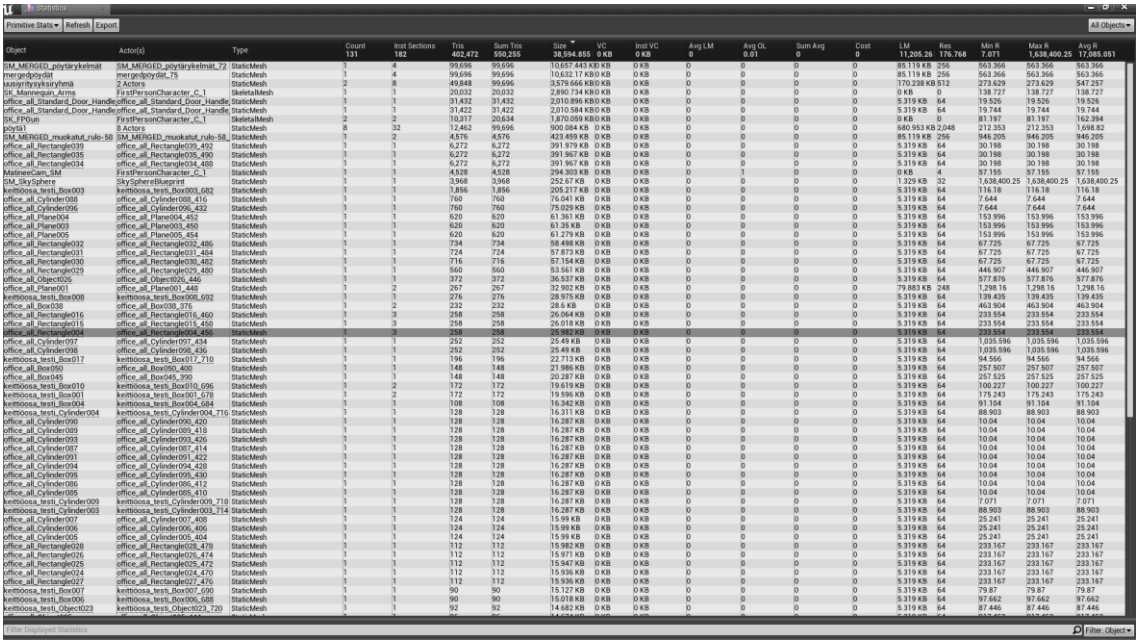
Unrealissa piirtokäskyjen määrää pääsee tarkkailemaan konsolikomennolla stat scenerendering, kuten kuvissa 38 ja 39 näkyy.



Kuva 38. Esimerkkinäkymä ja komennon antaminen konsoliin.

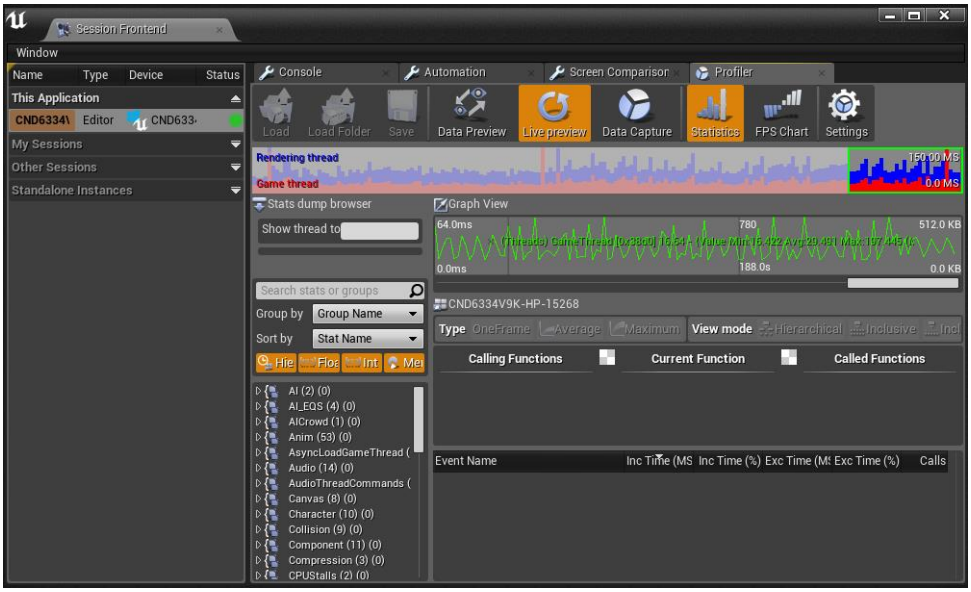
Kuva 39. Scenerendering-tiedot.

Myös Statistics-ikkuna (kuva 40) on hyödyllinen työkalu sen selvittämiseen, mitkä mallit ovat raskaimpia ja paljonko niiden renderöimiseen vaaditaan. Ikkunan saa avattua Window > Statistics -kohteesta.



Kuva 40. Statistics-ikkuna.

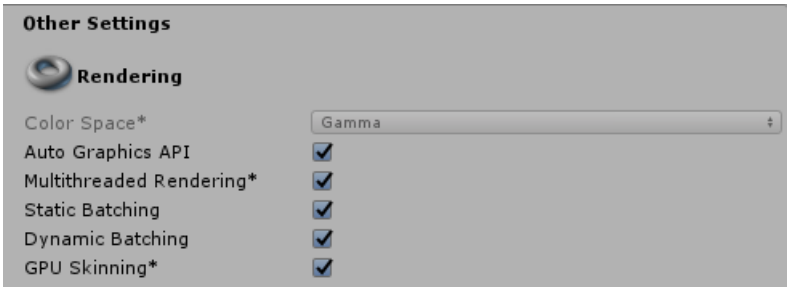
Itse Profiler-ikkunan saa auki kohteesta Window > Developer Tools > Session Frontend > Profiler. Se näyttää kuvan 41 mukaiselta.



Kuva 41. Unrealin sisäänrakennettu Profiler.

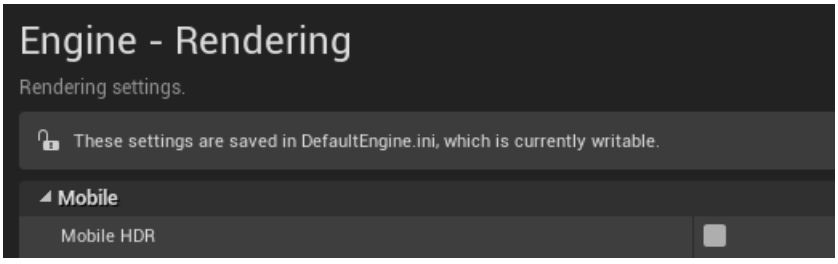
4.4.3 Asetukset

Unity-sovelluksissa Player-asetuksista valitaan käyttöön Static Batching, Dynamic Batching, GPU Skinning ja Multithreaded Rendering, kuvan 42 mukaisesti. Nämä asetukset mahdollistavat piirtokäskyjen vähentämisen ja antavat tarvittavaa lisätehoa. (Pruett 2015.)



Kuva 42. Renderöintiasetukset.

Unreal-projektin asetuksista otetaan pois päältä Mobile HDR, kuten kuvassa 43 näkyy. (Unreal Engine Documentation n.d.).

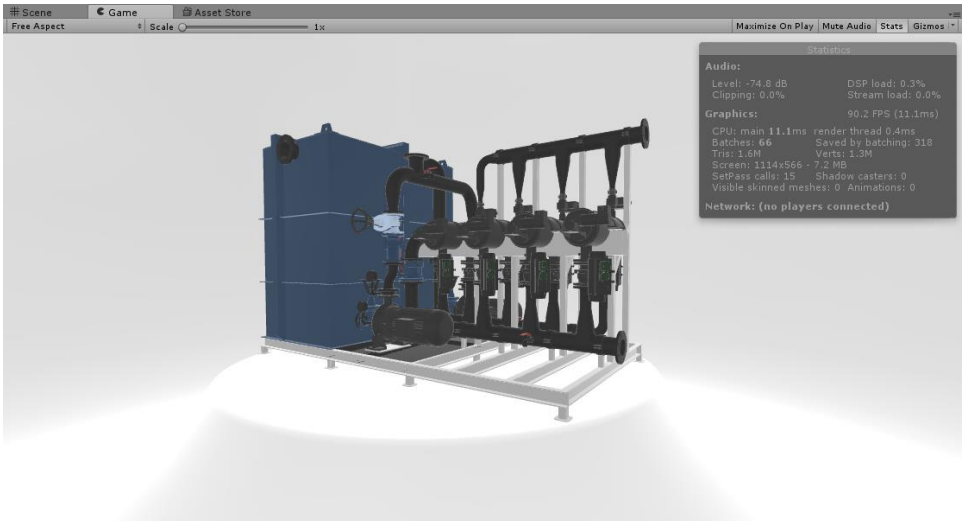


Kuva 43. Mobile HDR -asetus.

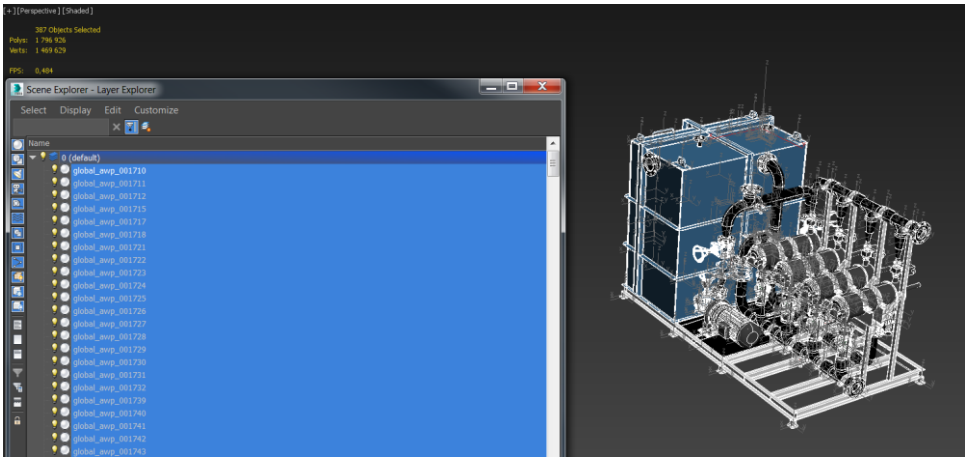
4.4.4 Mallit

Suositeltavaa on, että Gear VR -sovelluksissa käytettäisiin mahdollisimman yksinkertaistettuja tai jopa low-poly -malleja. Tämä tarkoittaa vähennettyä polygonien määrää, mikä vähentää sovelluksessa tris- ja verts -lukumääriä. Myös osien poistaminen malleista voi auttaa, joskin pitää ottaa huomioon, että sovelluksen käyttäjä pääsee mahdollisesti katsomaan malleja monipuolisemmin, kuin tavallisessa sovelluksessa. Jonkin olennaisen osan puuttuminen voi vähentää immersiota. (Unity Technologies n.d.)

Kuvissa 44 ja 45 nähdään malli, jonka polygoni, tris- ja verts -lukumäärä ylittää raja-arvot huomattavasti, jopa ideaaliolosuhteissa noin kuusitoistakertaisesti. Yritettäessä pyörittää tätä mallia Gear VR sovelluksena, on sovellus käyttökelvottoman hidas, vaikka optimointia onkin tehty.

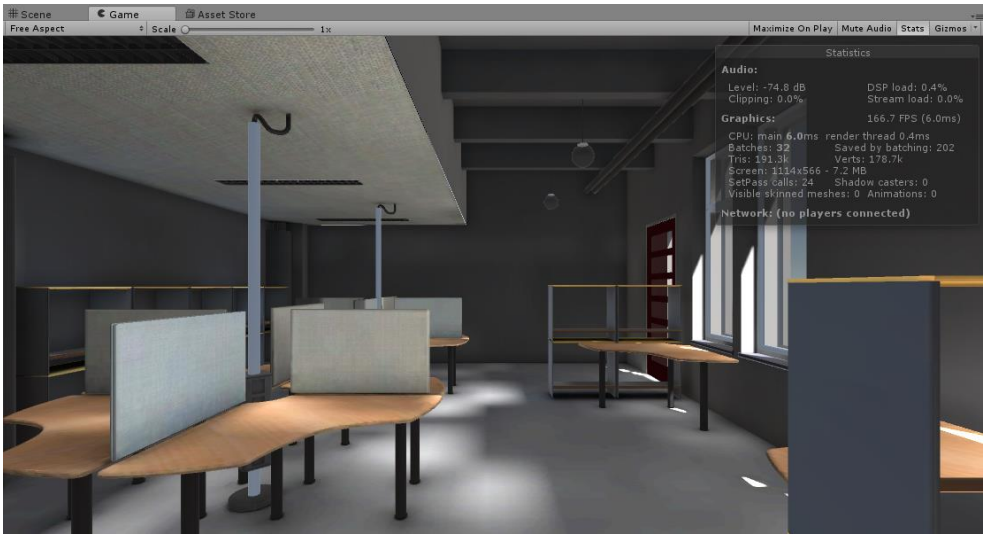


Kuva 44. Monimutkainen malli tuotuna Unityyn.



Kuva 45. Monimutkainen malli 3ds Maxissa.

Sen sijaan kuvan 46 kentässä suositellut tris- ja verts -lukumäärät ylitetään vain noin kaksinkertaisesti, mutta sovellus pyörii silti tasaisesti yli 57 fps muutamaa poikkeusta lukuun ottamatta. Kyseistä sovellusta on optimoitu samoilla tavoilla kuin edellistä sovellusta.



Kuva 46. Huonenäkymä.

Kuvia 47, 48 ja 49 verrattaessa nähdään vielä, kuinka lisätyt objektit vaikuttavat suorituskykyyn. Kuvien 47 ja 48 näkymissä nähdään vain pieni ero fps-lukemassa. Kun näkymään lisätään kuvan 50 mukainen paljon yksityiskohtia sisältävä malli, tipahtaa fps-lukema noin 30-40 kuvaan sekunnissa, halutun 60 fps sijaan.

Kuva 47. Huonenäkymä kuvakulmasta, jossa puhelimen suorituskyky on lähellä kattoarvoja.

Kuva 48. Huonenäkymä, johon on lisätty muutama lisäobjekti.

Kuva 49. Huonenäkymä, johon on lisätty objekti, jossa on paljon yksityiskohtia.

Kuva 50. Suurennos yksityiskohtaisesta mallista.

4.4.5 Materiaalit

Kummassakin pelimoottorissa materiaalien määrä kannattaa pitää mahdollisimman pienenä. Tämä vähentää piirtokäskyjä. Materiaalit kannattaa pitää myös mahdollisimman yksinkertaisina.

Unrealissa kannattaa välttää translucent materiaaleja. Suositeltuja materiaaliarvoja ovat Base color, Roughness, metallic, specular, normal, emissive, opacity, opacity mask. Blend Moodeina kannattaa käyttää vain Opaque, Masked, Translucent, Additive, Modulated -moodeja. (Unreal Engine Documentation n.d.)

4.4.6 Tekstuurit

Tekstuureita kannattaa käyttää vain tarvittaessa. Tekstuurien kirjastoiminen (texture atlasing) vähentää myös erillisten tekstuurien ja materiaalien määrää.

Unreal-projekteissa tekstuurien koon on oltava pienempi tai yhtä suuri kuin 2048 pikseliä (Unreal Engine Documentation n.d.).

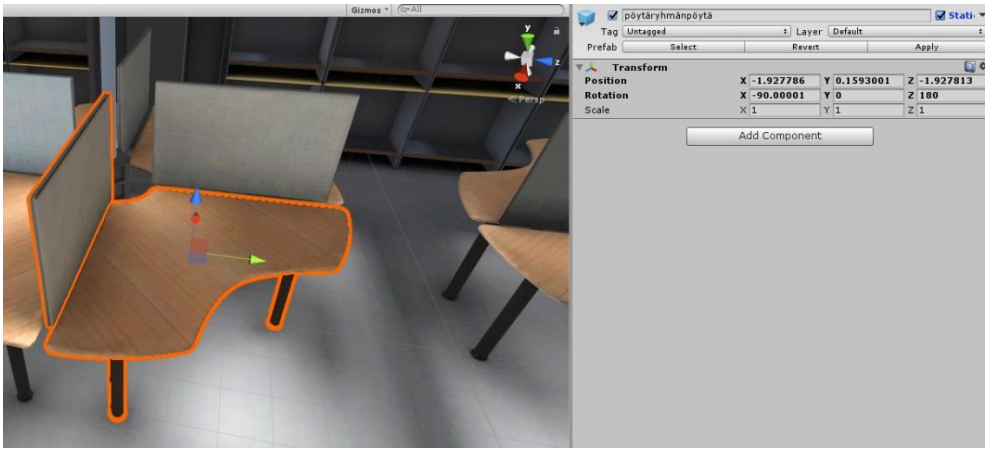
4.4.7 Shaderit

Kaikista yksinkertaisimmat shaderit kuten Unityssä Mobile > Unlit (Supports Lightmap) (Unity Technologies n.d.) tai Unrealissa Default Lit ja Unlit (Unreal Engine Documentation n.d.), ovat Gear VR:lle parhaita.

4.4.8 Piirtokäskyjen niputtaminen (Draw Call Batching)

Sekä Unityssä että Unrealissa, jos objektia ei liikuteta, pyöritetä tai skaalata, valitaan objekti staattiseksi (kuva 51; kuva 52).

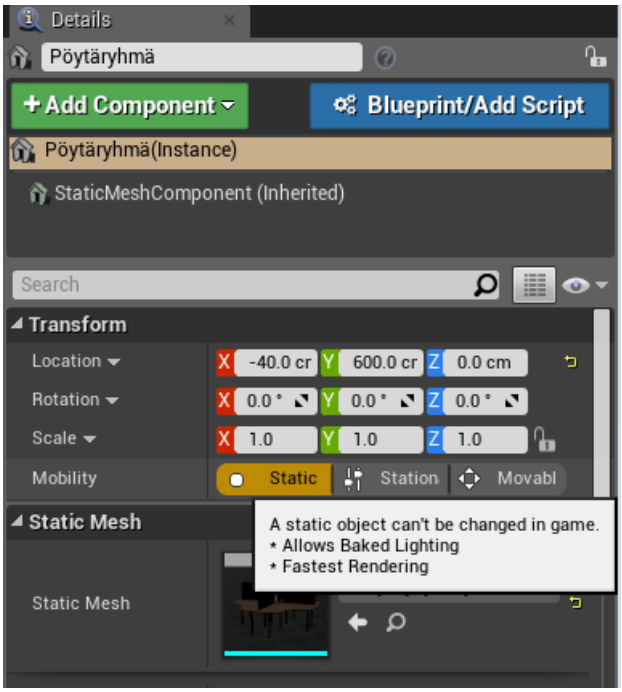
Unityssä kaikki staattiset objektit niputetaan yhteen, jos niillä on sama materiaali. Näin saadaan piirrettyä kerralla suuri, automattisesti tuotettu Mesh-objekti. Tällä saadaan vähennettyä huomattavasti piirtokäskyjä, mutta sovellus kuluttaa enemmän muistia. Myös geometrian monimutkaisuus voi joillakin alustoilla vaikuttaa siihen, mitkä kaikki objektit pystytään niputtamaan kerralla. (Unity Documentation n.d.)



Kuva 51. Unityssä staattiseksi muutettu objekti.

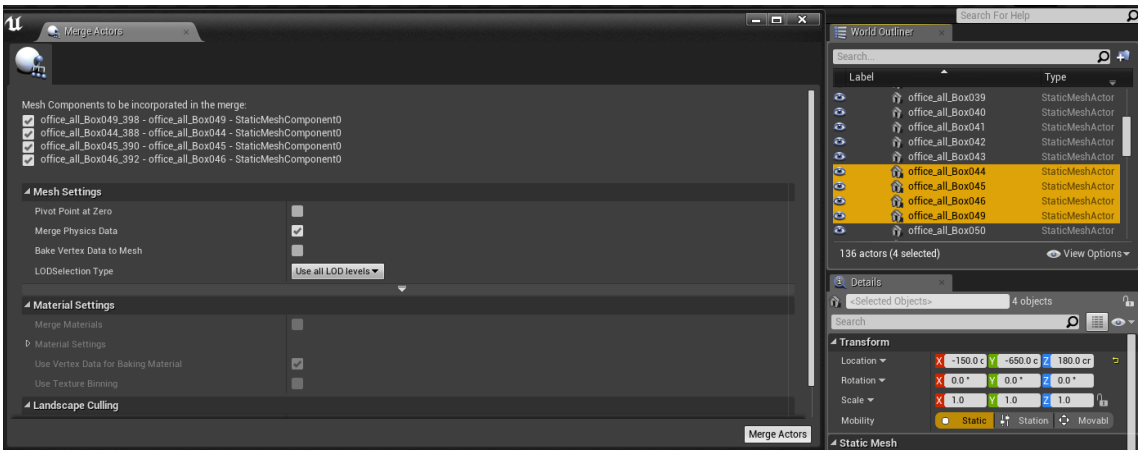
Jos objekti liikkuu, on käytettävä dynaamista niputtamista. Prosessi on Unityssä automatisoitu. Jos Mesh-objekti on tarpeeksi pieni, CPU laskee yhteen samankaltaiset verteksit (verticles) ja piirtää nämä yhtä aikaa. Operaatio vie laskentatehoa CPU:lta, mutta useissa tapauksissa lisää suorituskykyä. (Unity Documentation n.d.)

Kuten Unityssä, myös Unrealissa objekteja voi asettaa staattisiksi (kuva 52). Jotta piirtokäskyjä saataisiin kuitenkin huomattavasti vähennettyä Unrealissa, on käytettävä Merging Actors -työkalua samanlaisille objekteille, joissa käytössä samat materiaalit. Yhdistämällä saman materiaalin mesh:it yhteen, saadaan vähennettyä piirtokäskyjä. Prosessi on aikaavievämpi, kuin Unityssä. Yksittäisiä objekteja ei voi siirrellä, pyörittää tai skaalata Merge Actors -operaation jälkeen.



Kuva 52. Unrealissa staattiseksi merkitty objekti.

Merge Actors ikkunan saa auki Window > Developer Tools > Merge Actors kautta. Halutut Objektit valitaan World Outlinerissä ja tämän jälkeen ne voidaan yhdistää, kuten kuvassa 53 näkyy.



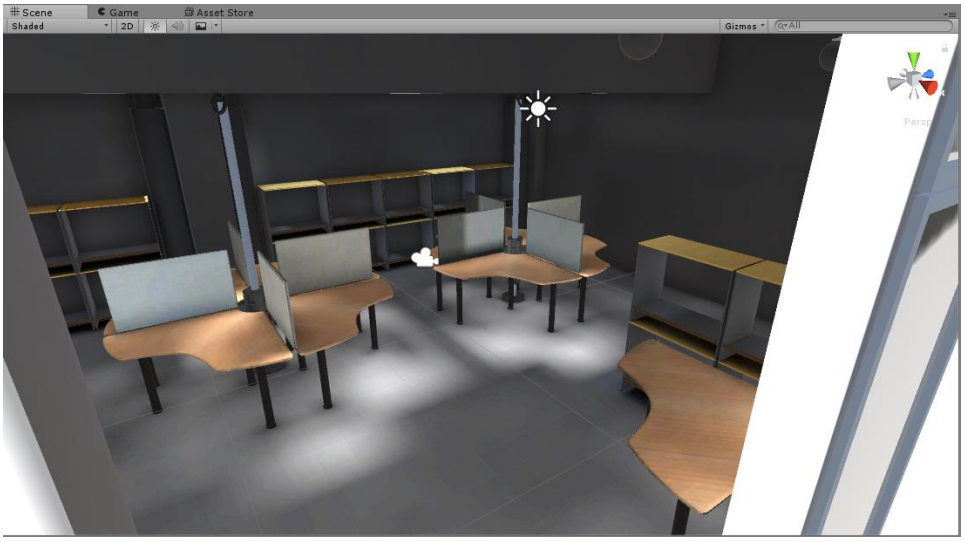
Kuva 53. Merge Actors -ikkuna.

Unityssä voidaan käyttää myös samaa tekniikkaa. Mesh-objekteja voidaan yhdistää Mesh.CombineMeshes-funktion avulla (Unity Documentation n.d.). Unity-projekteissa saatiin kuitenkin tekemissäni sovellusprototyypeissä tiputettua piirtokäskyjen määrä tarpeeksi alas ilmankin CombineMeshes-tekniikkaa, staattisella ja dynaamisella niputuksella.

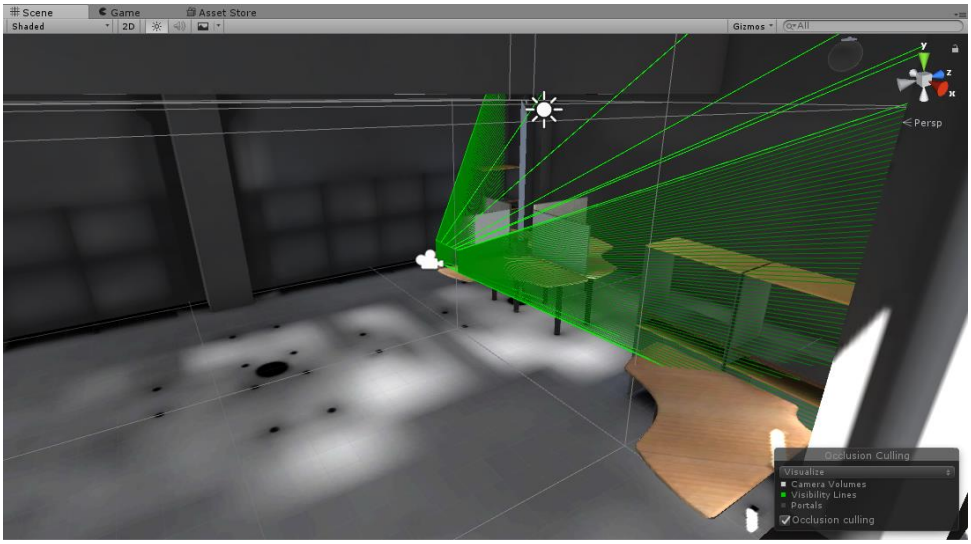
4.4.9 Occlusion Culling

Occlusion Culling on ominaisuus, joka estää kameralle näkymättömissä olevien objektien renderöinnin. Esimerkiksi Unityssä prosessi etenee niin, että virtuaalinen kamera käy arvioimassa eri kohdissa, mikä objekti on milläkin hetkellä näkyvissä ja mikä ei. Näin ollen vain näkyvissä olevat objektit pitää renderöidä, mikä vähentää piirtokäskyjen määrää ja lisää sovelluksen suorituskykyä. (Unity Documentation n.d.)

Kuvissa 54 ja 55 nähdään, miten Occlusin Culling vaikuttaa Scene-näkymään. Kuvasta 55 huomataan myös, että jos pienikin osa objektia on kameralle näkyvissä, kuten pöydänkulma, piirretään silti koko objektin pinta. Joskus suuret kappaleet voi olla järkevä pilkkoa pienemmiksi paloiksi Occlusion Culling -laskentaa varten.

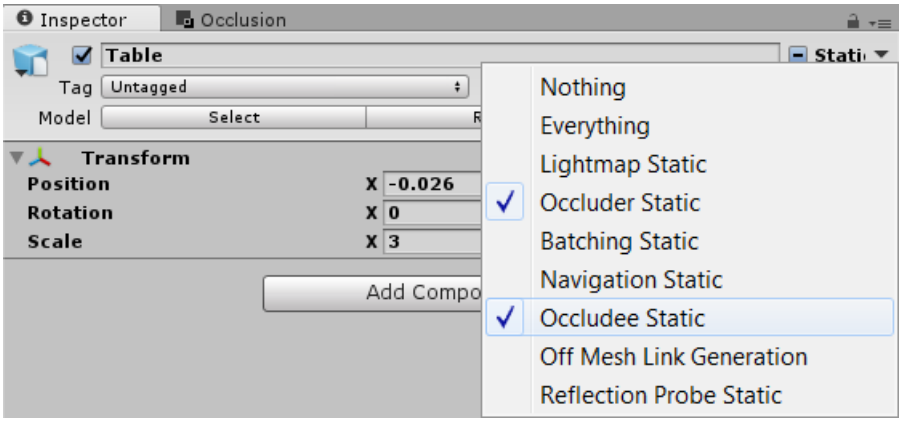


Kuva 54. Tavallinen Scene-näkymä.



Kuva 55. Occlusion Culling -visualisointi. Vain kameralle näkyvät osat piirretään.

Unreal-projekteissa Occlusion Culling on jo valmiina päällä. Unity-projekteissa Occlusion Culling -ikkunan saa auki Window > Occlusion Culling. Tämän jälkeen OC-datan pääsee laskemaan (bake), kuten kuvassa 57 näkyy. Jotta objektit voidaan ottaa mukaan Occlusion Culling -laskentaan, on nämä ennen laskentaa merkittävä Occluder Static ja Occludee Static -objekteiksi, kuten kuvassa 56 näkyy.



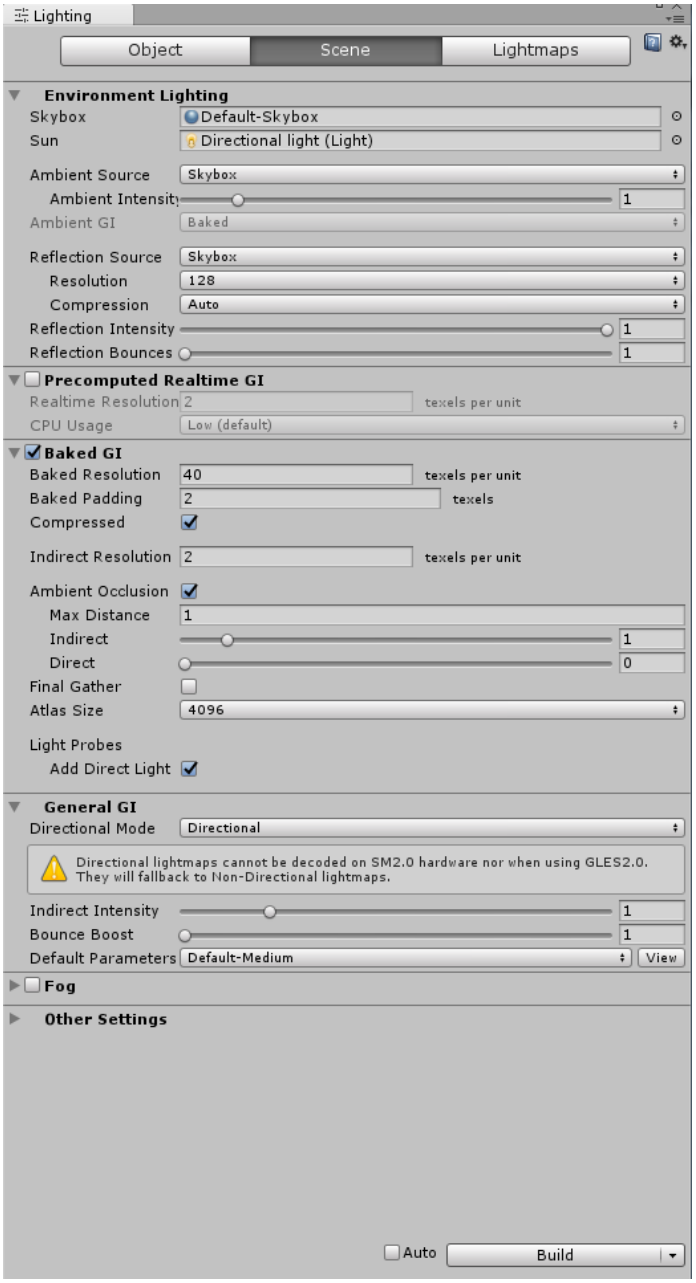
Kuva 56. Objektin merkitseminen.

Kuva 57. Occlusion Culling -ikkuna.

4.4.10 Valaistus

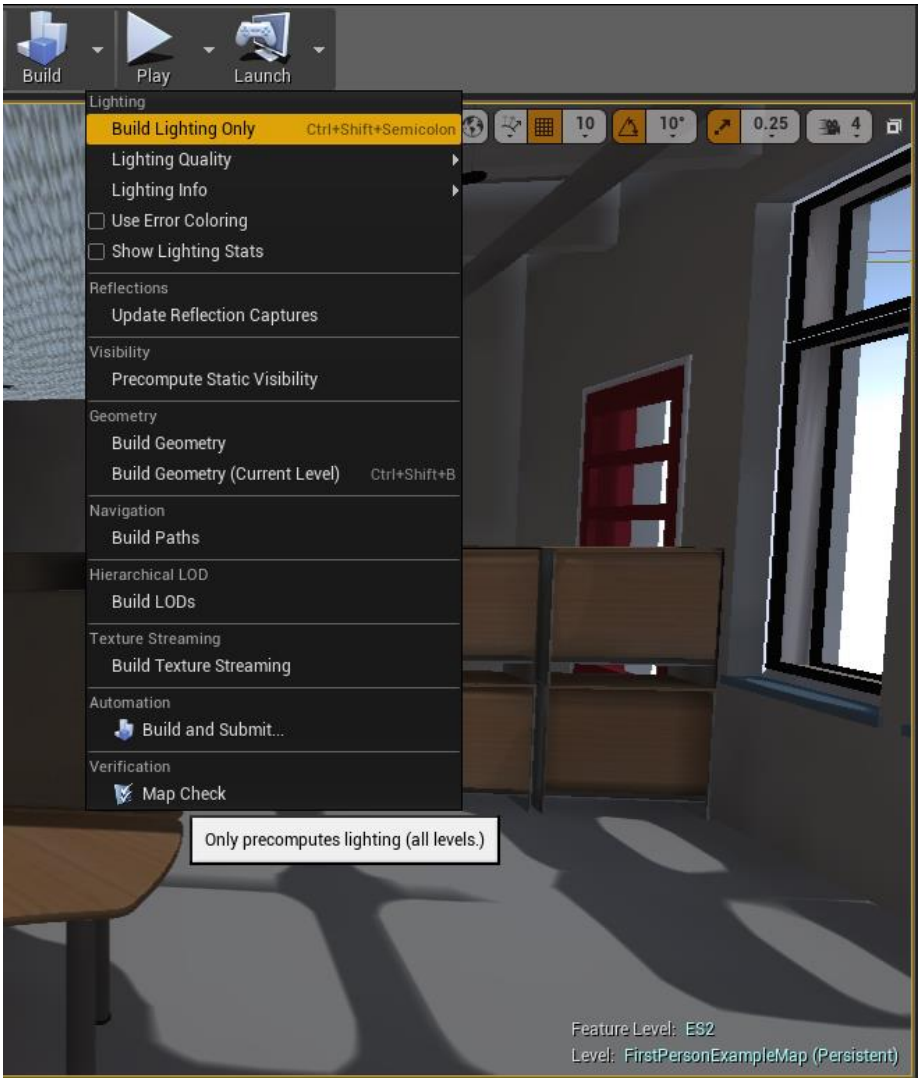
Aina kun mahdollista, kannattaa käyttää valmiiksi laskettua (baked / built) valaistusta. Tämä lisää suorituskykyä, sillä valaistusta ei tarvitse laskea uudelleen kuvaa renderöidessä. Myös reaaliaikaisia varjoja kannattaa välttää. (Unity Technologies n.d.; Unreal Engine Documentation n.d.)

Unityssä valaistuksen pääsee laskemaan Window > Lighting -ikkunan kautta (kuva 58). Arvoja säätämällä voi myös saada aikaan pientä optimointia.



Kuva 58. Unityn Lighting -ikkuna.

Unrealin valaistuksen pääsee laskemaan kuvan 59 mukaisesti. Kannattaa myös varmistaa, että kaikki käytetyt valot ovat asetettu staattisiksi, sillä tämä on kevyin tapa renderöidä valaistus. (Unreal Engine Documentation n.d.)



Kuva 59. Unrealin valaistuksen laskeminen.

4.4.11 Laatu (Quality) Asetukset

Asetuksista saa säädettyä monilla tavoilla visuaalisen ilmeen tasoa. Näitä säätelemällä voidaan vaikuttaa suorituskykyyn visuaalisen laadun kustannuksella. (Unity Technologies n.d.)

Yleisenä ohjeena Pixel Light Count kannattaa Unityssä vaihtaa yhteen. Näin piirrettävä kuva (frame) saadaan piirtymään nopeammin. (Pruett 2015.)

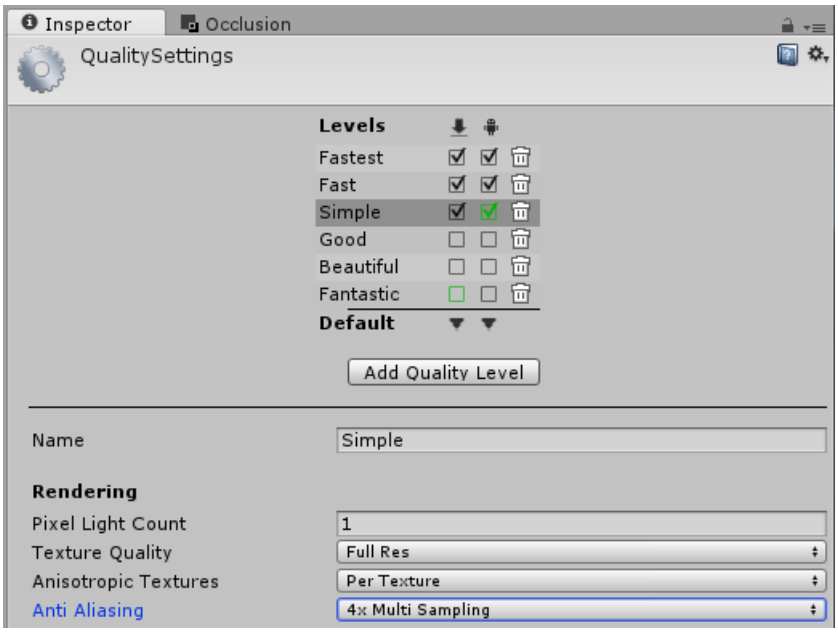
4.4.12 Kuvan jälkikäsittely ja kuvaefektit

Sekä Unityn että Unrealin tapauksissa Gear VR -sovelluksista on karsittava kaikki kuvaefektit (Post-Processing / Image Effects) moottorikohtaisesti paria poikkeusta vaille, sillä ne käyvät liian raskaaksi renderöidessä mobiililaitteella. (Unreal Engine Documentation n.d.; Unity Technologies n.d.)

Unrealin tapauksessa on joissain projekteissa turvallista käyttää Auto Exposure -käsittelyä (Unreal Engine Documentation n.d.).

Anti-Aliasointi on suorituskykyä vievä efekti, mutta sitä suositellaan käytettävän Gear VR -sovelluksissa. Se tekee kuvasta luontevamman ja vähentää rosoisia ja pikselisiä reunoja (Unity Technologies n.d.). Arvo 4 on suositeltava Unityssä ja sitä vastaa Unrealissa Medium.

Unityssä Anti-Aliasointiarvoa pääsee säätämään Edit > Project Settings > Quality polusta. Kuvassa 60 näkyy Quality Settings -näkymä.



Kuva 60. Quality Settings ja Anti-Aliasointi.

4.4.13 Render Scale

Unityssä on Render Scale -arvo, joka kontrolloi texel : pixel -suhdetta ennen linssikorjausta (lens correction). Näin ollen Render Scale -arvolla voidaan vaihtaa suorituskykyä terävyyteen. Default-arvo on 1.0. Arvoa säätämällä alaspäin voidaan saada raskaampikin ympäristö pyörimään sujuvammin, joskin terävyyden kustannuksella. Jos ympäristö pyörii ongelmitta, voidaan arvoa nostaa hieman, jotta saavutetaan terävämpi kuva. (Unity Technologies n.d.)

Seuraavassa koodiesimerkissä GameController-Skripti (skripti 2), jossa osoitetaan uusi Render Scale -arvo.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.VR;

public class GameController : MonoBehaviour {
    private float m_RenderScale = 1.4f;
```

```
    void Start () {
        VRSettings.renderScale = m_RenderScale;

    }

}
```

Skripti 2. GameController-Skripti, joka säätää RenderScale-arvoa.

4.4.14 Yhteenveto

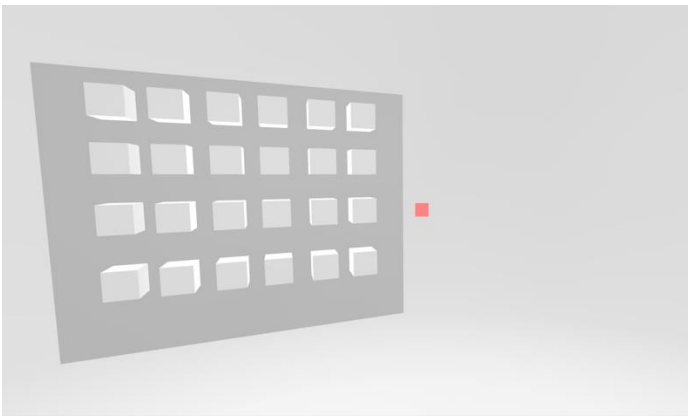
Optimoidessa näkymää päästiin parempiin tuloksiin Unity-pelimoottorilla. Toisaalta Unityn ollessa minulle tutumpi moottori, ei tulos ole objektiivisesti pätevä. Unrealilla tehdyssä tuotoksessa kuvataajuus (frame rate) tippuu jatkuvasti ja tämä aiheuttaa jatkuvaa nykimistä. Unity-tuotoksessa pystyttiin lisäämään muutama lisäyksityiskohta ja vain raskaimmissa näkymissä kuvataajuus (frame rate) tippuu hetkellisesti noin 47 fps ja aiheuttaa nykimistä.

4.5 **Vuorovaikutus**

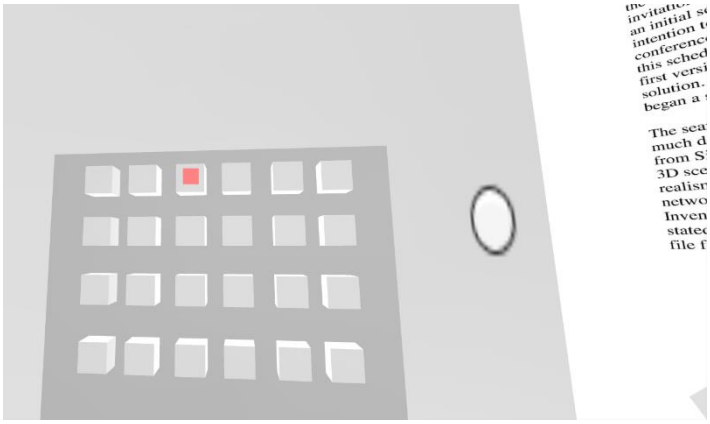
Vuorovaikutuksesta tehtiin kaksi esimerkkisovellusta. Kummatkin toteutukset tehtiin Unityllä, sillä tämä osoittautui helpommaksi optimoida.

4.5.1 Katseen suunnan tarkastelu ja valinnat

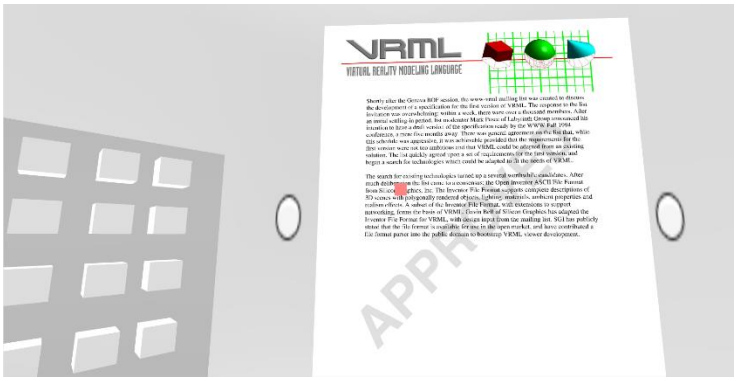
Sovellustestauksessa oli tarkoituksena luoda ympäristö, jossa objekteja katsomalla ja nappia painamalla saadaan avattua objektiin liitetty tiedosto tai tiedostot. Testiin luotiin yksinkertainen laatikkomalli, joka simuloi sähkökaappia, kuten kuvassa 61 näkyy. Sähkökaapin objekteja (valkoiset laatikot) katsomalla ja ohjaimen valintanappia painamalla saadaan avattua objektiin liitettyjä kuvia, kuten kuvassa 62 näkyy. Jos kuvia on liitetty enemmän kuin yksi, voidaan kuvaa vaihtaa vasemmalle ja oikealle ilmestyvistä napeista, kuva 63.



Kuva 61. Sähkökaappinäkymä.



Kuva 62. Kuvan avaaminen.



Kuva 63. Tarkasteltava kuva, napit molemmin puolin.

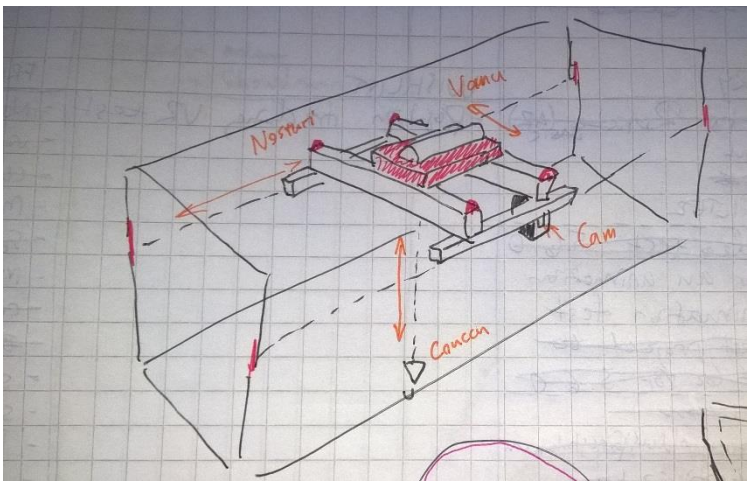
Toiminnollisuus on toteutettu Raycast-metodin avulla, kuten skriptistä 3 nähdään. Jos säde osuu nappeihin, kutsutaan niitä vastaavat funktiot. Muutoin vaihdetaan kuvataulua ja näin näytetään uudet kuvat.

```
void Update(){

        if(Input.GetButtonDown("Fire1")){
            Ray ray = new Ray(camera.transform.position, camera.trans-
form.forward);
            RaycastHit hit;

            if (Physics.Raycast(ray, out hit)) {
                if (hit.trans-
form.name == "RIGHT" || hit.transform.name == "LEFT") {
                    if (hit.transform.name == "RIGHT") {
                        RIGHT ();
                    } else {
                        LEFT ();
                    }

                } else {
                    Transform objectHit = hit.transform;
                    changeImageArray (objectHit);
                }
            }
        }
    }
```

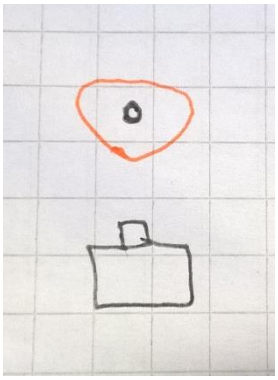
Skripti 3. Raycast-tarkastelu.

4.5.2 Ohjaimella tapahtuva nosturin ohjaaminen

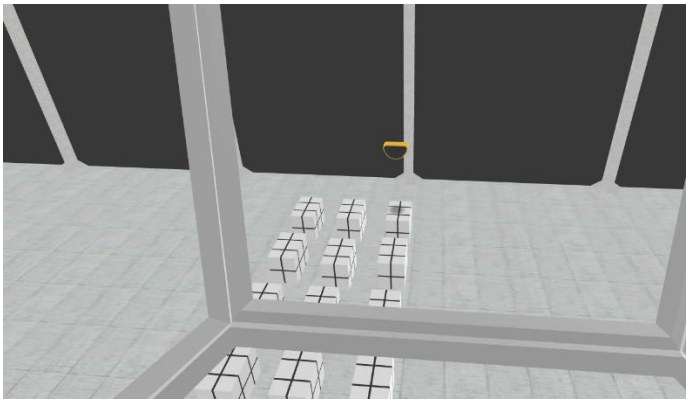
Tarkoituksena oli tuottaa yksinkertainen nosturisimulaatio, jossa käyttäjä pääsee vapaasti katsomaan ympärilleen ohjaamosta. Malleina käytettiin vähä-polygonisia malleja. Liikkuminen suunniteltiin tapahtuvan kuvan 64 mukaisesti. Peliohjaimen tikuilla ohjataan nosturia ja vaunua ja koukkua pääsee laskemaan ja nostamaan napeista. Laatikkoon tartutaan automaattisesti, kun koukku on tarpeeksi lähellä laatikkoa, kaavakuvassa 65 nähdään laatikon tarttumapinta. Otteen voi irrottaa laatikosta nappia painamalla, jolloin laatikko tippuu fysiikan lakien mukaisesti. Kuvissa 66, 67 ja 68 nähdään lopputulos.



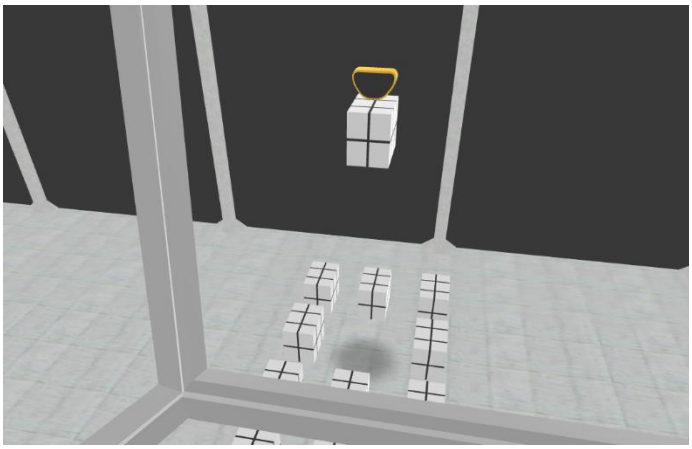
Kuva 64. Nosturisimulaation pohjapiirustus.



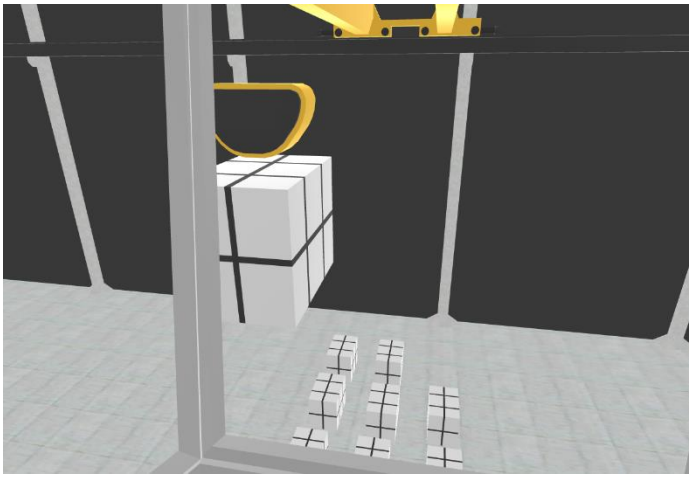
Kuva 65. Koukku ja laatikon tarttumapinta.



Kuva 66. Koukku laatikon yläpuolella.



Kuva 67. Laatikkoon tarttunut koukku.



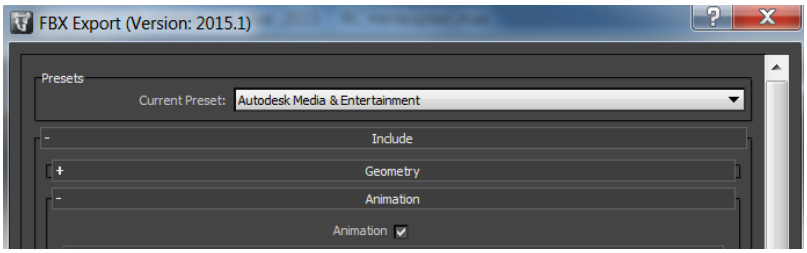
Kuva 68. Nostettu laatikko.

4.6 **Animaatiot ja Videot**

Kappaleessa esitellään, miten animaatiota ja videoita saa pyörimään Unity-pelimoottorilla.

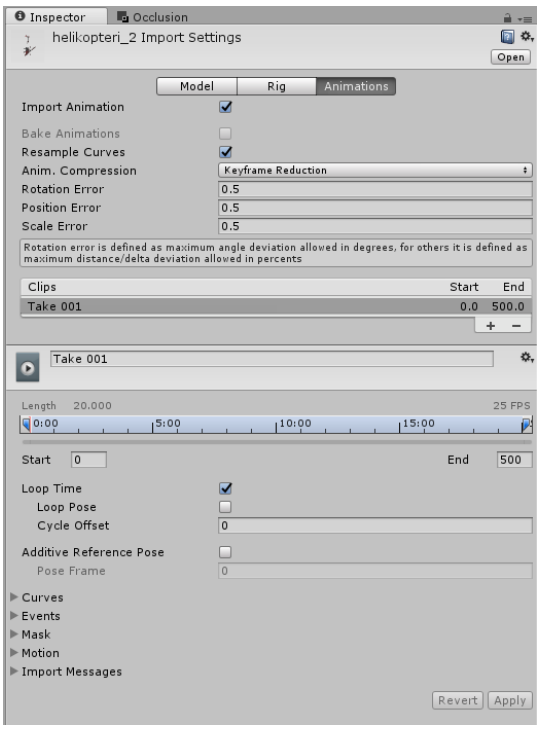
4.6.1 Animaation tuominen 3ds Maxista ja pyörittäminen Unityssä

Jotta animaatio saadaan näytettyä Unityssä, malli tuodaan kuten aiemminkin Maxista. Huolehditaan kuitenkin että FBX Export -ikkunassa Animation-valikossa on Animation-optio valittuna, kuten kuvassa 69 nähdään.



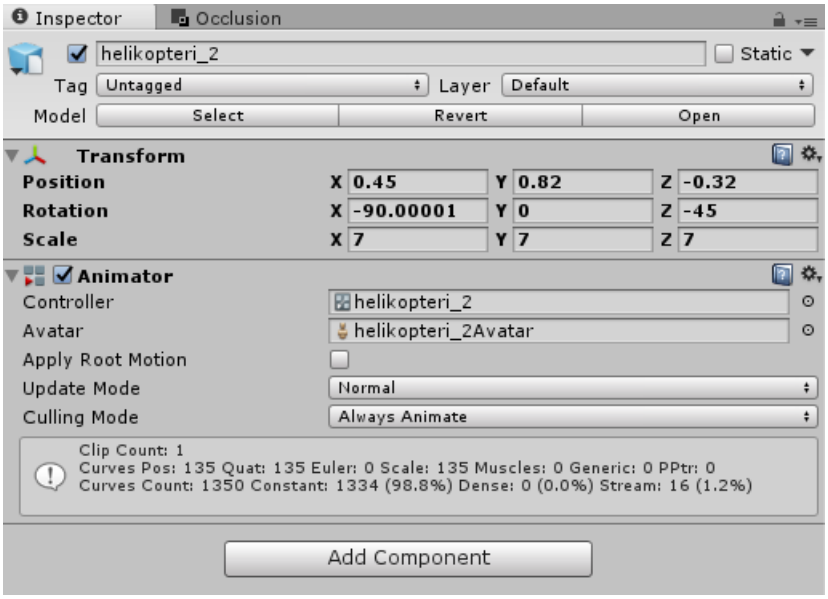
Kuva 69. FBX Export -ikkunan Animation-optio.

Tämän jälkeen malli tuodaan Unityyn ja Import-ikkunasta varmistetaan että Import Animation -optio on valittu, kuten kuvassa 70 näkyy. Jos animaatio tahdotaan pyörimään ilman loppupistettä (loop), valitaan Loop Time. Tuodaan malli Scene-näkymään.

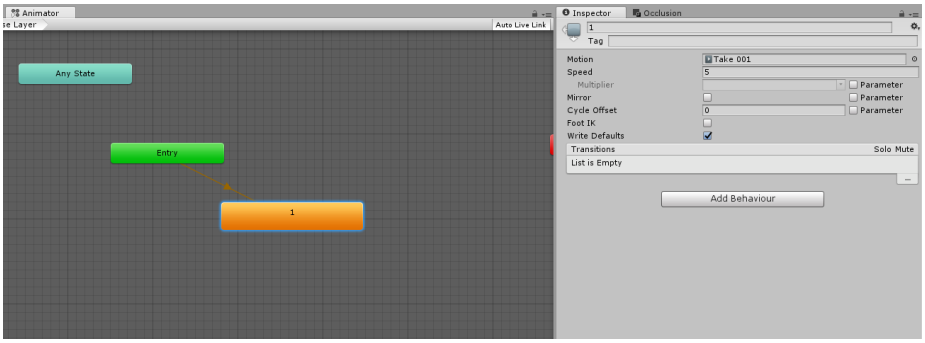


Kuva 70. Import Settings ja Animation -välilehti.

Seuraavaksi objektille luodaan Animation Controller ja se asetetaan Animator-komponenttiin, kuten kuvassa 71 nähdään. Tuplaklikkaamalla Controlleria, saadaan avattu Animator-näkymä, jota tarkastellaan kuvassa 72. Valitaan Animaation Motion-kohteeksi mallin mukana tullut Take 001 -animaatio. Tämän jälkeen säädetään vielä nopeus halutuksi.



Kuva 71. Animator-komponentti.



Kuva 72. Animator-työkalu.

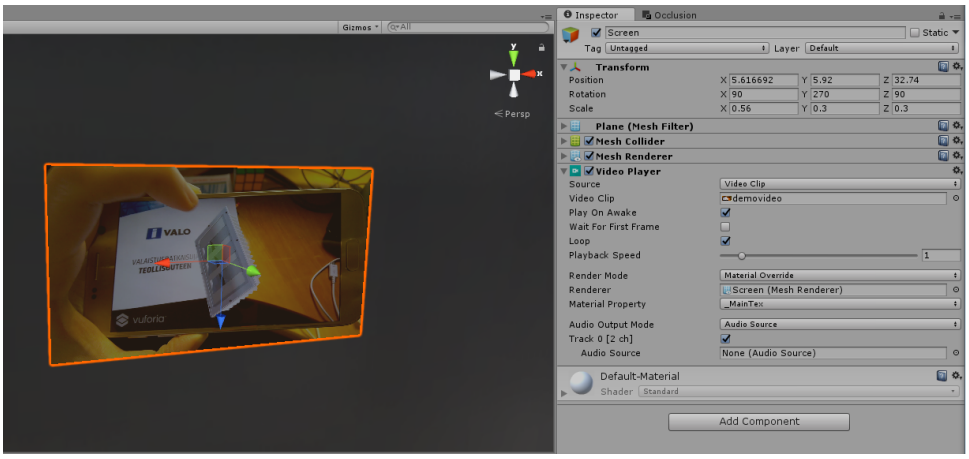
Ajaessa peliä, objektin osat pyörivät kuten ne oli Maxissa animoitu. Kuvassa 73 nähdään still-kuva pyörivästä animaatiosta.

Kuva 73. Otos pyörivästä animaatiosta.

4.6.2 Unityn videotoistin

Aiemmissa sovellustestauksissa käytettiin Unity 5.5.2f1 -versiota. Kyseisessä versiossa videoiden pyörittäminen oli erittäin hankalaa. Uusimmassa Unityn versiossa 5.6.0f3 tuotiin mukaan uusi Video Player -luokka, joka kehitettiin erityisesti VR-ympäristöjä ajatellen. Tämän takia videotestit tehtiin Unityn uusimmalla versiolla.

Kuvassa 74 on esimerkki Video Player -komponentin käytöstä. Kuten kuvassa näkyy, Plane-objektiin on liitetty Video Player -komponentti. Video Clip -kenttään annetaan haluttu video ja lopuista optioista säädetään esimerkiksi loop, toistonopeus ja renderöintimoodi.



Kuva 74. Video Player yhdistettynä Plane-objektiin.

5 JOHTOPÄÄTÖKSET

Kaikki toivotut sovellustestaukset saatiin toimimaan Unity-pelimoottorilla. Myös optimointi saatiin toimimaan hyvin annetuissa raja-arvoissa. Unreal-moottorilla ei saatu yhtä hyviä optimointituloksia, eikä näin ollen lähdetty tekemään monimutkaisempia sovellustestauksia.

En lähtisi sanomaan kumpi moottori on huonompi tai parempi. Osaavissa käsissä kummasta tahansa voidaan saada aikaiseksi erittäin hyvää jälkeä. Koen itse Unityn helppokäyttöisemmäksi moottoriksi ja se on mielestäni helpompi optimoida, sillä esimerkiksi piirtokäskyjen niputus on automatisoitu. Tuloksiin vaikutti se, kuinka paljon enemmän kokemusta minulla oli Unitystä.

Gear VR -laitteisto asettaa tietyt vaatimukset sisällölle, jota halutaan tuottaa. Tietokoneen ympärillä pyörivän virtuaalilaitteiston avulla saadaan luotua graafisesti hienompaa sisältöä, sillä laskentatehoa on huomattavasti enemmän. Mobiili VR -sovelluksia luotaessa on hyvä ymmärtää raja-arvot, joiden sisällä pyöritään. Tämä vaikuttaa esimerkiksi siihen, miten monimutkaisia malleja voidaan pyörittää sovelluksessa, tai kuinka paljon dynaamisia objekteja, animaatioita tai efektejä voi olla näkymässä.

Opin käyttämään Unreal-moottoria kohtalaisesti. Eniten kuitenkin opin Unityn optimoinnista ja sovellusten optimoinnista ylipäänsä. Opin myös paljon virtuaalitodellisuudesta. Sovelluskehityksessä oli hyötyä ymmärtää syvemmin esimerkiksi vaatimuksista, jotka ihmisen biologia asettaa virtuaalitodellisuuslaseille ja -sovelluksille. Oli myös hyvä ymmärtää, mitä nykylaseilla voidaan tuottaa ja minkälaisia sovelluksia laseille on jo tuotettu.

LÄHTEET

Abrash, M. (2015). Why Virtual Reality Will Matter to You. Julkaistu 26.3.2015. Haettu 28.03.2017 osoitteesta https://www.youtube.com/

Amazon Web Services (n.d.). Lumberyard FAQ. Haettu 07.04.2017 osoitteesta https://aws.amazon.com/lumberyard/faq/

Autodesk Stingray Help (n.d.). Get Started in VR. Haettu 07.04.2017 osoitteesta http://help.autodesk.com/view/Stinray/ENU/?guid=__stingray_help_getting_started_get_started_vr_html

Autodesk Stingray Help (n.d.). Lua or Flow: which do I use? Haettu 07.04.2017 osoitteesta http://help.autodesk.com/view/Stingray/ENU/?guid=__stingray_help_creating_gameplay_lua_vs_flow_html

Billinghurst, M. (2016). Introduction to Virtual Reality. Haettu 20.04.2017 osoitteesta https://www.slideshare.net/marknb00/comp-4010-lecture-1-introduction-to-virtual-reality?qid=49cfd826-3082-4ba4-966b-ff2e2014e395&v=&b=&from_search=1

Cryengine Documentation (2016). VR Support. Haettu 07.04.2017 osoitteesta http://docs.cryengine.com/display/CEMANUAL/VR+Support

Crytek (n.d.). Features. Haettu 04.04.2017 osoitteesta https://www.cryengine.com/features

Deemec Oy (n.d.). Etusivu. Haettu 06.04.2017 osoitteesta http://www.deemec.com/

Dvorsky, G. (2016). Paralyzed Patients Learn to Walk Again Using Virtual Reality. Haettu 05.04.2017 osoitteesta http://gizmodo.com/paralyzed-patients-learn-to-walk-again-using-virtual-re-1785162361

Epic Games (n.d.). What is Unreal Engine 4?
Haettu 04.04.2017 osoitteesta
https://www.unrealengine.com/what-is-unreal-engine-4

Experimental Game Design – Postgaming (2016). A brief history of Virtual Reality. Haettu 20.04.2017 osoitteesta
http://mycours.es/gamedesign2016/presentations/a-brief-history-of-virtual-reality/

Langley, H. (2016). This is what virtual reality will (probably) look like in 2021. Haettu 04.04.2017 osoitteesta
https://www.wareable.com/vr/michael-abrash-what-vr-will-look-like-in-2021

Marchant, J. (2017). The surgeon giving his patients VR instead of sedatives. Haettu 29.03.2017 osoitteesta
http://www.bbc.com/future/story/20170202-the-surgeon-using-virtual-reality-instead-of-sedatives

Neal, M. (2014). This Is What It's Like to Be Anesthetized by Virtual Reality. Haettu 29.03.2017 osoitteesta
https://motherboard.vice.com/en_us/article/this-is-what-its-like-to-be-anesthetized-by-virtual-reality

Oculus Documentation (n.d.). Debugging and Performance Analysis in Unity. Haettu 31.03.2017 osoitteesta
https://developer3.oculus.com/documentation/game-engines/latest/concepts/unity-integration-perf/

Oculus Documentation (n.d.). Introduction to Best Practices.
Haettu 28.03.2017 osoitteesta
https://developer3.oculus.com/documentation/intro-vr/latest/concepts/bp_intro/

Oculus Documentation (n.d.). Oculus Documentation Overview.
Haettu 27.03.2017 osoitteesta
https://developer3.oculus.com/documentation/game-engines/latest/

Oculus Documentation (n.d.). Testing and Performance Analysis in Unreal. Haettu 31.03.2017 osoitteesta
https://developer3.oculus.com/documentation/game-engines/latest/concepts/unreal-debug/

OculusRift Blog (n.d.). John Carmack's delivers some home truths on latency. Haettu 28.03.2017 osoitteesta
http://oculusrift-blog.com/john-carmacks-message-of-latency/682/

OSVR Developer Portal (n.d.). Integrating OSVR with Unity.
Haettu 07.04.2017 osoitteesta
http://osvr.github.io/doc/unity/

OSVR Developer Portal (n.d.). Integrating OSVR with Unreal Engine.
Haettu 07.04.2017 osoitteesta
http://osvr.github.io/doc/unreal/

Prasuethsut, L. (2016). Oculus Connect 3: Big news about Oculus Touch, Rift and much more. Haettu 05.04.2017 osoitteesta
https://www.wareable.com/vr/oculus-connect-3-oculus-touch-standa-lone-rift-social-vr

Proteus Vr labs LTD (n.d.). Time Travel Through Virtual Reality.
Haettu 20.04.2017 osoitteesta
https://www.freeflyvr.com/time-travel-through-virtual-reality/

Pruett, C. (2015). Squeezing Performance out of your Unity Gear VR Game.
Haettu 10.04.2017 osoitteesta
https://developer3.oculus.com/blog/squeezing-performance-out-of-your-unity-gear-vr-game/

Shanklin, W. (2016). 2016 VR Comparison Guide.
Haettu 28.03.2017 osoitteesta
http://newatlas.com/best-vr-headsets-comparison-2016/45984/

tomodachi.wikia.com (n.d.). Virtual Boy.
Haettu 20.04.2017 osoitteesta
http://tomodachi.wikia.com/wiki/Virtual_Boy

Unity Documentation (n.d.). Draw Call Batching.
Haettu 10.04.2017 osoitteesta
https://docs.unity3d.com/Manual/DrawCallBatching.html

Unity Documentation (n.d.). Mesh.CombineMeshes.
Haettu 10.04.2017 osoitteesta
https://docs.unity3d.com/ScriptReference/Mesh.CombineMeshes.html

Unity Documentation (n.d.). Occlusion Culling.
Haettu 10.04.2017 osoitteesta
https://docs.unity3d.com/Manual/OcclusionCulling.html

Unity Technologies (n.d.). Getting Started with VR Development.
Haettu 10.04.2017 osoitteesta
https://unity3d.com/learn/tutorials/topics/virtual-reality/getting-started-vr-development

Unity Technologies (n.d.). Multiplatform.
Haettu 04.04.2017 osoitteesta
https://unity3d.com/unity/multiplatform

Unity Technologies (n.d.). Optimisation for VR in Unity.
Haettu 27.03.2017 osoitteesta
https://unity3d.com/learn/tutorials/topics/virtual-reality/optimisation-vr-unity

Unreal Engine Documentation (n.d.). Samsung Gear VR Best Practices.
Haettu 10.04.2017 osoitteesta
https://docs.unrealengine.com/latest/INT/Platforms/GearVR/BestPractices/

Unreal Engine Documentation (n.d.). Unreal Engine 4 For Unity Developers. Haettu 04.04.2017 osoitteesta
https://docs.unrealengine.com/latest/INT/GettingStarted/FromUnity/

Virtual Reality Society (n.d.). Applications of Virtual Reality.
Haettu 29.03.2017 osoitteesta
https://www.vrs.org.uk/virtual-reality-applications/

Virtual Reality Society (n.d.). History Of Virtual Reality.
Haettu 20.04.2017 osoitteesta
https://www.vrs.org.uk/virtual-reality/history.html

Virtual Reality Society (n.d.). Immersion.
Haettu 27.03.2017 osoitteesta
https://www.vrs.org.uk/virtual-reality/immersion.html

Virtual Reality Society (n.d.). What is Virtual Reality?
Haettu 27.03.2017 osoitteesta
https://www.vrs.org.uk/virtual-reality/what-is-virtual-reality.html

Visbox (n.d.). VisCube™ C4.
Haettu 20.04.2017 osoitteesta
http://www.visbox.com/products/cave/viscube-c4/

Whelan, D. (2014). 1995 Virtual IO I-Glasses.
Haettu 20.04.2017 osoitteesta
http://www.virtualrealityreviewer.com/1995-virtual-io-glasses/

Wiltshire, A. (2017). How many frames per second can the human eye really see? Haettu 05.04.2017 osoitteesta
http://www.pcgamer.com/how-many-frames-per-second-can-the-human-eye-really-see/