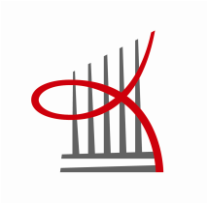


Virtuaalimaailman toteuttaminen Unity 3D -pelinkehitystyökalulla

Case: VirtuaaliViipuri

Eero Kopu

Opinnäytetyö
Lokakuu 2010
Tietojenkäsittelyn koulutusohjelma
Digimedian suuntautumisvaihtoehto
Tampereen ammattikorkeakoulu

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma, Digimedia

Tekijä	Eero Kopu
Työnnimi	Virtuaalimaailman toteuttaminen Unity 3D -pelinkehitys-työkalulla Case: VirtuaaliViipuri
Sivumäärä	39
Valmistumisvuosi	2010
Ohjaaja	Petri Heliniemi
Työn tilaaja	VirtuaaliViipuri-projekti, TAMK

Tiivistelmä

Opinnäytetyön toimeksiantajana toimi Tampereen ammattikorkeakoulun alainen VirtuaaliViipuri-projekti. Opinnäytetyössä tarkoituksena oli selvittää, voisiko Unity 3D -pelinkehitystyökalua soveltaa VirtuaaliViipurin liikkuvan maailman toteuttamiseen. Osana selvitystä toteutettiin erilaisia demoja, joita VirtuaaliViipuri-projektin parissa jatkossa työskentelevät voivat käyttää hyödyksi, mahdollista laajempaa kokonaisuutta tehdessä.

Selvityksen lopputuloksena saavutettiin käsitys siitä, millaisia kokonaisuuksia VirtuaaliViipurista Unity 3D:llä on järkevä käyttää hyväksi.

Hyötyä opinnäytetyöstä saavat niin VirtuaaliViipuri-projektin parissa jatkavat opiskelijat kuin muutkin Archicadilla toteutettujen mallien esittelykäyttöön valjastamista suunnittelevat.

Tämän opinnäytetyön tarkoituksena on myös antaa reaaliaikaiseen grafiikkaan perehtymättömälle käsitys siitä, mitä reaaliaikainen grafiikka tarkoittaa, millaisia mahdollisuuksia ja millaisia rajoitteita sillä on.

Avainsanat Reaaliaikainen grafiikka, 3D-grafiikka, Archicad, Cinema 4D, Unity 3D

TAMK University of Applied Sciences

Writer	Eero Kopu
Thesis	The Creation of a Virtual World with Unity 3D Game Development Tool Case: VirtualViipuri
Pages	39
Graduation time	2010
Thesis Supervisor	Petri Heliniemi
Co-operating Company	VirtualViipuri -project, TAMK

Abstract

This thesis is a case work for the VirtualViipuri project of TAMK. The purpose of the thesis was to find out if a virtual world of VirtualViipuri could be made with the use of Unity 3D game development tool.

As a part of the thesis a few demo versions of the virtual world were produced. These demos can be used as a reference material for future development of the virtual world of VirtualViipuri. The making of these demos gave a clear vision of what kind of a virtual world could and should be done with the existing models of Viipuri.

This thesis also sheds light on what real-time graphics are and what kind of possibilities and requirements they have.

Keywords Real-Time Graphics, 3D-graphics, Archicad, Cinema 4D, Unity 3D

Sisällysluettelo

1 Johdanto

Tein koko harjoittelujaksoni ajan 3D-malleja, pääasiassa erilaisia patsaita Cinema 4D:llä VirtuaaliViipurin virtuaalimaailmaa varten. VirtuaaliViipuri on opiskelijavoimin edistyvä hanke, ja sitä kuvaillaan VirtuaaliViipurin Internet-sivuilla seuraavasti: ”*Hankkeen päätavoitteena on luoda 3D-mallinnuksella www-ympäristöön virtuaalinen Viipurin kaupunki sellaisena kuin se oli syyskuussa 1939 suomalaisena kaupunkina.*” (VirtuaaliViipuri, 2010)

Harjoittelun alussa projektissa toimiva Toni Pippola ehdotti, että tutustuisin Unity 3D:n mahdollisuuksiin VirtuaaliViipurin liikkuvan maailman toteuttamiseksi. Päätin tutustua aiheeseen vasta opinnäytetyönä, koska harjoittelun aikana mielenkiintoista työsarkaa riitti myös 3D-mallinnuksen puolella ja halusin ehdottomasti kehittää itseäni sillä saralla.

VirtuaaliViipuri-projektissa liikkuva maailma määritellään seuraavasti:

> *Hankkeen päätavoitteena on luoda 3D-mallinnuksella WWW-ympäristöön virtuaalinen Viipurin kaupunki sellaisena kuin se oli syyskuussa 1939 suomalaisena kaupunkina.* (VirtuaaliViipuri, 2010)
>
> *Lopullisena tavoitteena on, että kaupungin kaduilla liikutaan katutasolla, lähempänä olevat kohteet* (ovat) *tarkkoja* (ja kauempana esitettäviä kohteita karsitaan automaattisesti). *VirtuaaliViipuriin pyritään saamaan elämää liikkuvien ihmisten, autojen, raitiovaunujen, jne muodossa. Projektin toteutusvaiheen aikana selvitetään vapaata liikkumista kaupungissa.* (VirtuaaliViipuri, 2010)

Kuvatun kaltaisen pelimaailman luominen toimi jonkin tasoisena tavoitteena aiheeseen perehtyessäni. Kaikki mainitut asiat ovat ainakin periaatteellisella tasolla mahdollisia toteuttaa Unity 3D:llä. Ainoa kysymyksiä herättävä asia olikin oikeastaan se, soveltuvatko Archicadilla luodut 3D-mallit tähän käyttötarkoitukseen.

Tampereen ammattikorkeakoulun Rakennustekniikan insinööriopiskelijat ovat vuodesta 2003 lähtien luoneet vuoden 1939 Viipurin rakennuksia Archicad-arkkitehtisovelluksella. Rakennuksien malleista on renderöity kuvia ja videoita VirtuaaliViipurin

Internet-sivuille. Näitä malleja olisi tarkoitus käyttää hyväksi virtuaalimaailman luomisessa.

Opinnäytetyön toimeksiantona oli siis saada aikaan käsitys Unity 3D:n mahdollisuuksista VirtuaaliViipurin liikkuvan maailman toteuttamiseksi. Selvitystyötä tehdessä syntyi liikkuvan maailma demo-versioita, eli eräänlaisia esittelyversioita. Tällaisia demoja ei ole hiottu loppuun asti, mutta niiden avulla pystytään esittelemään käytettyjen menetelmien ominaisuuksia ja mahdollisuuksia. Kyseisten demojen onnistumisesta riippuen työn tuloksena saattoi olla esimerkiksi toimiva liikkuvan maailman pohja, tietty osa Viipurin kaupungista, jossa liikkuminen on toteutettu tai vaihtoehtoisesti olisi voitu todeta käytössä olevien työkalujen ja 3D-materiaalien soveltumattomuus liikkuvan maailman toteuttamiseen.

Hyvin tärkeää on myös muistaa, että kaiken täytyy näyttää juuri oikealta. Koska kyse on historiallisen tiedon säilyttämisestä ja välittämisestä, on ehdottoman pakollista, että kaikki vastaa todellista Viipuria sellaisena, kuin se oli vuonna 1939.

Sovellusta tehdessä täytyi pitää mielessä myös se, että koko lopullisen sovelluksen käyttäjäryhmällä ei voida olettaa olevan aivan tuliteriä tietokoneita. Tästä syystä sovelluksen täytyisi olla ensisijaisesti kevyt ja toissijaisesti näyttävä.

Opinnäytetyö on suunnattu ihmisille, jotka omaavat jo perustiedot 3D-grafiikasta. Opinnäytetyössä käsittellään virtuaalimaailmoja ja reaaliaikaisen grafiikan periaatteita. Tämän jälkeen käydään läpi joitakin suurimmista reaaliaikaisen grafiikan virstanpylväistä, jotka ovat saattaneet sen alkuajoista tämän päivän fotorealistisiin esityksiin. Lopuksi esittelen opinnäytetyössäni käyttämäni sovellukset ja kuvaan prosessin, jolla Archicad-mallit lopulta saatiin Unity 3D -pelinkehitystyökaluun ja siitä valmiiksi liikkuvaksi maailmaksi.

Opinnäytetyössä käyttämäni kirjallisuus on mielestäni lähdemateriaaliksi sopivaa, niin sisältönsä, kuin luotettavuutensa kannalta. Myös verkkolähteiksi on valittu pääasiassa luotettaviksi koettuja verkkosivustoja ja alan toimijoiden tekstejä. Tekstien asiasisältöä on myös tarkasteltu muista lähteistä, ja todettu päteväksi.

2 Virtuaalimaailma

Virtuaalimaailma pyrkii esittämään todellista maailmaa, eli on toisin sanoen keino- tai virtuaalitodellisuutta. Keinotodellisuuden oikeaa maailmaa heijastelevan ympäristön luomisessa käytetään hyväksi juuri tähän tarkoitukseen soveltuvaa näyttöteknologiaa kuin myös eri aisteja hyväksi käyttäviä tekniikoita. (Puhakka, 2008, 24)

Esimerkiksi vanhoihin valokuviin verrattuna olisi virtuaalimaailma täysin ylivertainen keino historiallisen Viipurin havainnollistamiseen, koska tällöin käyttäjä voisi tutkia Viipuria juuri sillä tavalla ja siitä perspektiivistä kuin itse haluaisi. Lisäksi maailmassa olisi mahdollista vaihdella esimerkiksi valaistusta sen mukaan, miltä alueen halutaan näyttävän. Tällöin maailmaa pystyisi tutkimaan yöllä, talvella, pilvisellä säällä ja vaikka pilvettömänä, aurinkoisena päivänä.

Visuaalisen säädettävyyden lisäksi virtuaalimaailmaan on mahdollista luoda paljon erilaista interaktiota. Esimerkiksi liikkuessa lähelle nähtävyyksiä voitaisiin haluttaessa kuunnella nauhoitettu kertomus nähtävyydestä. Esimerkki tällaisesta nähtiin Valven Half-Life 2-pelin The Lost Coast -demossa, jossa esiteltiin eräänlaisen kommenttiraidan avulla muun muassa uusia HDR-ominaisuuksia. Demossa tietyn pisteen ylitettyään pelin tekijät kertoivat pelaajalle uudistuksista ja siitä, mihin asioihin demoa pelatessa pitää kiinnittää huomiota. (Valve Software, 2005)

Virtuaalimaailman elävöittämisessä on sään, vuorokauden- ja vuodenaikojen vaihteluiden lisäksi tärkeää luonnollisesti maailmaan kuuluva kasvillisuus, eläimet, ihmiset ja ihmiskäden tuottamat objektit. Ulkoasun lisäksi on tärkeää, että maailman fysiikanmallinnus on kunnossa. Objektit eivät saa läpäistä toisiaan tai jäädä leijumaan kaksi metriä maan pinnan yläpuolelle.

2.1 Aiemmin toteutettuja virtuaalikaupunkeja

Kuten aiemmin on mainittua, 3D-mallinnettuja rakennuksia on toteutettu eri tavoin ja erilaisiin tarkoituksiin. Kaupunginisät ja muuten kotiseutuaan rakastavat ihmiset ovat

halunneet esitellä niitä paikalla ja totta kai myös muualla asuville. Hyviä esimerkkejä tällaisista projekteista ovat VRML-tekniikalla toteutettu ja jo verkosta poistunut VirtuaaliTampere, Google Earthia hyödyntävä ja pääasiassa Google Sketch Up:lla mallinnettu VirtuaaliBerliini. Pelien puolelta esimerkiksi käy vuonna 2002 ilmestynyt The Getaway -peli jatko-osineen. Peliin luotiin suuria alueita Lontoosta korkeatarkkuuksisena esityksenä. Kuvassa 1 on kuvankaappaus pelin kolmannen, julkaisemattoman osan mallinnetusta Lontoosta.

Pelimaailmoissa mallien tarkkuuksilla on eroa. Jos kyseessä olevan pelin maailma on suurempi kaupunki, kuten esimerkiksi Grand Theft Auto 4:ssä, ovat sen rakennukset pääasiassa mallinnettu todella yksinkertaisesti, mutta teksturoitu siten, että rakennukset ovat kuitenkin hyvännäköisiä. Myös rakennuksen funktio pelimaailmassa vaikuttaa merkittävästi sen polygonrakenteeseen. Mikäli kyseessä on rakennus, joka liittyy olennaisesti pelin tehtäviin, on se usein myös mallinnettu hieman tarkemmin. Toinen esimerkki on Call of Duty: Modern Warfare 2, jonka kentissä on hieman vähemmän rakennuksia, mutta niidenkin yksityiskohdat on pyritty pitämään pieninä, jolloin pelitilanteen prosessointi olisi nopeampaa. Tämän pelin omakotitaloalueelle sijoittuvan kentän omakotitalot ovat todella yksinkertaisia. Niissä mallinnettu muun muassa ilmastointilaitteet rakennusten seinille, ikkunoiden karmit sekä muita yksinkertaisia objekteja. Kuitenkin lautaseinän laudat ja muut vastaavat objektit on tehty teksturoimalla.

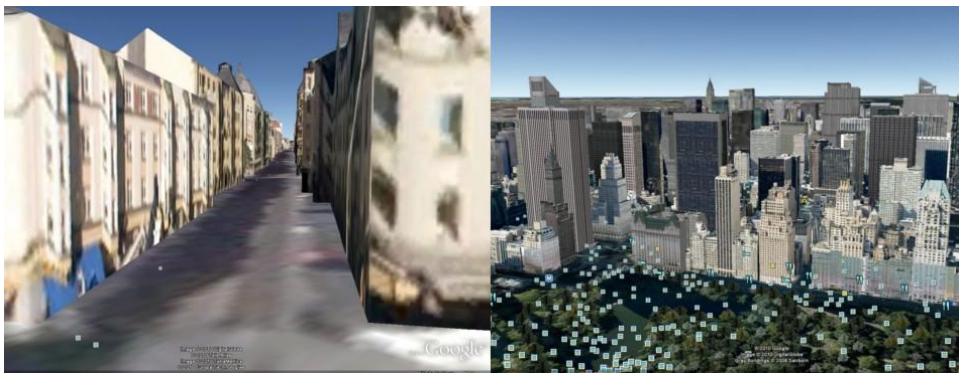
VRML eli Virtual Reality Modelling Language on jo vanhentunut tekniikka, jolla on luotu erityisesti verkkoon 3D-maailman sovelluksia. X3D on syrjäyttänyt sen. Se on monipuolisempi ja tarjoaa paremman tuen 3D-mallinnusohjelmille. Esimerkiksi maailmojen tutkiminen on helpompaa, koska yleisimmille verkkoselaimille on olemassa X3D-liitännäiset, jolloin erillistä ohjelmaa ei välttämättä tarvita.

Kuva 1: Playstation 3:lle ilmestyväksi suunnitellun The Getaway 3 -pelin 3D-mallinnettua Lontoota (Sony Computer Entertainment Europe, 2005)

Oikeaa maailmaa vastaavien virtuaalikaupunkien tutkiminen osoittautui jokseenkin hankalaksi, koska erilaiset projektit esitellään eri nimillä: virtual reality, virtual world, virtual tour ja niin edelleen. Näiden hakusanojen avulla löytää todella paljon muutakin kuin pelkästään virtuaalikaupunkeja.

Koska aihe ei kuitenkaan ollut pääasiallinen tutkimuskohde, päätin yksinkertaisesti selvittää, miten muutamien suurten kaupunkien virtuaaliesittelyt on toteutettu. Paljastui, että kirjoitushetken suosituin tapa luoda virtuaalimaailmoja on Google Earth -ohjelma. Ohjelma on pinnanmuotoja tukeva karttapalvelu, ja sitä varten on suunniteltu myös Google Sketchup -mallinnusohjelma. Googlen ohjelmilla on luotu muun muassa Virtual Berlin. (Virtual Berlin, 2010)

Google Earth -ohjelma mahdollistaa virtuaalimaailmojen yhteisöllisen ja ilmaisen luomistavan. Google Earth -ohjelman suosiota lisää myös se, että maailmojen pohjana toimivat satelliittikuvat, jolloin esimerkiksi vain puoliksi 3D-mallinnetun kaupungin esittely on selkeää ja aukoton sisältönsä osalta, kun aluetta tarkastellaan yläpuolelta. Google Earth on alueiden hahmottamiseen todella hyvä, mutta sen tarkkuus ei riitä VirtuaaliViipurin tarpeisiin. Kuten kuva 2:kin kertoo, Google Earthin kaupunkien visualisointi toimii paremmin kaukaa kuvattuna, kuin aivan katutasosta.



Kuva 2: Kuvassa vasemmalla Google Earthin näkymää Helsingin kaduilta, oikealla ilmakuvaa New Yorkista.

2.2 Unity 3D-sovellus liikkuvan maailman pohjana

Unity on yhtiön omien sanojen mukaisesti monen alustan pelinkehitystyökalu, joka on suunniteltu alusta asti helppokäyttöiseksi. Unityssa on muun muassa integroitu editori, jossa peliä on helppo ja nopea testata luomisen ohessa. Ohjelmassa on myös tehokas grafiikkaliukuhihna, joka sisältää Direct X- ja OpenGL-renderin.

Unity 3D tukee monia eri ohjelmia, esimerkiksi Tampereen Ammattikorkeakoululla käytössä olevia Adobe Photoshopia sekä Cinema 4D -3D-mallinnusohjelmaa, joiden avulla VirtuaaliViipurin rakennuksien malleja ja tekstuureja pystyi käsittelemään ja viemään Unity 3D:hen.

Unity 3D:lla on mahdollista luoda pelejä tai muita interaktiivisia tuotteita Mac OS X:lle, Windows 2000:lle ja uudemmille Windows-käyttöjärjestelmille sekä WWW-selaimille. Erityisesti WWW-selaimille materiaalin luominen houkutteli tutkimaan Unity 3D:n mahdollisuuksia VirtuaaliViipurin sisällön luomisessa. (Unity Technologies, 2010, A.)

3 3D-grafiikka

Kolmiulotteinen, eli 3D-grafiikka on tietotekniikassa jo itsessään oksymoron, sillä käytännössä kaikki kolmiulotteinen materiaali projisoidaan kaksiulotteiselle kuvapinnalle. Käyttäjälle luodaan siis vain vaikutelma kolmiulotteisuudesta. Tämä saadaan aikaan esimerkiksi erilaisten varjostusten ja pintojen heijastusten avulla.

Kolmiulotteisen grafiikan peruskoordinaatit ovat kaikille tutut x, y, z eli leveys, korkeus ja syvyys. Mallintaessa näiden avulla määritellään mallin rakenne ja kuvaa ruudulla esittäessä määritellään niiden avulla, mitä kussakin ruudun pisteessä sijaitsee. Kuvaa esittäessä z-koordinaatti kertoo sen, mikä ruudulle piirrettävistä objekteista esitetään päällimmäisenä. Kolmiulotteista grafiikka on pääsääntöisesti vektorigrafiikkaa ja sen pääelementteinä ovat kolmiot tai muut monikulmiot. Yksittäisistä kolmioista puhuttaessa käytetään usein termiä polygoni.

Kolmiulotteisen grafiikan piirtämisestä ruudulle käytetään nimitystä renderöinti. Renderöinti voi tapahtua joko etukäteen tai reaaliaikaisesti. Reaaliaikaisesta renderöinnistä puhuttaessa käytetään yleisesti termiä reaaliaikainen grafiikka. Etukäteen renderöityjä, eli esi-renderöityjä, kuvia hyödynnetään sellaisenaan tai kolmiulotteisia animaatioita luodessa. Tällaisia animaatioita käytetään esimerkiksi elokuvissa, kun taas reaaliaikaista grafiikkaa näkee peleissä tai vaikkapa joka kesä Helsingissä järjestettävässä Assembly-tapahtumassa esiteltävissä epäinteraktiivissa demoissa ja introissa. Kuvassa 3 on kuvankaappaus Assembly Summer 2010 -tapahtuman demo-kilpailun voittaneesta työstä. (Assembly Organizing, 2010; Puhakka, 2008, 24 – 27)

Kuva 3: Voittajateos, Andromeda Software Developmentin Happiness is around the bend

3.1 Reaaliaikainen grafiikka

Reaaliaikaisella grafiikalla tarkoitetaan reaaliajassa kuvaruudulle piirrettävää materiaalia. Reaaliaikaisesta grafiikasta voidaan puhua jo silloin, kun sekunnin aikana ruudulle pystytään piirtämään viisi kuvaa. Käytännössä ohjelmistokehittäjät pyrkivät kuitenkin saamaan ohjelman toimimaan siten, että se piirtää vähintään 30 tai 60 kuvaa sekunnissa. Mitä nopeammin kuvia pystytään ruudulle tuottamaan, sitä paremmin käyttäjä pystyy reagoimaan kuvan muutoksiin, ja sitä sulavimmilta kuvassa tapahtuvat muutokset näyttävät. Samalla myös käyttäjän tekemät muutokset kuvaan tapahtuvat nopeammin. Yli 72 kuvaa ruudulle piirtäessä erot kuvataajuuksien välillä ovat jo melko vaikeita hahmottaa.

Reaaliaikaisen grafiikan vastakohtana on elokuvateollisuudessa käytetty esirenderöity grafiikka, jossa on voitu käyttää laskennallisesti hyvin vaativia menetelmiä, jolloin 3D-grafiikasta on saatu hyvin näyttävää. Reaaliaikaista grafiikkaa käytetään pääasiassa peleissä, mutta myös esimerkiksi demoissa eli reaaliajassa pyörivissä, epäinteraktiivisissa graafisissa esityksissä. Koska peleissä vaaditaan interaktiivisuutta, ja käyttäjällä täytyy olla mahdollisuus vaikuttaa kuvan muodostamiseen, ei kuvia voida renderöidä etukäteen.

Reaaliaikaisessa grafiikassa on käytettävä melko yksinkertaisia algoritmeja grafiikan piirtämiseen, koska kuva täytyy piirtää ruudulle sekunnin murto-osassa. Konetehon kas-

vaessa voidaan käyttää yhä haastavampia menetelmiä entistä näyttävämmän grafiikan tuottamiseen. Reaaliaikaisen grafiikan parissa joutuukin tasapainottelemaan prosessointitehokkuuden ja grafiikan näyttävyyden välillä.

Pelialan räjähdysmäinen kasvu 70- ja 80-luvuilta ajaa eteenpäin myös reaaliaikaisen grafiikan tekniikoiden ja 3D-grafiikan käyttämän laitteiston kehitystä. Monien mielestä reaaliaikaisen grafiikan aikakausi alkoi vuonna 1996 3Dfx:n Voodoo 1 -näytönohjaimen ilmestyttyä kuluttajamarkkinoille. Nykyään näytönohjain on standardi, joka löytyy jokaisesta kotikoneesta. (Akenine-Möller, Haines & Hoffman, 2008, 1; Puhakka, 2008, 24)

3.2 Reaaliaikaisen grafiikan liukuhihna – kuvan muodostaminen näytölle

3D-piirron liukuhihnalla tarkoitetaan kuvan prosessoinnin eri vaiheita raakadatasta aina ruudulle piirretyksi kuvaksi. Korkealla tasolla ajatuksena on se, että pelissä tai muussa ohjelmasta materiaalia siirretään API:lle (Direct 3D tai OpenGL), sieltä laiteohjaimelle ja laiteohjaimelta 3D-kiihdyttimelle, eli näytönohjaimelle.

Liukuhihnan päätehtävänä on luoda eli renderoida virtuaalisella kameralla kaksiulotteista kuvaa kolmiulotteisista objekteista, valonlähteistä, varjostuksista ja tekstuureista. Tuotetussa kuvassa objektien sijainnit ja muodot määrittyvät geometrian, ympäristön piirteiden ja kameran sijoitus ympäristössä. Objektien ulkoasuun vaikuttavat materiaalin ominaisuudet, valonlähteet, tekstuurit ja varjostukset.

Esimerkkinä käytetyn reaaliaikaisen grafiikan 3D-piirron liukuhihnan päävaiheet ovat:
- applikaatiovaihe (Application)
- geometriavaihe (Geometry)
- rasterointi- tai renderöintivaihe (Rasterizer)

Applikaatiovaiheessa käyttäjällä on täysi kontrolli siihen, mitä tapahtuu, koska applikaatiotaso suoritetaan prosessorilla. Tästä johtuen ohjelman kehittäjä voi muokata ohjelmaa, jotta se toimisi mahdollisimman tehokkaasti. Esimerkiksi käyttäjä voi 3D-

mallinnussovelluksessa pyörittää mallintamaansa objektia haluttuun asentoon tai muuta vastaavaa. Applikaatiotason lopussa renderöitävä geometria syötetään geometriatasolle.

Geometriavaiheen ensimmäisessä vaiheessa (**Model & View Transform**) määritellään, missä malli sijaitsee, missä asennossa se on, ja minkä kokoinen se on. Myös renderöitävän kuvan muodostavan kameran sijainti ja suunta määritellään tällä tasolla. Renderöitäväksi menevät vain ne mallit, jotka näkyvät kameralle

Toisessa vaiheessa (**Vertex Shading**) määritellään erilaisia materiaali ja varjostusasetuksia, joilla on suuri rooli näyttävän grafiikan luomisessa. Varjostukset ja muut ulkoasuun vaikuttavat asiat asetetaan kerralla yksittäiselle verteksille.

Kun ensimmäisessä vaiheessa määriteltiin kameran sijainti ja suunta, pitää kolmannessa vaiheessa (**Projection**) määritellä tapa, jolla kuva muodostetaan. Yleisesti käytössä olevia kuvaustyylejä on kaksi; ortografinen- ja perspektiiviprojektio.

Neljännessä vaiheessa (**Clipping**) optimoidaan ruudulle piirrettäviä objekteja. Vain objektit, jotka näkyvät kameralle joko kokonaan tai osittain, piirretään ruudulle. Käyttäjä voi myös itse vaikuttaa siihen, jätetäänkö jotain osia piirtämättä ruudulle.

Geometriatason viidennessä ja viimeisessä vaiheessa (**Screen Mapping**) muutetaan edellisessä vaiheessa piirrettäväksi valittujen objektien koordinaatit ruudulle piirtämistä varten.

Rasterointi- tai renderöintivaiheessa kaikki primitiivit rasteroidaan, eli muutetaan pikseleiksi ikkunaan. Jokaisen objektin jokainen näkyvä viiva ja kolmio siirtyy liukuhihnalla eteenpäin. Ne kolmiot, joille on määritelty tekstuuri, renderöidään tekstuurin kanssa. Näkyvyys määritellään Z-puskurin syvyys-testillä, mahdollisten alpha- ja siluettipuskuritestien kera. Jokainen objekti prosessoidaan vuorollaan ja kirjoitetaan videopuskuriin. Lopuksi valmis kuva esitetään ruudulla.
(Akenine-Möller ym., 2008, 11 – 27; Puhakka, 2008, 163 – 169)

4 Reaaliaikaisen 3D:n kehitysaskelia pelien näkökulmasta

Reaaliaikainen grafiikka ja pelit ovat tieteen ja taiteen saralla jokseenkin nuori ala. Kuitenkin kehitystahti on ollut huimaava. Vuorovedoin graafinen kehitys on luonut uusia vaatimuksia tietokoneen komponenteille ja laitteiston kehitys on avannut uusia mahdollisuuksia pelien tekijöille kokeilla kykyjään ja luoda uusia tekniikoita näyttävän elämyksen luomiseksi.

Edge-lehti keräsi vuoden 2008 artikkelissaan verkkosivuilleen pelien reaaliaikaista grafiikkaa mullistaneita tekniikoita. Tätä artikkelia lainaten käyn läpi joitakin tärkeimmistä ja nykyajan reaaliaikaisessa grafiikassa merkityksellisimmistä teknisistä edistysaskelista.

4.1 Rautalanka 3D ja täytetyt polygonit

Rautalanka-3D lisäsi vektori-grafiikkaan z-akselin koordinaatistoon. Periaatteiltaan vektorigrafiikka ja rautalanka-grafiikka ovat hyvin lähellä toisiaan. Ensimmäinen rautalanka-3D-peli oli Atarin Battlezone.

Värinäyttöjen tultua markkinoilla alettiin peleissä käyttämään täytettyjä polygoneja. Tällöin ruutu käytiin läpi ja polygonit täytettiin väreillä pikseli kerrallaan. Ensimmäinen täytettyjä 3D-polygoneja ja kamerakontrolleja käyttänyt peli oli Atarin I, Robot. (Edge Staff, 2008)

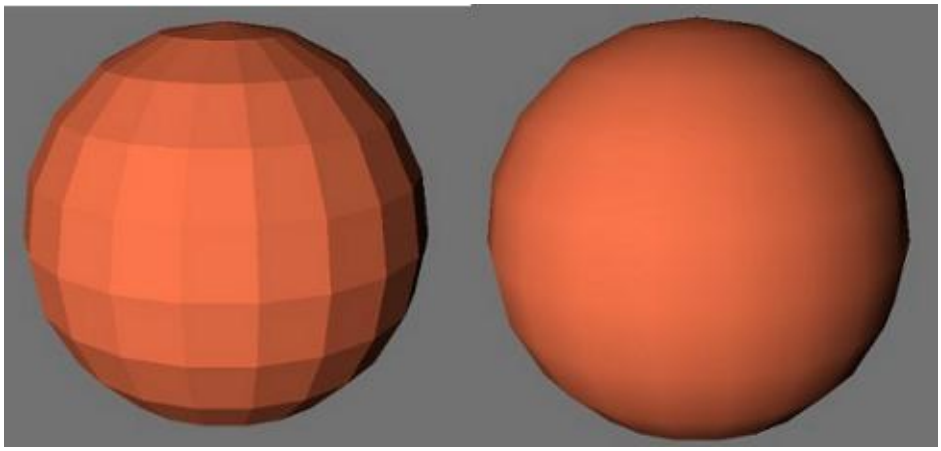
4.2 Gouraud- ja Phong-varjostus

Reaaliaikaisessa 3D-grafiikassa **tasasävytys (Flat Shading)** tuo ongelmia, mikäli kappaleen halutaan näyttävän pyöreältä, koska usein tehonsäästösyistä kappaleet joudutaan tekemään melko yksinkertaisella rakenteella. Tästä syystä tasasävytys saa aikaan sen, että yksittäiset monikulmiot tulevat esiin aivan liian selkeästi.

Tätä ongelmaa korjaamaan kehiteltiin Gouraud-varjostus. Vuonna 1971 esitellyssä, Henri Gouraudin kehittämässä, Gouraud-varjostus monikulmioista koostuvaan malliin saadaan kaartuvan pinnan vaikutelma interpoloimalla väri monikulmioiden kulmista. (Puhakka, 2008, 26)

Gouraud-varjostuksen ongelmana oli, että se lasketaan fyysisen näytön mukaan, eikä 3D-maailman, jota näytöllä yritetään esittää, mukaisesti. Tämä prosessi aiheuttaa interpolointi-virheitä. Phong-varjostus eroaa Gouraud-varjostuksesta siinä, että polygonien sisäosien pisteille lasketaan oma varjostus. Gouraud-varjostuksessa sisäosien pisteille laskettava varjostus johdetaan lineaarisesti kulmapisteistä. Suurten polygon-määrien malleissa phong- ja gouraud-varjostusten ero on kuitenkin merkityksetön. Vuonna 1973 esitelty Phong-varjostus on nykyään yleisesti käytetty menetelmä. (Akenine-Möller ym., 2008, 115 – 116)

Olen kuvaa 4 varten mallintanut pallon. Kuvassa vasemmalla näkyy kyseinen pallo tasasävytyksellä ja kuvassa oikealla sama pallo phong-varjostettuna.



Kuva 4: Varjostustekniikoiden kehitysaskeleet

4.3 Teksturointi eli pintakuviointi eli tekstuurimappaus (Texturing)

Yksinkertaisimmillaan teksturoinnissa 3D-mallin pinnalle lisätään bittikarttakuva. Nykyään teksturointi kulkee käsi kädessä mallinnuksen rakenteellisen laadun kanssa. Tiettyyn tarpeeseen voidaan tehdä malli, joka on joko todella näyttävä tai vaihtoehtoisesti

sellainen, joka on riittävän tarkka, mutta koneelle kevyt ja nopea piirtää ruudulle. Mallin rakenteen kannalta tämä tarkoittaa tietysti sitä, että mallissa on joko paljon tai vähän polygoneja. Tekstuurien osalla voidaan valita tekstuurin resoluutio.

Kuvassa 5 on mallintamani pelihahmo, jolle olen luonut tekstuurikartan. Tekstuurikartta on tässä tapauksessa .psd-muodossa, jolloin sitä on helppo käsitellä kuvankäsittelyohjelmilla. Esimerkiksi peliin luodun hahmon tekstuurin lopullinen versio kannattaisi tallentaa yhtä tasoa käyttävään tiedostoformaattiin ja käyttötarkoituksesta riippuen pakata. Kuvassa vasemmalla on malli ilman tekstuuria ja oikealla sama malli, johon on lisätty luomani tekstuuri. (Akenine-Möller ym., 2008, 147 – 148)

Kuva 5: Vasemmalla 3D-malli ilman tekstuuria ja oikealla sama malli tekstuurilla

4.4 MIP-kartta – *Multum in Parvo* – ”paljon vähässä”

Yksinkertainen teksturointi korkean resoluution tekstuurien kanssa käyttää paljon muistia ja prosessoritehoa. Tätä ongelmaa korjaamaan kehitettiin MIP-kartta, englanniksi MIP MAP. MIP-kartta tallentaa jokaisesta tekstuurimapista sarjan kevyemmäksi skaalattuja karttoja, jokaisen kartan ollen puolet aiemman koosta. Kun tekstuuri lisätään polygoniin, lasketaan pikselin etäisyys kameraan, jonka mukaan valitaan käytettävä tekstuurin versio. MIP-kartassa alkuperäisen tekstuurin mukana talletetaan pienemmät teks-

tuuriversiot, jolloin tilaa tarvitaan vain 4/3 alkuperäisestä. Unity 3D:ssä MIP-kartat generoidaan Texture Inspectorissa. (Puhakka, 2008, 211 – 214; Unity Technologies, 2010, B.)

4.5 Z-Puskuri eli syvyyskartta (Z-Buffer)

70-luvulla kehitetty Z-Puskurointitekniikka käsittelee 3D-kuvan syvyyttä. Z-puskurointi voidaan toteuttaa niin laitteistolla kuin ohjelmallisestikin. Z-puskurointitekniikalla ratkaistiin ongelma, jossa piti kyetä valitsemaan, mitkä elementit ruudulla ovat näkyviä, ja mitkä piilossa. Z-puskuri käyttää nimensä mukaisesti hyväkseen koordinaatiston z-akselia, eli syvyyttä. Kun 3D-malli on asetettu sijaintiinsa, annetaan sille syvyysarvo. Mikäli kaksi tai useampi objekti osuu samaan pikseliin lopullisen piirron yhteydessä, tarkastetaan z-puskurista, mikä objekteista on lähempänä kameraa ja vain se piirretään. (Puhakka, 2008, 225 – 227)

4.6 Ympäristökuvaustekniikka (Environment Mapping)

Myös ympäristökuvauksen toteuttamiseen on useita keinoja. Yksi yleisimmin käytetyistä tekniikoista lienee kuutiomappaus. Kuutiomappauksessa kuution keskeltä otetaan kuusi kuvaa, yksi jokaista kuution tahkoa kohti. Erityisesti ympäristömappauksessa voidaan ympäristöstä ja käyttötarkoituksesta riippuen tasapainotella laskentatehokkuuden ja laadun välillä. Ympäristömappauksen laadusta voidaan tinkiä, mikäli ympäristö on hyvin yksinkertainen, vaihtuu tiivistä tahtia, tai kun se on molempia, kuten esimerkiksi autopeleissä. (Akenine-Möller ym., 2008, 297 – 308; Puhakka, 2008, 219 – 222)

Kuvassa 6 esitetystä käytävästä on luotu ympäristökartta, ja se esitetään heijastuksena pelattavan hahmon aseen tähtäimen takaosasta.

Kuva 6: Call of Duty 4: Modern Warfare -pelin pelihahmot ovat asemissa käytävällä (Activision, 2007)

4.7 Tekstuurien suodatus

Bilinear Filtering on yleisesti käytetty tapa suodattaa tekstuureja. Pikselien väriarvo lasketaan painotetusti sitä ympäröivien tekselien (tekstuurin pikseli) väriarvojen ja etäisyyden perusteella. Suodatuksen avulla tekstuurit eivät näytä karkeilta, kun niitä tarkastellaan läheltä. Bilinear Filtering -tekniikkaa käytetään lähes poikkeuksetta MIP-kartta -tekniikan kanssa. (Puhakka, 2008, 209)

Trilinear Filtering on itse asiassa sama suodatustapa kuin Bilinear Filtering, mutta tekniikassa suoritetaan lisäksi lineaarista interpolointia MIP-karttojen välillä. (Cross, 2007)

Anisotrooppisessa suodatuksessa tekstuurien suodatuksen tarkkuus riippuu siitä, missä kulmassa tekstuurin pinta on kameraan nähden. Mitä terävämpi kulma on, sitä enemmän tekstuurista otetaan näytteitä. (Puhakka, 2008, 213)

Kuvan 7 ruudunkaappauksissa vasemmalla bilineaarinen suodatus, keskellä trilineaarinen suodatus ja oikealla trilineaarinen sekä anisotrooppinen suodatus. Kuvan tiiliseinän

yksittäiset tiilet erottuvat trilineaarisen ja anisotrooppisen suodatuksen yhteiskäytössä parhaiten.

Unity 3D:ssä tekstuurien suodatukset asetetaan Texture Inspectorissa. (Unity Technologies, 2010, B.)

Kuva 7: Infinity Ward -pelikehittäjän näkemys tiiliseinästä (Activision, 2007)

4.8 Antialiasointi

Koska reaaliaikaisessa grafiikassa pyöritellään objekteja ja niitä esitetään eri kulmista, on niissä usein porrastusta ja kulmikkuutta linjoissa, jotka eivät kulje tarkan vertikaalisesti tai horisontaalisesti. Tämä reunojen porrastus johtuu siitä, että käytetyn resoluution tarkkuus ei ole riittävä esittämään kuvan yksityiskohtia tarkasti ja tästä käytetään termiä aliasoituminen. Antialiasointi on luonnollisesti kehitetty poistamaan tätä porrastusta. (Ryynänen, 2010)

Kuvassa 8 näkyvät aliasoitumisen ja antialiasoinnin vaikutukset. Esimerkiksi kiväärin takatähtäimen sahalaitaisuus erottuu selkeästi ennen antialiasointia. Kuva on pelistä Call of Duty 4: Modern Warfare.

Unity 3D:ssä antialiasoinnin käyttöä säädellään Quality Settings-valikosta käsin. (Unity Technologies, 2010, C.)

Kuva 8: Vasemmalla on antialiasointi poissa käytöstä ja oikealla nelinkertainen antialiasointi (Activision, 2007)

4.9 Globaali valaistus (Global Illumination)

Globaali valaistus -tekniikalla mallinnetaan valon poukkoilemista objekteissa. Ennen globaalia valaistusta reaaliaikaisessa grafiikassa on käytetty ambienttia valoa, mutta koska objektit eivät oikeasti luo itse valoa, tarjoaa globaali valaistus realistisemman vaihtoehdon.

Käytännössä reaaliaikaisessa grafiikassa on hyvin hankalaa mallintaa varsinaisesti valon poukkoilua objekteista toiseen ruudunpäivitysnopeutta pudottamatta, valojen reaaliaikaisen mallintamisen ollessa erittäin raskasta.

Usein käytetty ratkaisu on laskea globaalin valaistuksen vaikutukset etukäteen. Käytännössä siis globaali valaistus laskee valon poukkoilun arvoa ja lisää ne tekstuureihin, jotka otetaan huomioon viimeisiä väriarvoja määrittäessä reaaliaikaisen grafiikan renderöinnissä. Tätä tekniikkaa käytetään etenkin konsolipeleissä, konsolien tehorajoitusten vuoksi. (Puhakka, 2008, 405 – 424; Akenine-Möller ym., 2008, 327 – 438)

4.10 Reaaliaikaiset varjot (Realtime shadows)

Niin 2D- kuin 3D-peleissäkin ovat valot ja varjot korvaamaton väline etäisyyksien hahmottamiseksi. Reaaliaikaisen varjostuksen toteuttamiseen on useita eri keinoja. Yksi käytetyimmistä keinoista on **Shadow Mapping**. Reaaliaikaisia varjoja luotaessa määritellään valon lähtöpiste ja suunta ja tämän jälkeen z-puskuria hyväksi käyttäen määritellään mikä kaikki näkyy valolle. Määrittelyn tuloksena luodaan Shadow Map, joka kertoo, mitä kaikkea kuvasta valaistaan, ja mitä jätetään varjoon. Tämän jälkeen pelaajan näkymä renderöidään ja siinä voidaan ottaa huomioon valo itsessään sekä sen aiheuttamat varjot. Valon liikkuessa Shadow Map täytyy päivittää. Liikkuvia valoja ovat esimerkiksi taskulamput, autojen valot, heiluvat kattovalaisimet. Myös valonlähteiden edessä liikkuvat, varjoja aiheuttavat objektit aiheuttavat sen, että Shadow Map täytyy päivittää. (Akenine-Möller ym., 2008, 348 – 372)

Unity 3D:ssä varjojen ominaisuudet määritellään Quality Settings-valikon kautta. Uuden 3-version myötä Unity 3D tukee myös Lightmappingia. Lightmappingin avulla voidaan määritellä pintojen kirkkausarvoja ja sen myötä pintojen eräänlainen kimmellys vähenee ja valaistuksesta tulee realistisempi. (Unity Technologies, 2010, C. & D.)

Kuvassa 4 pelihahmot ovat kulkeneet käytävällä sijaitsevan lampun eteen. Tästä johtuen pelihahmot synnyttävät varjoja toisiinsa ja seiniin.

4.11 Level of Detail

Level of Detail tai LOD tarkoittaa sitä, että 3D-mallin rakennetta kevennetään, kun sitä viedään kauemmas pelaajan näkökentästä. Heikennykseen vaikuttavat muun muassa 3D-mallin tärkeys ja sijainti. Pelaajalle mallin heikentyminen ei useinkaan välity, koska malli on joko kaukana tai liikkuu hyvin nopeasti. Kun kauempana esiintyviä malleja kevennetään, vähentää se samalla koneelle asetettavaa taakkaa ja pelin ruudunpäivitys paranee. (Luebke, Reddy, Cohen, Varshney, Watson & Huebner, 2003, 3 – 6)

Mikäli Level of Detailia halutaan käyttää Unity 3D:ssä, täytyy sitä varten kirjoittaa skripti, jossa määritellään, miltä etäisyydeltä mallien rakennetta aletaan pudottamaan. Mallien rakenteen keventäminen voidaan tehdä joko skriptissä tai vaihtoehtoisesti teke-

mällä toinen, kevyempi malli. Skriptaamalla kevennystä ei tarvitse tehdä erikseen käsin jokaisen yksittäisen mallin osalta, mutta mallin rakenne saattaa kärsiä.

Kuvassa 9 näkyy, miten Call of Duty 4: Modern Warfare -pelissä taustalla olevan sotilaan mallia on laskettu automaattisesti yksinkertaisemmaksi Level of Detail-tekniikan avulla.

Kuva 9: Käytännön esimerkki Level of Detail -tekniikan käytöstä. (Activision, 2007)

4.12 Jälkikäsittely (Post-processing)

Esimerkiksi **bloom-**, **depth of field-** ja **blur**-efekti ovat jälkikäsittelyssä tehtäviä asioita. Nykyaikainen grafiikkaprosessori kykenee muuttamaan kuvan välimuistissa sijaitsevat materiaalit tekstuuriksi missä tahansa yksittäisen kuvan renderöinnin aikana. Näitä tekstuureja hyödyntämällä saadaan aikaan erilaisia realismia tehostavia efektejä.

4.12.1 High Dynamic Range Rendering

Tavallisessa valokuvauksessa, mutta myös reaaliaikaisessa grafiikassa, on ollut ongelmia sellaisissa kuvissa, joissa on voimakkaita varjostuksia ja voimakasta valoa. Yleinen esimerkki HDR:n vaikutuksista on kuva, jossa on varjoisa vuorenseinämä ja kirkkaasti paistava aurinko. Tällöin on täytynyt päättää, halutaanko ottaa kuvaa, jossa auringon värit ja valoisuus ovat sopivat, mutta varjostetut alueet jäävät todella synkiksi. Vaihtoehtoisesti on otettu kuva, jossa aurinko on todella ylivalottunut, mutta varjoisat tummat osat kuvasta ovat selkeitä.

HDR-tekniikka korjaa nämä ongelmat siten, että kuvia otetaan kolmella eri valotusajalla siten, että kolmesta kuvasta yksi on todella synkkä, yksi ylivalottunut ja yksi näiden välistä. Nämä yhdistämällä saadaan yksi valokuva, jossa kaikki osat kuvasta on pystytty esittämään selkeästi. Reaaliaikaisessa grafiikassa valotusajan sijaan kuvan muokkaus tapahtuu kontrastiarvojen avulla. Koska ihmissilmässä on monin kerroin parempi kontrastin erotuskyky kuin parhaimmissakin näytöissä, käytetään HDR-renderöinnissäkin useampia kontrastiarvoja, kuin käytännössä pystytään esittämään. Tämä tarkoittaa sitä, että kontrastiarvot joudutaan säätämään lopuksi näyttöön soveltuviksi.

Eri HDR-tekniikoissa on kuitenkin hyvät ja huonot puolensa, esimerkiksi Elder Scrolls IV: Oblivion -pelissä käytetyn HDR-tekniikan kanssa ei alun perin ollut mahdollista käyttää samanaikaisesti antialiasointia. Lisäksi tekniikat eroavat muun muassa muistinkäytön ja prosessointinopeuden osalta. (Edge Staff, 2008; McTaggart, 2007)

Unity 3D mahdollistaa tehokkuutensa puolesta HDR-valaistuksen käyttämisen, mutta koska ohjelman mukana ei tule valmista HDR-valaisukomponenttia, pitäisi se ohjelmoida erikseen.

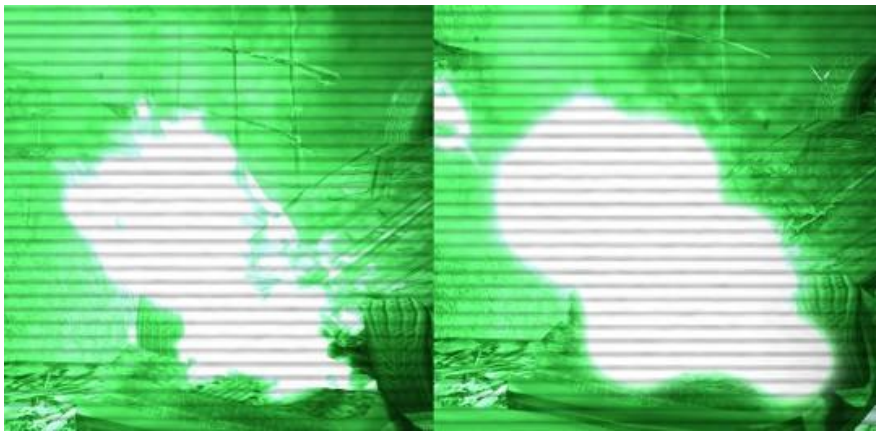
4.12.2 Bloom

Bloom-varjostusefektistä käytetään joskus myös nimityksiä glow ja light bloom. Bloom on osa HDR-tekniikkaa. Se sisältää efektin, jossa kirkkaasti valaistut objektit näyttävät luovat hehkua ympärilleen. Hehkua esittävä efekti saadaan aikaan luomalla 3D-scenestä kuvasta tekstuuri, jossa on vain hehkua tuottavat valot. Tekstuurin valoja sekoitetaan ja

laajennetaan. Seuraavaksi tekstuuri liitetään kuvaan, josta tekstuuri alun perin luotiin ja lopuksi yhdistetty kuva esitetään näytöllä. Pelkän bloomin etuna laajempiin HDR -efekteihin verrattuna on nopeus ja vähäisempi muistinkäyttö.

Kuvassa 10 on kaksi kuvankaappausta pelistä Call of Duty 4: Modern Warfare. Kuvassa on palava rengas, jossa liekit on muodostettu tekstuurien, partikkelien ja valojen yhdistelmällä. Vasemmanpuoleisessa kuvankaappauksessa liekit ovat normaalit. Oikeanpuoleisessa kuvankaappauksessa on otettu käyttöön bloom-efekti, joka saa liekit luomaan ympärilleen hehkua. (O'Rorke & James, 2004; McTaggart, 2007)

Unity 3D:n 3-versio toi mukanaan uusia efektejä, muun muassa Bloomin. Bloomin käyttö vaatii Unity Pro –lisenssin. (Unity Technologies, 2010, E.)



Kuva 10: Bloom-efektillä saadaan aikaan hehkua (Activision, 2007)

4.12.3 Blur

Reaaliaikaisessa grafiikassa on monia syitä, miksi ruutua haluttaisiin sumentaa. Yksi hyvä esimerkki on tilanne, kun pelihahmo kääntyy ympäri, sekoitetaan esitettävää kuvaa sen verran, että käyttäjälle välittyy tuntemus liikkumisesta. Ruutua sekoitetaan kameran liikkeestä vastakkaiseen suuntaan.

Valve Softwaren testeissä ruudun sekoittamisen huomattiin aiheuttavan testihenkilöille pahoinvointia. Pahoinvointia aiheutui myös sen takia, että sekoituksen käyttäminen laskee sekoituksen ajaksi ruudunpäivitystä. Tästä syystä ruudun sekoittaminen kannattaa pitää mahdollisimman pienenä, vain muutamissa prosenteissa. (Vlachos, 2008)

Kuvassa 11 on kuvankaappaus pelin Call of Duty 4: Modern Warfare alkupuolen kohtauksesta. Kohtauksessa on kaltoin kohdeltu mies, jonka näkö on uupumuksen takia sumea. Tästä syystä kaikki kauempana näkyvä on sumennettu

Myös Blur-efekti löytyy Unity Pro -lisenssin mukana tulevien assettien mukana. (Unity Technologies, 2010, F.)

Kuva 11: Blur-efektiä on hyödynnetty kuvankaappauksessa kaikkeen, pelihahmon edessä sijaitsevaa videokameraa lukuun ottamatta (Activision, 2007)

4.12.4 Depth of Field

Depth of Field on tekniikka, jossa luodaan vaikutelmaa syvyysnäöstä sumentamalla vain sitä osaa ruudusta, jota ei ole tarkennettu. Esimerkiksi sotapeleissä, kun pelaaja tähtää rautatähtäimen läpi, näkyy tähtäin selkeästi ja terävänä, koska pelihahmo oletetusti tarkentaa katseensa tähtäimeen. Z-puskuria hyväksi käyttävän syvyys-testin avulla selvitetään, mitkä objektit ovat pelihahmon lähellä. Näihin lähellä oleviin objekteihin sovelletaan Blur-efektiä. (Evans, 2001)

Kuvassa 12 on kaksi kuvankaappausta pelistä Call of Duty 4: Modern Warfare. Ylemmässä kuvankaappauksessa Depth of Field on käytössä ja alemmassa se on poissa käy-

töstä. Kuvassa esiintyvä autonraato on riittävän lähellä pelihahmoa, jolloin Depth of Field -efektiä sovelletaan siihen.

Unity 3D:n 3-version Pro assettien mukana tuli myös Depth of Fieldin mahdollistava skripti. (Unity Technologies, 2010, G.)

Kuva 12: Depth of Field -tekniikan käytännön vaikutus (Activision, 2007)

4.13 Tekniikoista lopuksi

Jokainen edellä mainituista tekniikoista on merkittävästi kehittänyt reaaliaikaisen grafiikan näyttävyyttä ja/tai realistisuutta. Esimerkiksi Depth of Field -tekniikka ei varsinaisesti ole teknisesti erityisen mullistava. Kuitenkin tapa, jolla sen avulla imitoidaan oikeaa maailmaa, on merkittävä lisä realistisen kohtauksen luomisessa.

Kuvassa 13 on kuvankaappaus pelistä Call of Duty 4: Modern Warfare. Kuvassa on käytössä muun muassa Trilinear Filtering, Depth of Field, Glow, anistrooppinen suodatus ja korkean resoluution tekstuurit.

Kuva 13: Kaikkia tekniikoita käyttämällä saadaan aikaan näyttävä kuva (Activision, 2007)

6 VirtuaaliViipuri – malleista liikkuvaksi maailmaksi

Ensimmäinen asia oli selvittää, miten insinöörit olivat luoneet rakennuksia ja tutustua sovelluksiin, joita he ovat käyttäneet ja sovelluksiin, joita minun tulisi käyttää. Eli toisin sanoen kävin läpi Archicadin, Unity 3D:n ja Cinema 4D:n toiminnallisuuksia. Lisäksi tutustuin siihen, minkälaisia Archicadilla luotujen mallien rakenteet olivat. Lopuksi testasin miten mallit toimivat käytännössä, Unity 3D:n pelimoottorissa. Testasin mallien toimivuutta sekä isojen että pienempien kokonaisuuksien kanssa.

6.1 Insinöörien mallit ja Archicad-sovellus

Archicad on Graphisoftin valmistama sovellus, jolla arkkitehdit voivat ”simuloida” rakennusta. Periaatteessa arkkitehdit voivat tehdä normaaliin tapaan piirustukset ja Archicad luo sen perusteella 3D-mallin, jota voidaan esitellä asiakkaalle.

VirtuaaliViipuri-projektissa insinööriopiskelijat ovat luoneet vuoden 1939 Viipurin rakennuksia kyseisellä sovelluksella ja näitä rakennusten malleja tulisi käyttää reaaliaikaisen, liikkuvan maailman sisältönä. Ristiriitaa kuitenkin aiheuttaa se, että Archicad-sovellusta ei ole luotu reaaliaikaista grafiikkaa silmällä pitäen, vaan kuten selvitystä tehdessäni saatoin todeta, luo Archicad jokseenkin raskaita malleja. Näitä malleja tulisi pystyä keventämään merkittävästi, jotta ne olisivat käyttökelpoisia reaaliaikaisessa grafiikkasovelluksessa. Koska kuten aiemmissa kappaleissa on todettu, erilaisia malleille tehtäviä efektejä lasketaan polygon kerrallaan, tulee Archicad-malleista vaikeita käsitellä reaaliaikaisessa grafiikassa.

Myös mallien rakenne aiheuttaa ongelmia. Koska malleissa on todella paljon päällekkäisiä objekteja, aiheuttaa tämä reaaliaikaisessa grafiikassa ongelmia. Päällekkäisillä objekteilla on sama z-puskuri-arvo ja tällöin tietokone ei osaa päättää, kumpi objekteista pitäisi esittää ylempänä. Tämä aiheuttaa ruudulle rumaa grafiikkavirhettä, kun kumpaakin objektia yritetään esittää samaan aikaan samassa paikassa.

Lisäksi mallinnetuissa rakennuksissa jokainen osa on teksturoitu erikseen eri tekstuureilla, joka hankaloittaa materiaalien käsittelyä Unity 3D:ssä ja lisää tiedostojen määrää.

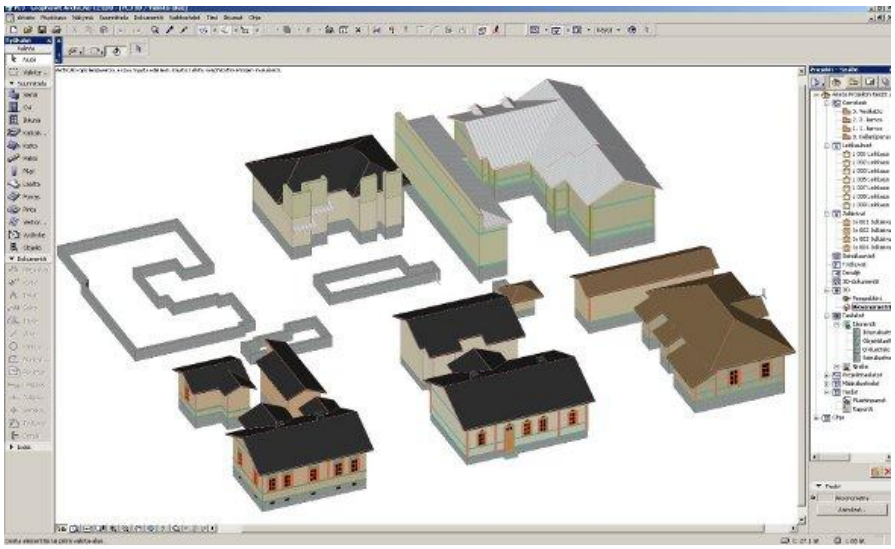
Nämä lisäävät vaadittua kovalevytilaa, tekstuurien asettelun työmäärää ja lopullisessa ohjelmassa vaadittua laskentatehoa.

6.2 Mallit Archicadista Cinema 4D:hen

Selvitystyöni alussa piti selvittää, mitkä Archicadin luomat tiedostomuodot ovat sellaisia, joita Maxonin Cinema 4D tukee. Cinema 4D:ssä malleja voitaisiin käsitellä ja keventää ja siirtää Unity 3D:hen. Tutkimustyön jälkeen Unity 3D on päivittynyt ja tukee tätä nykyä suoraan .c4d-tiedostomuotoa, jolloin Cinema 4D-vaihe olisi periaatteessa voitu sivuuttaa.

Archicadin käyttäminen ei ollut aivan helppoa koskaan aiemmin sovellusta käyttäneelle, koska kyseinen ohjelma sisältää hyvin paljon termistöä, joka on ohjelmaa itseään varten kehitetty. Ohjelman tutkimisen ja help-tiedostojen lukemisen jälkeen selvisi, että malli tulee saattaa 3D-näkymä -tilaan. Kuvassa 14 on kuvankaappaus Archicad-sovelluksesta, jossa on auki VirtuaaliViipurin kortteli 3D-näkymä -tilassa. 3D-näkymä -tilassa syntyivät ensimmäiset ongelmat. Osa testaamistani malleista sisälsi niin paljon polygoneja, että Archicad yksinkertaisesti kaatui tai vaihtoehtoisesti jumittui pitkiksi ajoiksi. Tästä johtuen osa rakennuksista on todella hankalia, jopa mahdottomia muuttaa sopiviin 3D-formaatteihin.

3D-näkymä -tilaan saattamisen jälkeen mallin voi tallentaa johonkin yleisistä 3D-sovellusten ymmärtämistä tiedostomuodoista, kuten esimerkiksi juuri Cinema 4D:n .c4d-tiedostomuotoon. Taustaselvityksessä kuitenkin selvisi, että mallien saattaminen Cinema 4D:n ymmärtämään muotoon järkevästi vaatisi erityisen .ac4d+-liitännäisen.



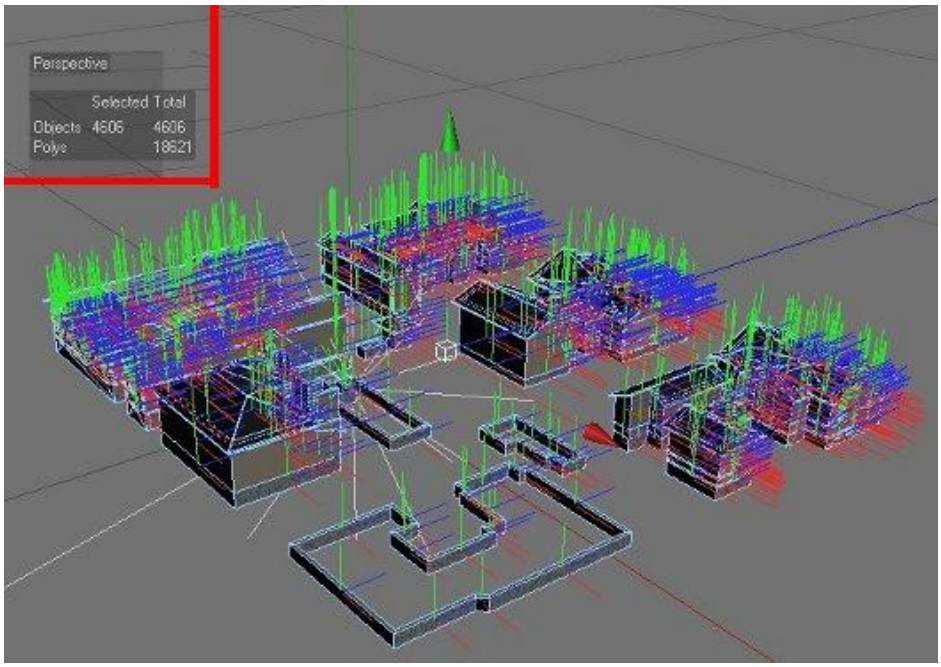
Kuva 14: Arcicad-sovelluksen 3D-näkymä VirtuaaliViipurin (ilmeisen keskeneräisestä) korttelista PL3

Vaihtoehtoja mallien saattamiseksi Cinema 4D:hen olivat .c4d, .ac4d, jossa asetuksista riippuen mallien rakenne muodostuu Archicadissa käytettyjen rakennusmateriaalien mukaan, tai jossa mallien rakenne muodostuu rakennusten osista sekä .ac4d+, jolla pystyi myös tallentamaan materiaalien tai rakennusten osien mukaisen mallin. Rakennusmateriaalien mukaisten mallien rakenne teki malleihin todella paljon yksittäisiä objekteja, kun taas rakennusten osien mukaan muodostetut mallit olivat yksinkertaisempia rakenteeltaan.

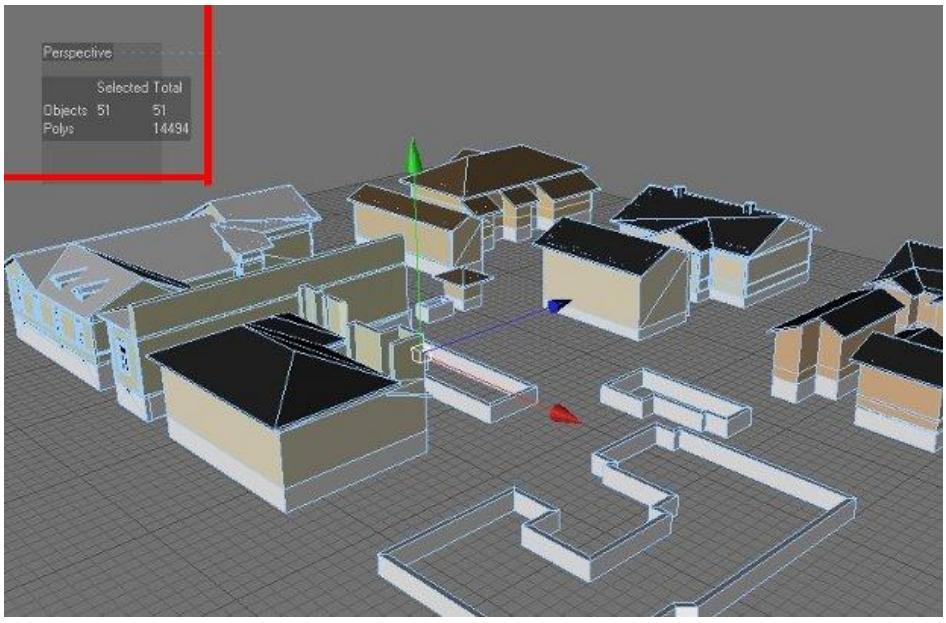
Mikäli insinöörien tekemissä malleissa olisi käytetty laajempia tekstuureja, olisi yksinkertaisen rakenteen mukainen malli ollut järkevämpi paremman käytettävyytensä ansiosta. Mutta koska insinöörit olivat kuitenkin asettaneet jokaiselle rakennusmateriaalille oman yksittäisen 3D-materiaalin tai tekstuurin, oli järkevämpää käyttää rakenteeltaan hankalampaa valintaa. Tällöin kuitenkin saavuttettiin mallin helppo käsittely Unity 3D:ssä, koska mallille saattoi asettaa valmiiksi luodut materiaalit tai tekstuurit suoraan rakennusmateriaalikohtaisille objekteille.

Koska Archicadin tai Maxonin ei ollut onnistunut luoda järkevää liitännäistä ohjelmien välillä toimimiseen, päätti unkarilainen, Dinnye-nimimerkillä internetissä toimiva ohjelmoija luoda oman AC4D+-liitännäisen. Kyseinen liitännäinen antaa käyttäjälle paremmat mahdollisuudet vaikuttaa mallien ominaisuuksiin, kun malleja siirretään Archicadista Cinema 4D:hen. Liitännäinen tekee myös merkittävästi kevyempiä ja järkevämpiä polygon-rakenteita malleille. (Dinnye, 2010)

Kuvasta 15 nähdään, että korttelin PL3 3D-mallin polygonien lukumäärä on valtaisa, 18621, kun malli on tuotu suoraan Archicadista .c4d-muodossa. Kuvassa 16 sama kortteli on tuotu Dinnyen luoman liitännäisen avulla ja rakenne on keventynyt merkittävästi. Polygon-määrä on tässä muodossa yli 4000 pienempi, kuin oletustavalla Cinema 4D:hen tuotuna.



Kuva 15: Korttelin PL3 polygonien lukumäärä on valtaisa .c4d-muodossa



Kuva 16: Kortteli PL3 kevyemmässä, .ac4d+-muodossa tuotuna

6.3 Mallit Cinema 4D:stä Unity 3D:hen

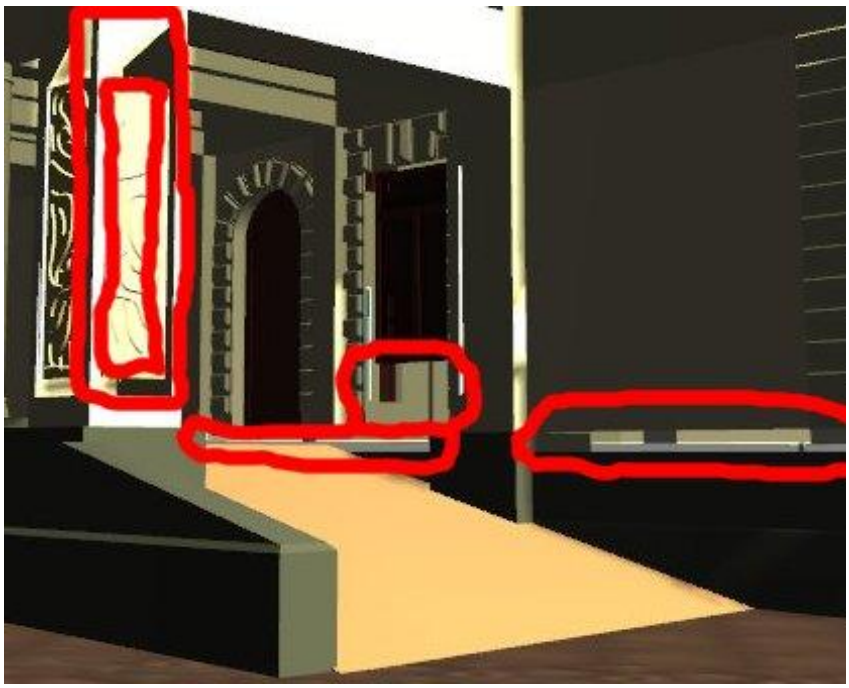
Mallien saattaminen Cinema 4D:stä Unity 3D:n puolelle toi mukanaan omat hankaluudet. Työtä tehdessä Unity 3D ei vielä tukenut suoraan .c4d-tiedostoja, toisin kuin nykyään, vaan mallit piti muuttaa .fbx-muotoon, jota Unity 3D käyttää. Nykyään .c4d-tiedostoja tuetaan suoraan, mutta tällöinkin Unity 3D muokkaa ne .fbx-muotoon ennen käyttöönottamista, joten muutoksen voi vieläkin tehdä Cinema 4D:n puolella.

Myös mallien rakenteessa oli ongelma, joka tosin oli helppo korjata. Kun malli viedään Archicadista Cinema 4D:hen, muuttuu mallien polygon-rakenne siten, että jokainen polygon on ikään kuin väärin päin. Tämä aiheuttaa sen, että mallit eivät näy Unity 3D:ssä oikein. Ongelman voi ratkaista yksinkertaisesti kääntämällä mallista kaikki polygonit oikein päin.

Mallien rakenteen monimutkaisuus aiheuttaa myös muita näkyvyysongelmia. Koska pinnoissa on todella lähellä toisiaan olevia polygoneja tai muita vastaavia ongelmia, tulee malleihin erilaisia virheitä, kuten katoavia seiniä. Monimutkaisuus aiheuttaa myös varjostusongelmia; koska rakennuksissa on paljon eri objekteja, luo jokainen näistä objekteista eri varjoa.

Yksi selvitystyön suurimmista ongelmista syntyi, kun malleja Unity 3D:hen tuotaessa jokaisen mallin jokainen polygon varjostui virheellisesti. Koska käyttöohjeet olivat puutteelliset, eikä varmaa tietoa virheen aiheuttajasta ollut, kulutin todella pitkiä aikoja Cinema 4D:ssä, koska oletin, että virhe olisi syntynyt jo siellä, joko vienti-asetuksissa tai mallin phong-varjostus -asetuksissa. Lopulta kuitenkin epävirallisten tukifoorumien avulla selvisi, että mallien polygonien normaalien kulma tuli asettaa oikein vielä Unity 3D:n puolella. Tämä korjasi varjostusvirheet, mutta samalla lisäsi polygonien näkyvyysongelmia.

Kuvassa 17 on eräs VirtuaaliViipurin rakennuksista Unity 3D:n pelitilassa. Kuvasta näkee, että erinäisiä listoja ja muita rakennuksen osia puuttuu. Syynä tähän voi olla mahdollisesti liian lähekkäin asetetut polygonit.



Kuva 17: Rakennuksen seinistä puuttuu osia mallin rakenteellisen monimutkaisuuden aiheuttamien virheiden vuoksi

6.4 Unity 3D ja valmis tuote

Mallien siirtämisen jälkeen Unity 3D:hen piti luoda projekti ja scene. Projekti sisältää nimensä mukaisesti kaikki projektin tiedostot. Scene on varsinainen peliympäristö, johon kaikki halutut komponentit asetellaan. Erilaisia komponentteja ovat 3D-mallien lisäksi muun muassa valot, äänilähteet ja erilaiset skriptit, joiden avulla voidaan vaikuttaa pelimaailman toimintaan.

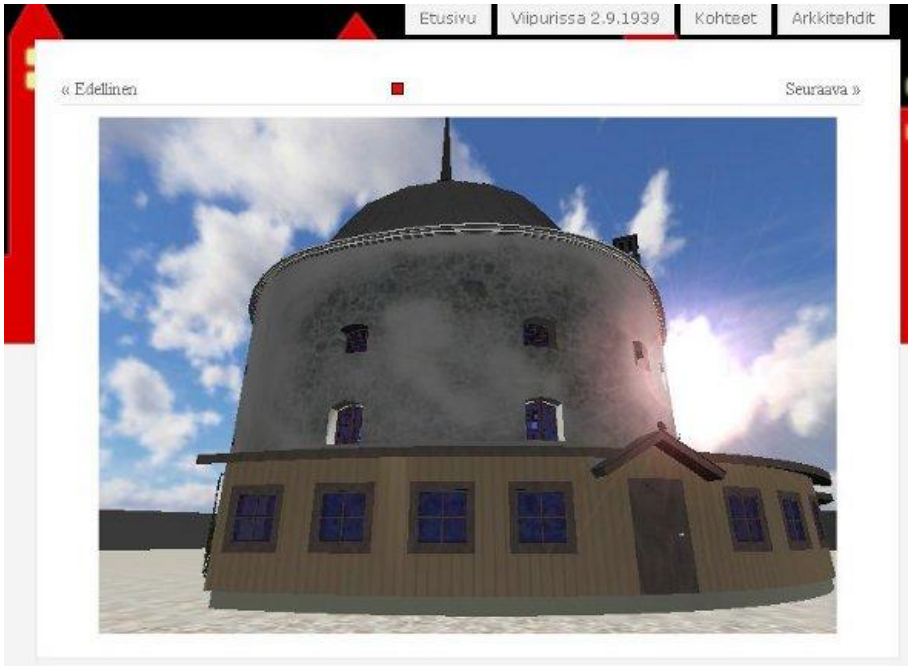
Yksinkertaisimman ympäristön VirtuaaliViipurille saa aikaiseksi, kun asettaa scenelle maan, halutut rakennukset, ja näille törmäyksen tunnistukset, jotta käyttäjä ei pääse kulkemaan seinistä läpi, valot ja First Person Controller -komponentin, joka lisää sceneen kameran ja skriptit liikkumisen mahdollistamiseen.

Kun halutunlainen scene on valmis, pystyy Unity 3D:ssä valitsemaan erilaisia grafiikan laatuasetuksia. Esimerkiksi, käytetäänkö antialiasointia, tai millä laadulla varjot halutaan esittää.

Laatuasetusten asettamisen jälkeen voi valmiin pelin julkaista erillisen sovelluksen Windowsille, Mac OS X:lle ja WWW-selaimelle. Kuhunkin voi säätää erikseen laatuasetukset, jolloin uusia versioita julkaistaessa ei laatuasetuksiin enää tarvitse koskea.

VirtuaaliViipurin materiaaleista luotavien sovellusten ongelmana on se, että koska mallien rakenteet ovat niin monimutkaisia, tulee sovelluksista raskaita käsitellä Unity 3D:ssä. Samasta syystä jokaiselle rakennusmateriaalille erikseen määriteltyjen tekstuurien valtavan määrän vuoksi tulee sovellukselle todella paljon kokoa. Jo pari korttelia vie yli 100 megatavua tilaa. Tämä on suuri ongelma, koska tavoitteena olisi saada koko Viipurin liikkuva maailma internetiin.

Kuvassa 18 on kuvankaappaus VirtuaaliViipurin -projektin sisarprojektista, johon loin opinnäytetyön kautta oppimillani metodeilla VirtuaaliViipurin 3D-malleista materiaalia. Kuvassa on verkkosivu, johon on upotettu Unity 3D -liitännäinen, jonka avulla tekemääni peliympäristöä pystyi tutkailemaan WWW-selaimella.



Kuva 18: VirtuaaliViipuriin läheisesti liittyvään projektiin selvitystyön pohjalta tehty ympäristö, jonka keskeisenä elementtinä toimii Viipurin Pyöreä Torni

7 Lopuksi

Koska VirtuaaliViipuria varten luotuja rakennuksia ei ole luotu sovelluksella ja tekniikalla, jotka soveltuisivat erityisesti reaaliaikaisen grafiikan käyttötarkoituksiin, ovat mallit jokseenkin vaikeasti hallittavia, mutta myös aivan liian raskaita ja tilaa vieviä. Jatkokehitystä ajatellen, niitä ei ole järkevää käyttää suurten kokonaisuuksien luomiseen.

Mikäli tähän mennessä luotuja rakennuksia haluttaisiin esitellä internetissä Unity 3D:n avulla, niin näkemykseni olisi, että alueet tulisi pitää pitää mahdollisimman pieninä. Esimerkiksi yhden korttelin kokoisten alueiden esittelemiseen ne käyvät varsin mallikkaasti, koska tällöin tiedostokoko ei ole päässyt aivan hurjaksi, eikä koneelta vaadittaisi aivan järjettömiä määriä prosessointitehoa. Lisäksi pienten kokonaisuuksien luominen ja julkaisu on nopeampaa.

Jotta rakennukset olisivat käteviä käsitellä ja kevyitä koneelle, kannattaisi ne tehdä uudestaan eri tekniikalla. Tekstuureja pitäisi olla yksi per rakennus, jossa olisi kaikki rakennusmateriaalit. Rakennusten rakenteen tulisi olla merkittävästi yksinkertaisempi. Suoralla seinällä ei tarvitse olla suuria määriä polygoneja prosessointia hidastamassa, seinien kaksikerroksisuudesta puhumattakaan. Myös yksittäisten talojen tulisi olla yksittäisiä objekteja, jolloin vältyttäisiin muun muassa ylimääräisiltä varjostusvirheiltä. Tietysti rakennuksen erikoisempia, koristeellisia osia on ihan järkevä mallintaa, jolloin näyttävyys säilyisi. Kuitenkin painopiste täytyisi silti pitää mallien rakenteellisessa keveydessä ja panostaa hyvän näköiseen ja riittävän korkeatarkkuuksiseen teksturointiin.

Vaikka en henkilökohtaisesti suurempaa kokonaisuutta valmiiksi saanutkaan, niin uskon, että opinnäytetyönä tekemästäni selvityksestä on hyötyä projektin parissa työskenteleville opiskelijoille. Uskon, että tästä on hyötyä myös muille rakennustekniikan opiskelijoille, jotka haluavat esitellä luomuksiaan pelimaailman avulla.

Lähteet

Activision, 2007
[peli]
Call of Duty 4: Modern Warfare

Akenine-Möller, T, Haines, E & Hoffman, N, 2008
Real-Time Rendering. 3rd Edition
Natick, MA, USA: A K Peters, Ltd.

Assembly Organizing, 2010
[www][viitattu 19.10.2010]
http://www.assembly.org/summer10/compos/realtime

Cross, J, 2007
[www] [viitattu 19.10.2010]
http://www.extremetech.com/article2/0,2845,2136961,00.asp

Dinnye, 2010
[www] [viitattu 19.10.2010]
http://dinnye.neobase.hu/content/mission

Edge Staff, 2008
[www] [viitattu 19.10.2010]
http://www.next-gen.biz/features/a-brief-history-3d

Evans, A, 2001
Four Tricks for Fast Blurring in Software and Hardware
[www] [viitattu 19.10.2010]
http://www.gamasutra.com/view/feature/3102/four_tricks_for_fast_blurring_in_.php

Luebke, D, Reddy, M, Cohen, J.D., Varshney, A, Watson, B & Huebner, R 2003
Level of Detail For 3D Graphics
San Francisco, CA, USA: Morgan Kaufmann Publishers

McTaggart, G, 2007
HDR in Valve's Source Engine
[online] [viitattu 19.10.2010]
http://www.valvesoftware.com/publications/2006/SIGGRAPH06_Course_HDRI
nValvesSourceEngine_Slides.pdf

O'Rorke, J & James, G, 2004
[www] [viitattu 19.10.2010]
http://www.gamasutra.com/view/feature/2107/realtime_glow.php

Puhakka, A, 2008
3D-grafiikka
Helsinki: Talentum Media

Ryynänen, P
[www] [viitattu 19.10.2010]
http://www.niksula.cs.hut.fi/~pjryynan/stu4portfolio/antialiasointi/antialiasointi.html

Sony Computer Entertainment Europe, 2005
[peli]
The Getaway 3

Unity Technologies, 2010a
[www] [viitattu 19.10.2010]
http://unity3D.com/unity/

Unity Technologies, 2010b
[www] [viitattu 19.10.2010]
http://unity3d.com/support/documentation/Manual/Textures.html

Unity Technologies, 2010c
[www] [viitattu 19.10.2010]
http://unity3d.com/support/documentation/Components/class-QualitySettings.html

Unity Technologies, 2010d
[www] [viitattu 19.10.2010]
http://unity3d.com/support/documentation/Manual/Lightmapping.html

Unity Technologies, 2010e
[www] [viitattu 19.10.2010]
http://unity3d.com/support/documentation/Components/script-BloomAndFlaresEffect.html

Unity Technologies, 2010f
[www] [viitattu 19.10.2010]
http://unity3d.com/support/documentation/Components/script-BlurEffect.html

Unity Technologies, 2010g
[www] [viitattu 19.10.2010]
http://unity3d.com/support/documentation/Components/script-DepthOfFieldEffect.html

Valve Software, 2005
[www] [viitattu 19.10.2010]
http://developer.valvesoftware.com/wiki/Half-Life_2:_Lost_Coast

VirtuaaliViipuri
[www] [viitattu 19.10.2010]
http://virtuaaliviipuri.tamk.fi

Virtual Berlin, 2010
[www] [viitattu 19.10.2010]
http://www.virtual-berlin.de

Vlachos, A, 2008
Post Processing in The Orange Box
[online] [viitattu 19.10.2010]
http://www.valvesoftware.com/publications/2008/GDC2008_PostProcessingInTheOrangeBox.pdf