LAB University of Applied Sciences

Optimoidun 3D-mallin luonti peliympäristöön

LAB-ammattikorkeakoulu
Insinööri (AMK), Tieto- ja viestintätekniikka
Mediatekniikka
2022
Santeri Lehtonen

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Lehtonen, Santeri	Opinnäytetyö, AMK	2022
	Sivumäärä	
	30 + 2	

Työn nimi

Optimoidun 3D-mallin luonti peliympäristöön

Tutkinto ja koulutusala

Insinööri, Tieto- ja viestintätekniikan koulutus (AMK)

Tiivistelmä

Työn tavoitteena oli tutkia optimointia pelikehityksessä sekä 3D-mallien luonnissa. Tutkimuksen 3D-mallin osiosta saatiin tietoa tutkimuksen case-osaan, jossa luotiin 3D-malli mahdollisimman optimaalisesti. Työn aineistona käytettiin pääosaisesti internetistä löydettyjä lähteitä.

Tutkimuksen alussa tutkittiin optimaatiota pelinkehityksessä sekä tapoja, joilla kehittäjä pystyy optimoimaan kehitettävän pelin eri osa-alueita. Seuraavaksi tutkittiin minkälaisia pelimoottoreita ja mallinnusohjelmia käytetään.

Case-osuudessa valmistettiin sekä optimoitiin 3D-malli rivitalosta. Työssä käytettiin Gimp-ohjelmaa tekstuuripohjan luomiseen, 3DS Max -ohjelmaa mallintamiseen ja Unreal Engine -pelimoottoria mallin testaamiseen sekä tarkasteluun. Erityisesti työssä huomioitiin mallin tehonkulutus sekä mallin toimiminen Unreal Engine -ympäristössä.

Lopputulos oli hyvä. Mallin tehonkulutus saatiin pidettyä alhaisena ja mallissa ei Unreal Enginen sisällä huomattu suurempia ongelmia. Suurimmat ongelmat johtuivat mallinnusohjelmien kaatumisesta mallinnuksen aikana. Jonkin verran ongelmia tuotti myös mallinnusohjelman valintojen runsaus.

Tutkimusta tehdessä tuli opittua paljon mallinnusohjelmien käytöstä sekä mallien viemisestä Unreal Engineen. Opitun tiedon avulla projektia voisi halutessaan jatkaa esimerkiksi mallin sisätilojen yksityiskohtien lisäämisessä tai pelimaailman laajentamisessa.

Asiasanat

optimointi, pelimoottori, tekstuuri, 3D-malli

Abstract

Author(s)	Type of Publication	Published
Lehtonen, Santeri	Bachelor's thesis	2022
	Number of Pages	
	30 + 2	

Title of Publication

Creating an optimized 3D model for the gaming environment

Abstract

The aim of the work was to study Optimization in game development and in the creation of 3D models. The 3D model section of the study provided information on the case part of the study, where the 3D model was created as optimally as possible. Sources found on the Internet were mostly used as material for this work.

At the beginning of the study, optimization in game development and the ways in which a developer can optimize different aspects of the game being developed were studied. Next, the types of game engines and modeling programs commonly used were investigated.

In the case section, a 3D model of a terraced house was prepared and optimized. The work used Gimp to create a texture template, 3DS Max to model, and the Unreal Engine to test and review the model. In particular, the power consumption of the model and the operation of the model in the Unreal Engine environment were taken into account.

The end result was good. The power consumption of the model were kept down and no major problems were observed within the model within Unreal Engine. The biggest problems were due to the crash of modeling programs during modeling. The plethora of choices in the modeling program also posed some problems.

During the research, a lot was learned about the use of modeling programs and the export of models to Unreal Engine. With the help of the learned information, the project could be continued, for example, by adding details to the interior of the model or expanding the game world.

Keywords

optimization, game engine, texture, 3D model

Sisällys

1 Johdanto

Raportin tavoitteena oli tutkia 3D-mallien luontia optimoinnin näkökulmasta. Optimaatiolla tarkoitetaan 3D-mallien luomista mahdollisimman vähän laskentatehoa kuluttaen. Laskentatehoa vähentämällä yksittäisestä pelin osasta antaa kehittäjälle mahdollisuuden lisätä enemmän ominaisuuksia pelimaailmaan. Raportissa myös tutkittiin optimaatiota muissa pelikehityksen osa-alueissa.

Tutkimuksessa ensin käsitellään optimoinnin määritelmää sekä optimaation mahdollisuuksia pelikehityksen eri osa-alueilla. Eri menetelmiä käyttämällä kehittäjän on mahdollista säästää laskentatehoja monissa eri pelinkehityksen osissa.

Optimoinnin määrittämisen jälkeen tutkitaan yleisiä pelinkehityksessä käytettyjä ohjelmia. Tutkinnan kohteena on pelimoottorit sekä mallinnusohjelmat, ja minkälaisia ne ovat. Ohjelmien valinnalla on suuri merkitys, kun mietitään kehitettävän pelin kokoa ja mahdollisia ominaisuuksia.

Lopuksi tutkimuksessa luodaan oma 3D-malli, jonka on tarkoitus olla mahdolliisimman optimoitu, mutta silti hyvännäköinen. Työssä on tarkoitus käyttää teoriaosuudessa opittuja optimaatiotekniikoita. Projektissa käytetään internetistä ladattuja kuvia.

Tutkimuksessa käytettiin internetistä löydettyjä aineistoa. Aineiston löytäminen oli vaihtelevaa, sillä monet lähteet olivat hieman vanhoja. Projektin teossa myös ohjelmien omat dokumentoinnit olivat helposti ymmärrettäviä.

2 Optimointi

2.1 Optimoinnin määritelmä

Optimoinnin kirjallinen merkitys on teko, prosessi tai menetelmä tehdä jostakin (kuten suunnittelusta, järjestelmästä tai päätöksestä) mahdollisimman täydellinen, toimiva tai tehokas (Merriam-Webster). Videopeleissä optimoinnin avulla tehdään pelin käyttämistä ominaisuuksista mahdollisimman tehokkaita esimerkiksi vähentämällä yhden ominaisuuden tehovaatimusta. Tehovaatimusta vähentämällä pelimaailmassa sijaitseva ominaisuus kuluttaa vähemmän laskentatehoa käyttäjän käyttämästä tietokoneesta tai konsolista, minkä avulla käyttäjän pelikokemus paranee. Tehovaatimuksen alentaminen antaa myös kehittäjälle mahdollisuuden lisätä enemmän ominaisuuksia peliin. (Lowrey, 2017.)

Jokaisella pelillä on käytettävissään tietty kiinteä määrä laskentaresursseja (Garney & Preisz 2011). Pelin kehittämisessä tietokoneelle mietitään kohdeyleisölle tyypillisen tietokoneen laskentatehoa. Tärkeimmät tietokoneen laskentatehoon vaikuttavat osat ovat näytönohjain (GPU), prosessori (CPU) sekä RAM-muisti. Tietokoneelle kehitettäessä otetaan huomioon myös pelin erilaiset grafiikkamuodot sekä resoluutiot. (Pan, 2017.)

Konsolipeliä kehittäessä pelimaailmaa optimoidaan kohdekonsolin laskentatehon mukaan. Konsoleille kehittäessä otetaan myös huomioon saman konsolin mahdollisesti päivitetyt versiot (esimerkiksi Playstation 4 -konsolille optimoinnissa otetaan huomioon myös Playstation 4 Pro versio).

Optimoinnin tavoitteena on lisätä mahdollisimman paljon tehdyn työn määrää yhtä laskentatehoyksikköä kohden. Optimoimalla CPU:n ja GPU:n käytön voi lisätä enemmän polygoneja, fysiikkaa, tekoälyä, reagointikykyä sekä immersiota (Garney & Preisz 2011). Optimoinnilla on mahdollista myös vähentää pelin kokoa.

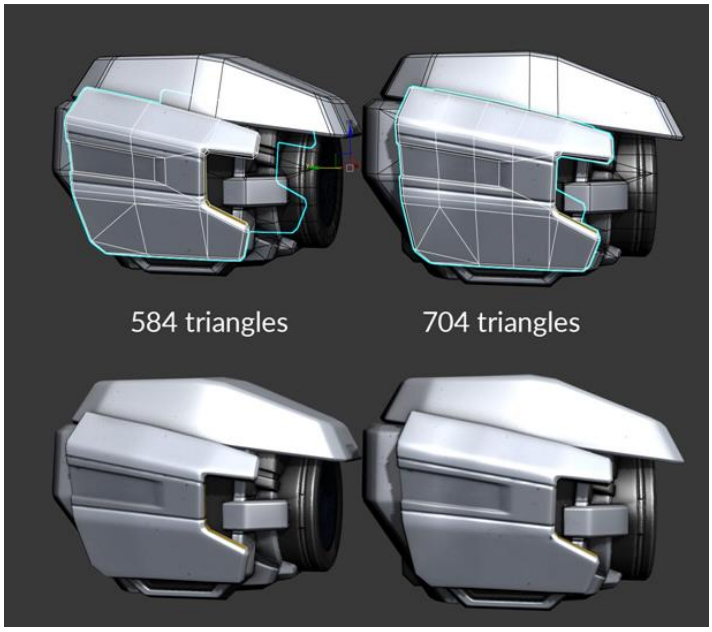
Optimoinnin ei pitäisi määrätä koko pelinkehittämisen suuntaa. (Sloka-Frey, 2018.) Vaikka pelin optimointi on tärkeää pelin pelattavuuden kannalta, pitää itse pelimaailman ja pelattavuuden olla myös hyvällä tasolla. Pelimaailmaa voi optimoida monella eri tavalla. Pelin laskentatehon optimointi kannattaa pitää mielessä koko kehitysprosessin ajan. Liikaa optimointia tulee kuitenkin välttää, sillä mikro-optimoinnin tekeminen liian aikaisin vain hidastaa työnkulkua, mikä johtaa virheisiin ja joskus tekee pelin ylläpidosta vaikeaa. (Pan 2017.)

2.2 Mallit

Pelimaailmassa on usein monta eri 3D-mallia ruudulla samaan aikaan. Malleina toimivat lähes kaikki fyysistä tilaa vaativat objektit, esimerkiksi puut, rakennukset ja huonekalut.

Pelinsisäiset hahmot kuuluvat myös malleihin, kuten ihmiset ja eläimet. Mallit luodaan yleensä mallinnukseen tarkoitetuilla ohjelmilla kuten Blenderillä tai 3DS Maxilla. Valmiit mallit sijoitetaan pelimoottorin (esimerkiksi Unreal Engine tai Unity) kautta pelimaailman sisälle. Pelimoottoreissa on myös omat työkalut mallien tekemiseen, mutta mallinnukseen tarkoitetut ohjelmat antavat useimmiten paremmat työkalut mallien luomiseen. Pelimoottoreihin sisältyy usein Asset Store, mistä on mahdollista ladata valmiita malleja ilmaiseksi tai maksamalla.

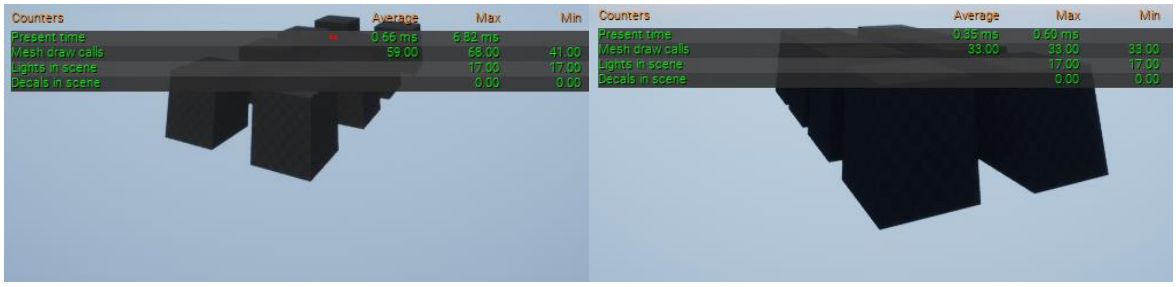
Polygonien vähentäminen on yksi tapa vähentää mallien tehonkulutusta. Polygoneilla tarkoitetaan 3D-mallin pinnan muodostavista muodoista, jotka ovat usein kolmioita tai neliöitä. Varsinkin kolmioiden laskeminen vaatii laskentatehoa tietokoneen GPU:lta. (Arm Developer.) Polygoneita poistamalla 3D-mallin tehonkulutus laskee, mutta myös 3D-mallin yksityiskohdat voivat kadota. Polygonien poistamiselle löytyy usein omat työkalut mallinnusohjelmista. Kuvassa 1 näytetään esimerkki mallista, josta poistetaan ylimääräisiä polygoneja mallin yksityiskohtia menettämättä.



Kuva 1. Ylimääräisten Kolmioiden poisto (Arm Developer)

Mesh draw call tapahtuu, kun CPU lähettää GPU:lle piirtopyynnön esimerkiksi 3D-mallista. Pyyntö sisältää esimerkiksi mallin muodon sekä materiaalin. Jos malliin lisätään ominaisuuksia, lähetettyjen draw call:ien määrä nousee. (Unity Documentation.) Suuret määrät mesh draw call:eja kuormittaa tietokoneen GPU:ta aiheuttaen tehonkulutusta pelimaailman osassa, jonka objekteissa on suuri määrä materiaaleja. (Unreal tips, 2019.)

Yksi tapa mallien optimoinnissa on tehdä samanlaiset objektit yhdestä instancesta. Instancella tarkoitetaan yhden muodon kopiointia sekä renderöintiä, minkä avulla yhdestä meshistä voi luoda haluttu määrä kopioita pelimaailmaan. (Unity Documentation.) Kuvassa 2a on kahdeksan kuutiota Unreal Engine -pelimoottorin sisällä, Mitkä on luotu yksittäin. Kuvassa 2b on kahdeksan kuutiota, jotka on luotu yhden instancen avulla. Instancen avulla luodut kuutiot kehittävät lähes puolet vähemmän mesh draw call:ia kuin yksittäin luodut kuutiot.

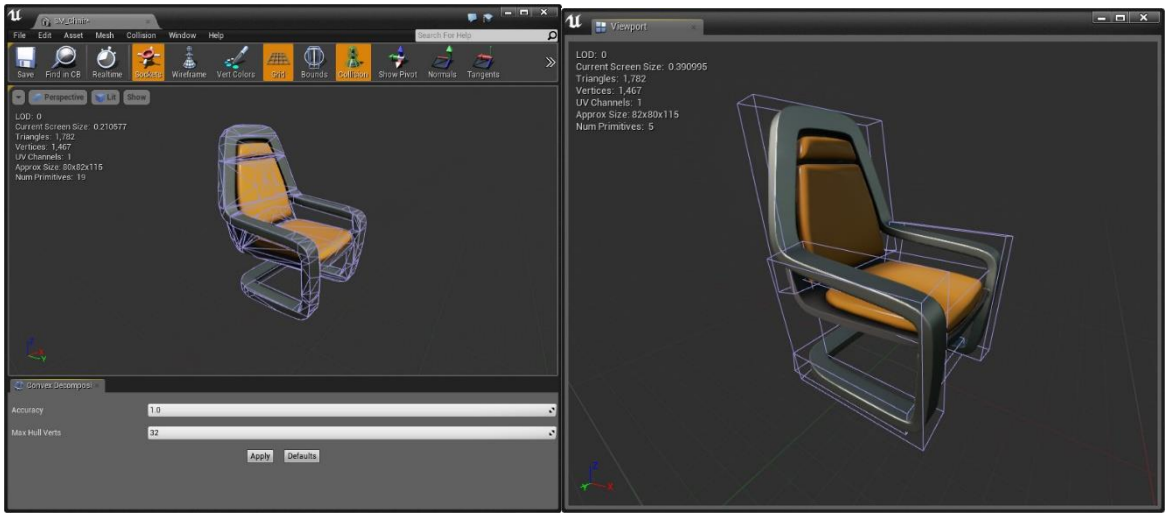


Kuva 2a-2b. Kahdeksan kuutiota toteutettu Yksittäin sekä Instancen avulla

Peliä kehittäessä päätetään myös, mitkä objektit ovat pelille välttämättömiä. Vähemmän tarpeellisista objekteista voidaan säästää paljon laskentatehoja; jos objekti sijaitsee paikassa, johon pelaajan ei ole tarkoitus päästä, voidaan objektista poistaa ominaisuuksia. Rakennuksessa, johon pelaaja ei voi mennä sisään, voidaan poistaa osa sisätiloista. Rakennuksen osat voidaan myös liittää yhteen yhdeksi isoksi objektiksi, mikä laskee rakennuksen tehovaatimuksia. Tällöin peliä pyörittävän koneen ei tarvitse ladata montaa eri objektia vaan peli lataa yhden ison objektin. Rakennuksien osien liittämisen toisiinsa voidaan suorittaa esimerkiksi 3DS Maxin sisällä Collapse-toiminnolla tai Unreal Enginen sisällä Merge Actors -komennolla.

Pelaajahahmolle välttämättömien mallien collision box:sta on mahdollista vähentää myös tehovaatimuksia. Collision box (jatkossa CB) tarkoittaa näkymätöntä aluetta mallin ympärillä, mihin muut CB:in sisältävät objektit voivat koskettaa. Esimerkiksi CB:in omaavan pelaajahahmon kävellessä seinään, johon on myös lisätty CB, pelaajahahmo pysähtyy. Mallien optimoinnissa myös näiden CB:ien tehovaatimuksia on mahdollista muokata. Yleinen peleissä käytetty tekniikka on fyysisten verkkojen lähentäminen sisäänrakennetuilla primitiiveillä, kuten kuutioilla ja palloilla. (Pan 2017.) Esimerkiksi pelaajahahmon lähellä oleville taloille voidaan lisätä yksinkertainen kuution muotoinen CB, jos taloon ei ole mahdollista päästä sisään. Kuvassa 3a Näytetään tuolin mallia, jonka CB on luotu tarkasti ja

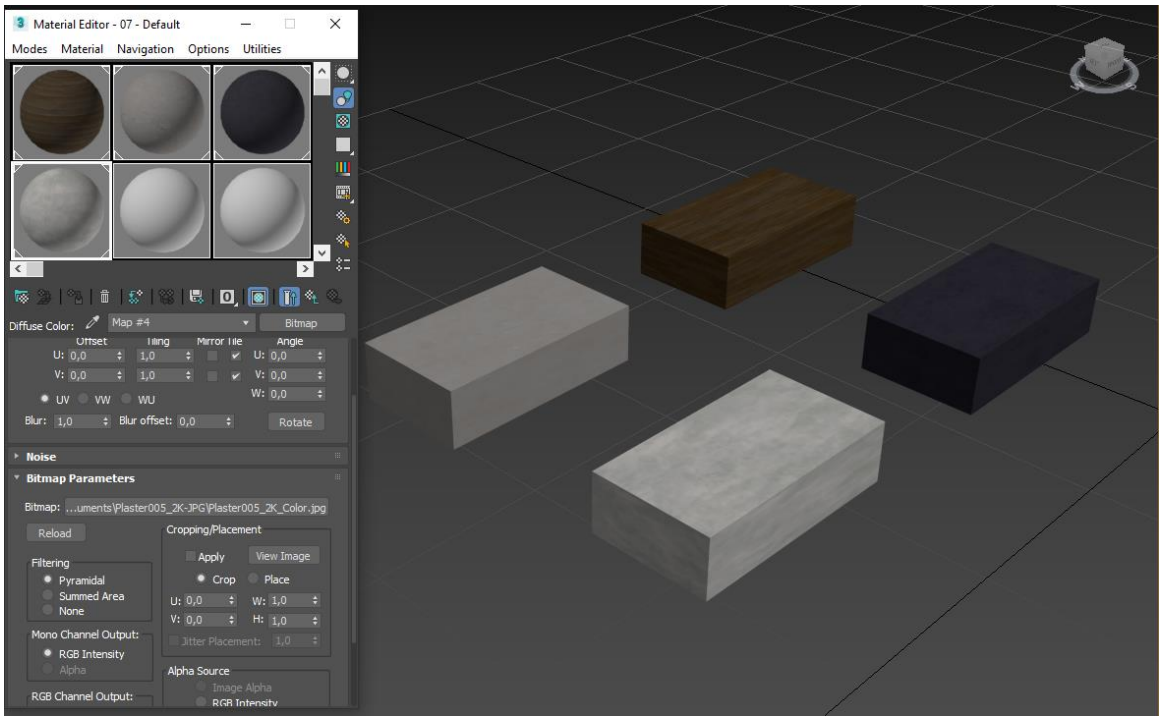
monimutkaisesti. Kuvassa 3b saman tuolin CB on luotu yksinkertaisemmista muodoista. Molempien kuvien CB:it luotiin Unreal Enginen avulla. Vaaleat viivat tuolien ympärillä kuvaavat mallin CB:ia.



Kuva 3a-3b. Mallien CB:it (Unreal Engine Documentation)

2.3 Materiaalit ja tekstuurit

Materiaaleja liitetään mallin pintoihin jotta saadaan pinnalle haluttu väri tai kuvio. Materiaalille on myös mahdollista liittää lisäominaisuuksia, kuten karkeutta jos materiaali esittää esimerkiksi asfaltin pintaa. Tekstuureilla tarkoitetaan kuvapohjaa, johon materiaalin värit ja kuviot perustuvat. Tekstuurit luodaan kuvankäsittelyohjelmalla (esimerkiksi PhotoShop tai GIMP) ja materiaalit liitetään malleihin mallinnusohjelman (esimerkiksi 3DS Max) tai pelimoottorin (esimerkiksi Unreal Engine) sisällä. (GameDevInsider.) Tekstuurit itse ovat eriformaattisia kuvatiedostoja (esimerkiksi JPG tai PNG). Kuvassa 4 näytetään materiaaleja 3DS Max -ohjelman sisällä. Kuvassa neljään eri suorakulmioon on lisätty eri materiaalit. Jokaisella materiaalilla on eri tekstuuri ja tekstuurina toimii yksinkertainen PNG-formaatin omaava kuva.



Kuva 4. neljä suorakulmiota eri materiaaleilla

Yleinen tapa käyttää materiaaleja onkin tehdä jokaiselle eriväriselle tai erikuvioiselle mallipinnalle oma materiaali. Jos jokaisella käyttöliittymän elementillä on erillinen tekstuuri, se piirretään erillisellä draw call:illa. Hyvä tapa säästää mallien tehovaatimuksia on käyttää uudelleen samaa materiaalia eri malleissa. Näin peli pyytää käyttäjän konetta lataamaan vain yhden materiaalin samanlaisiin pintoihin. (Sciutteri, 2016.)

Mallin materiaalin tekstuureita voidaan optimoida käyttämällä Atlas-tekstuureita 3D-mallien luonnissa. Atlas-tekstuuri tarkoittaa yhtä suurta tekstuurikuvaa monen pienemmän tekstuurikuvan sijaan. Suuremman tekstuurikuvan sisälle sijoitetaan kaikki yhdelle mallille halutut tekstuurit. Atlas-tekstuuriin pohjautuva materiaali kartoitetaan erikseen mallin pinnalle mallinnusohjelmassa tai pelimoottorin sisällä. (Sciutteri, 2016.) Koska tietokoneen tarvitsee ladata vain yhden suuren tekstuurin yhtä mallia kohti, säästää se GPU:n tehonkulutuksessa. Kuvassa 5 näytetään esimerkki Atlas-tekstuurista.

Kuva 5. Atlas-tekstuuri (envato tuts+ 2016)

Yksi tapa optimoida tekstuurien tehonkulutusta vanhemmille laitteille on myös käyttää power of two -ulottuvuuksia tekstuureja tehdessä. Tällä tarkoitetaan kuvan resoluution ulottuvuuksia, joiden pikseliarvot saadaan laskemalla numero kaksi eri potensseihin; 2x2, 4x4, 8x8, 16x16 ja niin edelleen. Vanhemmilla laitteilla kuvat, jotka eivät noudata power of two -ulottuvuuksia, laajennetaan automaattisesti seuraavaksi korkeimpaan power of two -ulottuvuuteen. (Pan, 2017.) Tuloksena esimerkiksi 260x260 px -kokoinen kuva vie saman verran muistia kuin 512x512 px -kokoinen kuva.

2.4 Valaistus

Kuten tosielämässä, suuri osa pelin estetiikkaa on pelin valaistuksessa. Valaistukseen kuuluu pelkän valon käytön lisäksi esimerkiksi varjojen käyttö, oikean linssin valinta sekä käyttö ja ympäristön tunnelma. Onnistuneesti käytettynä valaistuksen avulla saa pelimaailman tuntumaan elävämmältä. Pelimaailman estetiikkaa on myös mahdollista parantaa hyvän valaistuksen avulla. (Foundry, 2020.)

Valaistusta muokataan peleissä kohtauksen mukaan. Kun mietitään pelin eri skenejä, mietitään myös minkälainen valaistus sopii parhaiten kyseiseen kohtaukseen. Onko ympäris-

tö täysin valaistu vai tuleeko valo vain tietyistä osista kohtauksen ympäristöstä? Huomioon otetaan myös ympäristön objektien varjot ja varjojen muodot. Hämärämmässä kohtauksessa saatetaan lisätä lisäefektejä ympäristöön, kuten sumua ympäristön pohjalle. Pelaajakameran linssiä muokkaamalla voidaan myös muuttaa koko pelin tunnetta. (Foundry, 2020.) Hyvänä esimerkkinä on Capcomin vuonna 2009 julkaistu Resident Evil 5, minkä käyttämä kamerasuodatin yksinkertaistaa pelin värimaailmaa. Kuvassa 6 esitetään Resident Evil 5 -pelin maailmaa. Yläkuvassa Pelissä on kamerasuodatin päällä ja alakuvassa esitetään sama kohtaus ilman kamerasuodatinta.

Kuva 6. Resident Evil 5 kamerasuodatin (Capcom 2008)

Valaistusta luodessa on pidettävä mielessä valaistuksen eri osa-alueet ja miten niiden käyttäminen rasittaa tietokonetta peliä pyörittäessä. Enemmän valoja tarkoittaa hitaampaa suorituskykyä. Yksi tapa tehovaatimusten alentamiseen on vähentää pelissä käytettyä dynaamisen valaistuksen käyttämistä Staattisen valaistuksen sijaan. Dynaamisella valaistuksella tarkoitetaan valaistuksen muuttumista pelaajahahmon toimintojen ja pelimaailman

tapahtumien mukaan. (Pan, 2017.) Esimerkiksi jos pelimaailmassa on päivänaikaan perustuva dynaaminen valaistus, milloin pelin valaistus muuttuu hämärämmäksi päivänajan lähestyessä yötä ja kirkastuu yön muuttuessa aamuksi. Dynaamisella valaistuksella saa pelimaailman tuntumaan realistisemmalta ja elävämmältä. Staattinen valaistus ei muutu pelin tapahtumien mukaan, valonlähteet ja valaistu alue pysyvät samana pelin tapahtumista riippumatta. Peliä optimoidessa tulee kuitenkin käyttää dynaamista valaistusta maltilla, sillä dynaamisen valaistuksen liiallinen käyttö kuormittaa tietokoneen GPU:ta. Kehittäjän tulee miettiä pelin valaistusta optimoidessa missä ja milloin peli käyttää dynaamista valaistusta vai käyttääkö peli pelkästään staattista valaistusta, vaikka dynaaminen valaistus on laadukkaampi. Molempia valaistuksen muotoa käytettäessä hyvän tasapainon löytäminen auttaa paljon pelin laadussa sekä toimivuudessa.

Valosta syntyviä varjoja on myös mahdollista optimoida. Kuten valaistuksessa, varjojen käyttämisessä on kehittäjällä mahdollisuus valita dynaamisten- ja staattisten varjojen välillä. Erot dynaamisten ja staattisten varjojen välillä ovat hyvin samanlaiset kuin valaistuksessa. Varjojen tehovaatimuksia voi vähentää myös tekemällä varjoista yksinkertaisempia muodoltaan. Yksi varhaisimmista tavoista onkin lisätä tumma muoto mallin alle. (Evanson 2020).

2.5 Muut visuaaliset optimaatiomenetelmät

Pelaajahahmon ollessa tarpeeksi kaukana pelimaailma voi poistaa hetkellisesti pelaajahahmolle sillä hetkellä merkitsemättömät ominaisuudet (esimerkiksi mallit tai maastot). Tätä menetelmää kutsutaan culling-menetelmäksi. Culling-menetelmästä on 3 eri versiota: distance-, frustum- ja occlusion culling.

Distance culling -menetelmä poistaa kaiken tietyn etäisyyden päästä kamerasta. Objektien katoamisen pehmittämiseksi juuri ennen etäisyyttä usein tehdään esimerkiksi sumuefekti, etteivät pelin objektit vain katoa tyhjyyteen.

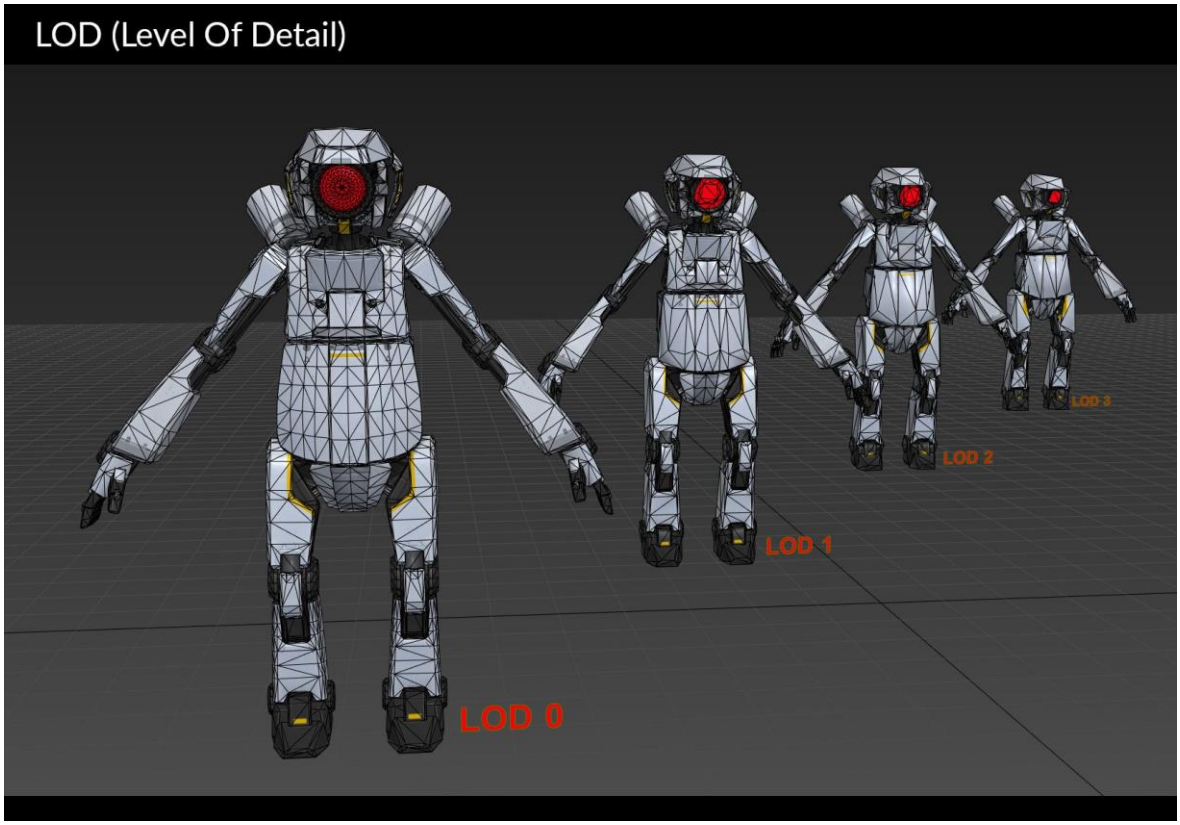
Toisen menetelmän, frustum cullingin, avulla pelimaailma rakentuu vain hieman pelaajahahmon näkökulmaa suuremmalle alueelle. Pelimaailman objektit ilmestyvät sekä katoavat pelaajan näkökulman liikkeen mukaan.

Occlusion culling -versiossa peli tarkastaa jatkuvasti, onko pelimaailman objektien edessä malleja, jotka estävät objektien näkymisen pelaajalle. Saadun tiedon avulla peli saa selville, mitkä pelin objektit ovat piilossa eli mitkä mallit voi poistaa maailmasta hetkellisesti. (Hider.) Occlusion culling -menetelmä sopii parhaiten pelimaailman keskitettyihin tiloihin, sillä menetelmä vaatii CPU:lta tehoja piilotettavien muotojen tunnistamiseen. Suurella alueella käytettynä occlusion culling -menetelmä piilottaa suuren määrän objekteja johtaen mahdolli-

sesti tehonkulutuksen kasvamiseen. (Torres Bonet, 2021.) Kuvassa 7 on esimerkki frustum culling menetelmästä. Kuva on otettu vuonna 2017 julkaistusta Horizon: Zero Dawn -pelistä. Kuvassa pelimaailma ilmestyy vain hieman pelaajan näkökentän ulkopuolelle. Seurauksena pelimaailman tehovaatimukset vähenevät, koska tietokoneen ei tarvitse pitää koko peli-maailmaa ladattuna jatkuvasti.

Kuva 7. Frustum culling -menetelmä Horizon: Zero Dawn:issa (Guerilla Games 2017)

Kehittäjä voi myös vähentää pelimaailman sisäisten mallien yksityiskohtia etäisyyden kasvaessa LOD-tekniikan avulla. Tekniikan avulla kehittäjällä on mahdollisuus vähentää mallin pinnan muodostuvia polygoneita, kun mallin etäisyys pelaajaan kasvaa mallin siluettia rikkomatta. LOD:ia käytetään myös eri etäisyyksillä, jolloin mallista poistuu eri määrä yksityiskohtia eri etäisyyksillä. (arm developer.) Kuvassa 8 on esimerkki LOD-tekniikan käytöstä. Kuvassa mallille on annettu eritasoiset LOD:it, joiden avulla mallin pinnasta poistuu kärkipisteitä etäisyyden kasvaessa.



Kuva 8. LOD-tekniikka eri tasoilla (arm developer)

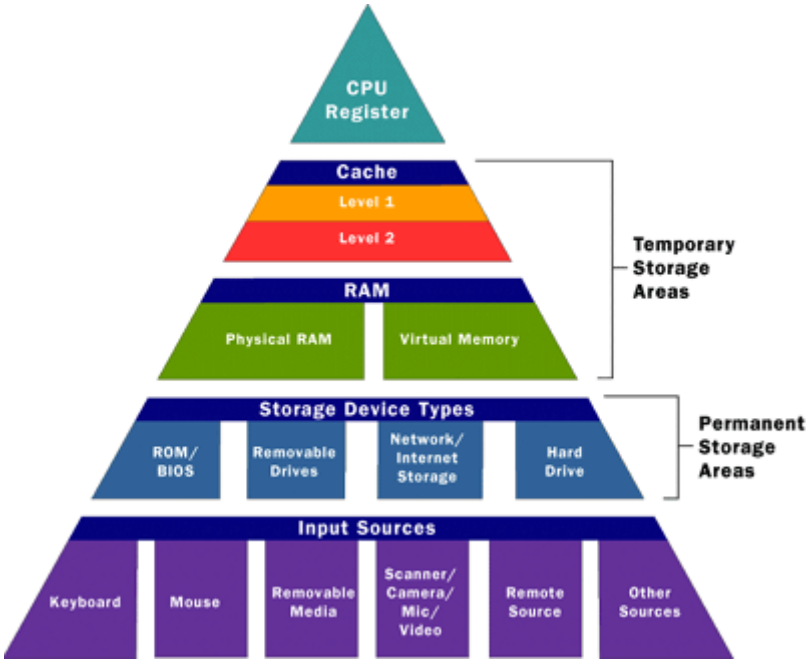
2.6 Koodi

Pelin koodi määrittää, miten pelimaailman sisäiset ominaisuudet toimivat keskenään. Pelkän mallin sijoittaminen pelimaailmaan ei aina riitä, sillä monien mallien tehtävänä on reagoida pelimaailman sisällä tapahtuviin tapahtumiin. Esimerkiksi pelimaailmassa sijaitsevat autot eivät itsenäisesti aja tiellä, ellei niitä koodin avulla ohjelmoida ajamaan itsenäisesti. Useimmissa peleissä koodi kirjoitetaan C-kieleen pohjautuvalla kielellä. Koodia kirjoitetaan usein koodaukselle tarkoitetuissa tekstieditoreissa kuten Microsoftin Visual Studiolla.

Pelin koodin optimoinnissa käytetään profilereita suorituskyvyn tutkimisessa. Profilerilla tarkistetaan ohjelman tehonkulutuksen tietokoneen eri osissa. Profiler myös näyttää koodin lähettämien kutsujen määrän tietokoneen eri osiin sekä kutsujen suorittamiseen kuluneen ajan. Saadun tiedon avulla kehittäjä tunnistaa mahdolliset tehonkulutukselta suuret osat pelin koodissa. (Altvater, 2020.)

Koodin avulla voidaan pelimaailman eri osia optimoida pidemmälle. Käytetystä pelimoottorista ja koodikielestä riippuen muistin vuorovaikutukset voivat tosin vähentyä. Kuvassa 9 on

kuvattuna muistihierarkia. Aivan hierarkian huipulla on tietokoneen prosessori (CPU) ja mitä alemmaksi hierarkiassa laskeudutaan, sitä enemmän aikaa kuluu tiedon siirtymisessä hierarkian huipulle. Suorituskyky on yleisesti osa dataa, ei koodia. (Lowrey, 2017.)



Kuva 9. Muistin hierarkia (Lowrey 2017)

Pelin koodia optimoidessa on hyvä sijoittaa mahdollisimman paljon pelin tehonkulutuksesta muistihierarkian huipulle. Riippuen käytetystä moottorista tai frameworkista, objektit voivat päivittyä jokaisen framen aikana. Jatkuva objektin päivittyminen vaatii CPU:lta tehoja, joten tehojen säästämiseksi voidaan vähentää päivityskutsujen määrää frameilta. Mahdollista on myös luoda kutsu koodiin, joka ilmoittaa objektin visuaalisten ominaisuuksien muutoksesta. Kutsu voidaan myös pakata erilliseen säiliöön, jolloin kutsu noudetaan vasta sitä tarvittaessa. (Sloka-Frey, 2018.)

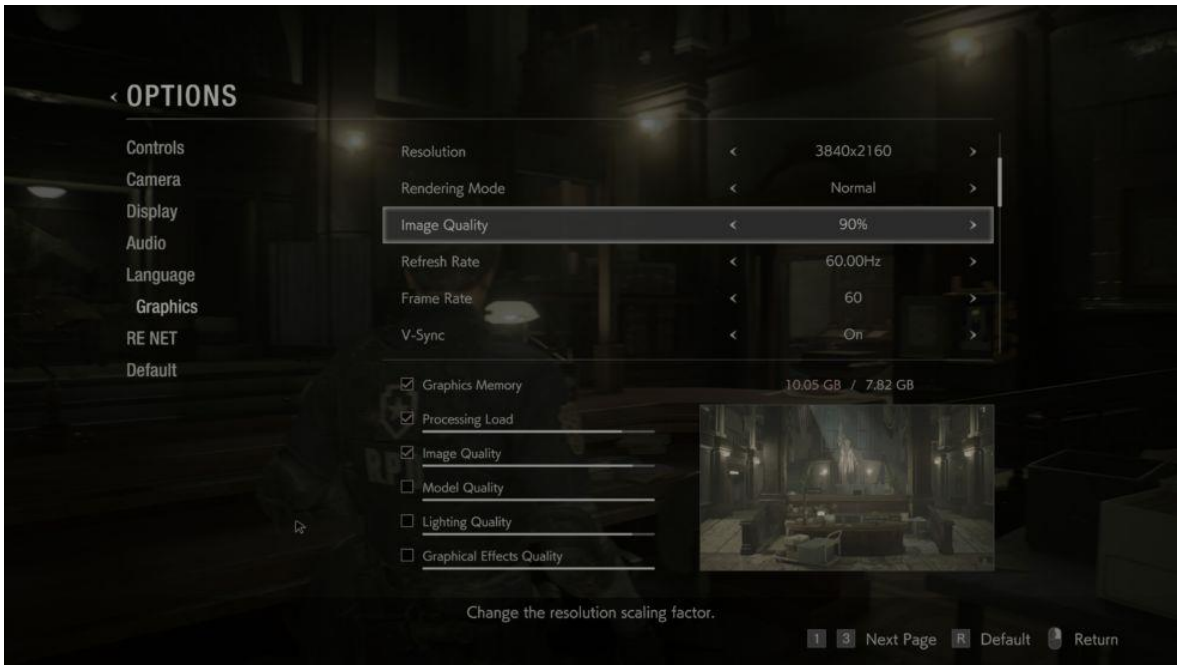
Pelien koodipuolella tyypillisesti tapahtuu monia laskufunktioita. Esimerkiksi peliohjainta käyttävän käyttäjän halutessa liikuttaa pelaajahahmoa käyttäjä liikuttaa ohjaimessa sijaitsevaa ”tattia”. Tatin sijainti ohjaimessa antaa arvon pelille, minkä avulla peli tunnistaa mihin suuntaan ja mahdollisesti kuinka nopeasti pelaajahahmo liikkuu. Laskennassa pelin koodi muuttaa ohjaimelta saadut arvot pelin omaan mittausjärjestelmään sopivaksi, kuten voimakertoimeksi. Jos pelissä on liikaa laskuja tai liian monimutkaisia laskukaavoja, CPU voi

kuormittua. Algoritmien toistuvien osien tekeminen yhdellä kertaa ja sen tallentaminen voi usein säästää huomattavia määriä prosessointitehoa. (Sloka-Frey 2018.)

Pelimaailman sisällä olevat mallit päivittyvät säännöllisin väliajoin. Mallien päivityksessä esimerkiksi mallin koko tai mallin animaatio voivat muuttua. Koodipuolella voidaan malleille määrittää eri päivitysrutiinit riippuen siitä, että onko päivitettävä malli käyttäjän näköpiirin sisäpuolella tai ulkopuolella. Mallin siirtyessä käyttäjän näkökentän ulkopuolelle mallin päivitykset voidaan pysäyttää, kunnes malli ilmestyy takaisin käyttäjän näköpiiriin. (Sloka-Frey, 2018.)

2.7 Asetukset

Peliä tietokoneelle kehitettäessä kehittäjän tulisi pitää mielessä pelin mahdolliset eritasoiset asetukset. Käyttäjän tietokoneen laskentatehosta riippuen joutuu käyttäjä mahdollisesti poistamaan käytöstä pelinsisäisiä ominaisuuksia sulavan käytön edellyttämiseksi. Kehittäjällä on mahdollisuus tehdä laadultaan eritasoiset asetukset eri tehoisille tietokoneille. Käyttäjä voi esimerkiksi poistaa varjot pelistä kokonaan tai valita erilaatuisia tekstuureita. Hyvillä asetuksilla pelin saavutettavuus nousee, sillä suuremmalla osalla pelin kohdeyleisöstä on mahdollisuus pelata peliä halutulla tasolla. Kuvassa 10 on esimerkki asetusvalikosta Capcomin vuonna 2019 julkaistusta Resdent Evil 2 Remake -pelissä. Kuvassa oikeassa alakulmassa on myös pieni kuva, joka näyttää, miten asetusten muuttaminen vaikuttaa pelimaailman näköön.



Kuva 10. Asetusvalikko (Capcom 2019)

Konsoleille peliä kehitettäessä on myös mahdollisuus antaa käyttäjälle valinta pelin suorituskyvyn muuttamisesta. Konsolien laskentakyky vaihtelee tosin paljon vähemmän tietokoneisiin verrattuna, vaikka joistain konsoleista on päivitettyjä ja tehokkaampia versioita olemassa. Myös eri valmistajien konsoleissa on pieniä eroja konsolien laskukyvyssä. Seurauksena usein käyttäjälle annetaan mahdollisuus valita paremman grafiikan ja paremman ruudunpäivittämisen välillä.

3 Ohjelmat

3.1 Ohjelmien valinta

Ennen kuin kehittäjä aloittaa pelin maailman ja -ominaisuuksien luomisen, on kehittäjän päätettävä, mitä ohjelmia pelin eri osien luonnissa käytetään. 3D-mallien tekoon valitaan sopivin mallinnusohjelma, pelille valitaan sopivin pelimoottori ja pelin koodikieli päätetään. Mahdolliset äänenmuokkausohjelmat valitaan sekä mietitään valmiiden ominaisuuksien hankinnan ja itse luotujen ominaisuuksien suhdetta. Eri ohjelmilla on eri vahvuudet ja heikkoudet esimerkiksi hinnassa ja käytön helppoudessa.

3.2 Mallinnusohjelmat

3D-mallinnuksella luodaan pelimaailman eri objektit, hahmot ja ympäristöt. 3D-mallinnuksella tarkoitetaan minkä tahansa fyysisen objektin virtuaalisen matemaattisen esityksen kehittämistä kolmiulotteisesti simuloidun ohjelmiston sisällä käyttämällä tietokonetta ja erityisiä mallinnusohjelmia luomisen apuna. (Niklaus.)

Kehittäjä luo mallin muokkaamalla virtuaalisessa tilassa kärkipisteitä (vertice, jatkossa pisteet), pintoja sekä reunoja. Pisteiden avulla ohjelma muodostaa tietyn muotoisen meshin. Mesh eli pisteiden kertymä tilassa toimii 3D-mallin perustana. Mallinnusohjelma luo 3D-mallille muodon mallin pisteiden antaman tiedon pohjalta 3D-ruudukkoon. Mallin pinnat syntyvät, kun mallinnusohjelma yhdistää ohjelman sisällä olevat pisteet monikulmioiksi eli polygoneiksi. Polygoneina toimii yleisesti neliöt tai kolmiot. Kehittäjä tekee syntyneen meshin polygoneja muokkaamalla mallista halutunkaltaisen pisteitä, polygonin reunoja tai polygonin sivua muokkaamalla. Polygonit on mahdollista luoda automaattisesti ohjelman sisällä tai manuaalisesti luomalla. (Niklaus.)

3D-mallin voi luoda myös monesta mesh-osasta, jotka voidaan mallin viimeistelyn yhteydessä yhdistää toisiinsa. Esimerkiksi taloa mallintaessa kehittäjä voi ensin mallintaa osan talon ulkoseinästä, mitä kehittäjä voi monistaa peräkkäin luodakseen yhden yhtenäisen seinän, jos talon seinä käyttää samanlaista materiaalia. Seurauksena 3D-mallin tekoon kuluva aika vähenee, koska kehittäjän ei tarvitse mallintaa ulkoseinän jokaista osaa erikseen.

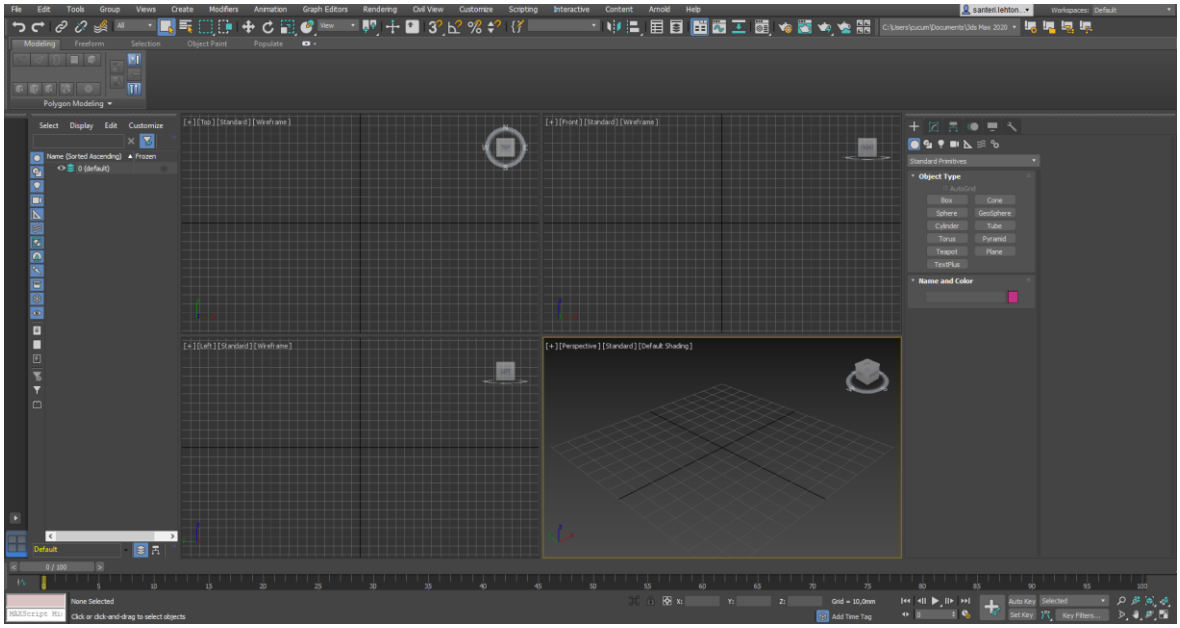
Ennen mallin luontia kehittäjän on hyvä tarkistaa, miltä malli näyttää renderöidyssä tilassa. Renderöidyllä tilalla tarkoitetaan näkymää, jossa mallin kaikki ominaisuudet ovat käytössä. Renderöidyssä tilassa malli näkyy tavalla, jolla pelaaja näkee mallin pelimaailman sisällä. Mallista tarkastetaan esimerkiksi näyttävätkö mallin materiaalit halutunlaiselta ja mallin sivut tarkastetaan. Jos materiaalin tekstuurikuva ei ole tarpeeksi tarkka, mallin materiaali näyttää

sumealta ja epätarkalta. Jos kaksi sivua ovat vastakkain samassa asemassa, renderöidyssä tilassa pinnat värähtelevät molempien sivujen välillä.

Mallin luomisen jälkeen kehittäjä tallentaa mallin pelimoottorille sopivaan tiedostoformaattiin, kuten .fbx formaattiin. Mallin tekstuurit tallennetaan sopivaan kuvaformaattiin, kuten .png formaattiin. Malli sekä mallin käyttämä tekstuuripohja liitetään pelimoottorin sisälle projekti-kansioon, josta malli voidaan lisätä pelimaailmaan.

3.2.1 3DS Max

3DS Max on 3D-mallinnus- ja renderöintiohjelma suunnittelun visualisointiin, peleihin ja animaatioihin (Kennedy 2020). Ohjelma julkaistiin alun perin vuonna 1996 3D Studio -nimellä. Ohjelma käyttää mallinnuksessa polygoni-pohjaista mallinnusta. 3DS Max on käytettävissä Windows-pohjaisille järjestelmille. Kuvassa 11 näytetään 3DS Maxin käyttöliittymä.



Kuva 11. 3DS Maxin käyttöliittymä

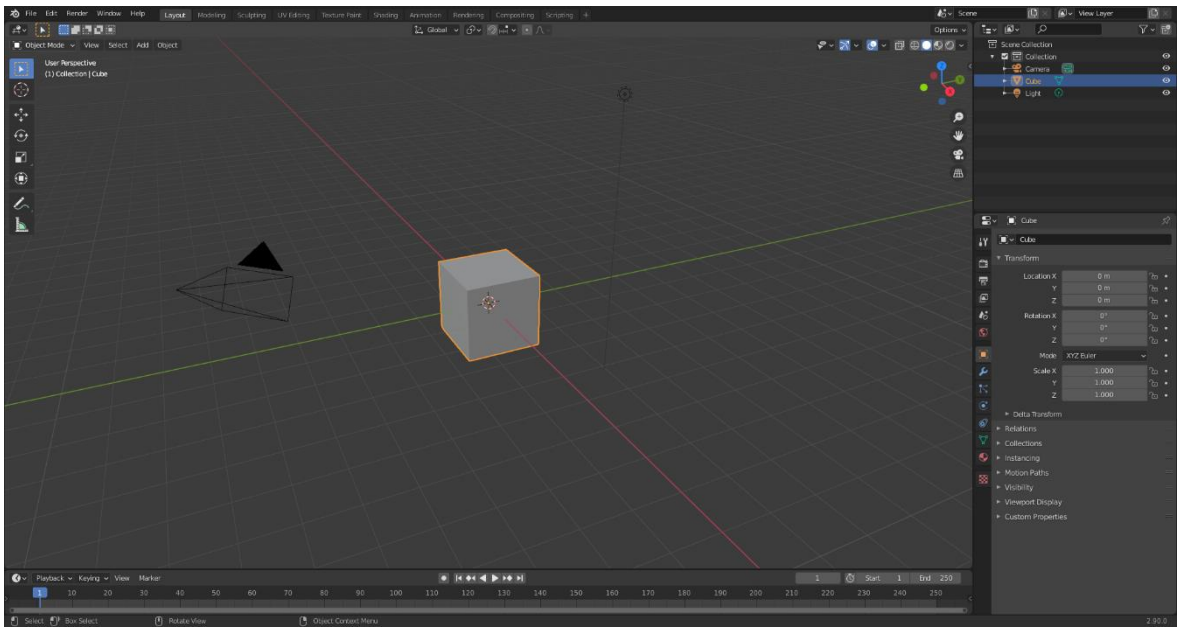
Kehittäjän avatessa 3DS Maxin ohjelmaan ilmestyy neljä eri ikkunaa, jotka kuvaavat projektin alkupistettä neljästä eri kuvakulmasta. Kuvakulmien näkökenttää voi muokata ohjelmassa valmiiksi sisältyvistä kuvakulmista tai itse manuaalisesti kuvakulmaa siirtämällä halutunlaiseksi. Kuvakulmista voi myös säätää, miten mallit näkyvät eri ikkunoissa. Esimerkiksi kehittäjä voi määrittää ikkunalle tilan, jolla mallit näkyvät rautalankamallina ilman pintoja.

Kuvakulmien vasemmalla puolella näkyy projektin layer-tasot, joihin lisätään projektin sisäiset muodot. Kuvakulmien oikealla puolella on lista eri muodoista, jotka voidaan liittää projektiin. Muotojen lisäksi oikealla on myös erilaisia asetuksia muotojen muokkaukseen. Esimerkiksi kehittäjän asettaessa yksinkertaisen laatikkomuodon kehittäjä voi laittaa muodossa Editable Mesh -nimisen muuttujan päälle. Näin kehittäjä pystyy muokkaamaan tai poistamaan kokonaan laatikon eri kärkipisteitä tai pintoja.

Kuvakulmien yläpuolella on erilaisia muokkauskomentoja. Näiden avulla kehittäjä voi halutessaan esimerkiksi liikutella, skaalata tai kiertää projektin sisällä olevia muotoja. Kuvakulmien yläpuolella sijaitsee myös esimerkiksi renderöinti-ikkuna, jolla voi tarkistaa miltä malli näyttäisi valmiissa tilassa materiaalien kanssa.

3.2.2 Blender

Blender on vuonna 1998 julkaistu open source -pohjainen mallinnusohjelma. Blender tukee koko 3D-kehittämisen pipelinea. Kuten 3DS Max, myös Blender käyttää polygoni-pohjaista mallinnusta. Toisin kuin 3DS Max, Blender on ilmaiseksi saatavilla oleva ohjelma. Blender on myös saatavilla Windows-käyttöjärjestelmän lisäksi Linux- ja MacOS-käyttöjärjestelmillä. Kuvassa 12 on kuvattuna Blenderin käyttöliittymä.



Kuva 12. Blenderin käyttöliittymä

Blenderin avautuessa ohjelmaan syntyy automaattisesti kuutiomuoto, mikäli kehittäjä valitsee ohjelman avautuessa general-projektin. Tilaan syntyy myös kameraobjekti sekä valoobjekti. Ikkuna näyttää objektin tyhjässä tilassa gridillä. Ikkunan oikeassa yläkulmassa sijaitsee asetuksia, millä ikkunan ulkonäön asetuksia voi muokata. Vasemmassa yläkulmassa on ylimpänä valinta-asetuksia. Valinta-asetusten alapuolella sijaitsee muokkaustilojen valikko.

Ikkunan oikealla puolella sijaitsee Scene Collection -ikkuna sekä asetusikkuna projektin eri osiin. Scene Collectionista löytyy kaikki projektissa käytettävät objektit. Asetusikkunasta löytyy asetuksia projektin eri osiin.

Näyttöikkunan alapuolella sijaitsee aluksi animaatioissa käytettävä Timeline. Timelinesta näkyy animaation aloitus- ja lopetuspiste, sekä animaatiossa käytettävien keyframejen sijainnit. Timelinen yläpuolelta löytyy painikkeet, joilla saadaan animaatio liikkeelle. Timelinen voi kehittäjä halutessaan vaihtaa itselle hyödyllisemmälle ikkunalle.

3.3 Pelimoottorit

Pelimoottorilla tarkoitetaan ohjelmiston kehitysympäristöä, jonka tarkoituksena on asetusten sekä konfiguraatioiden avulla optimoida ja yksinkertaistaa videopelien luontia eri koodikielien avulla (arm glossary). Pelin eri komponenttien luomisen jälkeen pelin eri osat yhdistetään toimivaksi kokonaisuudeksi pelimoottorin sisällä. Pelimoottorit sisältää omat renderöinti-, fysiikka-, ääni- ja animaatiomoottorit muiden ominaisuuksien lisäksi. Pelimoottoreiden ensisijainen koodikieli saattaa vaihdella eri moottoreita käyttäessä.

3D-mallin luomisen jälkeen malli viedään pelimoottorin sisälle sopivassa tiedostomuodossa projektikansioon. Pelimoottorin sisällä kehittäjä voi vielä tarkistaa, miltä 3D-malli näyttää pelimaailman sisällä. Mallista tarkastetaan mahdollisesti esimerkiksi tekstuurien laatu, päällekkäin sijaitsevat tasot ja pelimoottorin sisäisen valaistuksen vaikutus mallin materiaaliin. Pelimoottorin sisällä voi myös määrittää mallille CB:it, mikäli mallille ei ole CB:ia implementoitu vielä. Malli sijoitetaan pelimoottorin sisällä pelimaailmaan paikkoihin, jossa mallia tarvitaan. Pelimoottorin sisällä nähdään myös kuinka paljon luodut 3D-mallit käyttävät tietokoneen laskentatehoja.

3.3.1 Unreal Engine

Unreal Engine on Epic Gamesin vuonna 1998 julkaistu pelimoottori. Alun perin Unreal Engine kehitettiin ensimmäisen persoonan ammuntapelien kehittämiseksi, mutta myöhemmin pelimoottorilla on kehitetty monen muun genren pelejä, kuten tappelupelejä ja RPG-pelejä.

Unreal Engine käyttää C++ koodikieltä. Kuvassa 13 näytetään Unreal Enginen käyttöliittymä First Person Shooter -alkuasetuksella.

Kuva 13. Unreal Enginen käyttöliittymä

Unreal Enginen avautuessa ohjelman keskelle avautuu näyttöikkuna, joka näyttää projektin sisäisen pelimaailman. Ikkunan vasemmassa yläkulmassa sijaitsee asetuksia näyttöikkunan käytön muuttamiseen. Kuvatilan asetusten oikealta puolelta löytyy Unreal Enginen versiot Move-, Rotate- ja Scale-työkaluista sekä Snap-asetuksia. Snap-asetuksilla malleja liikuttaessa mallit liikkuvat ruudukon pisteiden mukaan.

Näyttöikkunan oikealla puolella on World Outliner – ja Details -ikkunat. World Outliner -ikkunasta näkyy projektissa käytössä olevat ominaisuudet. Details-ikkunasta näkyy asetukset aktiivisena valitulle ominaisuudelle.

Näyttöikkunan alapuolella on Content Browser. Content Browserista näkee kaikki projektikansion sisällä olevat ominaisuudet. Ohjelman ja näyttöikkunan vasemmalla puolella sijaitsee Place Actors -ikkuna, jossa on valmiiksi Unreal Enginen mukana tulevia ominaisuuksia.

Näyttöikkunan yläpuolella löytyy työkaluja projektia varten. Yksinkertaisimmillaan työkalujen avulla voi tallentaa aktiiviselle tasolle tehdyt muutokset. Työkaluista löytyy myös Marketplace, josta kehittäjä voi tarvittaessaan ladata jo valmiiksi luotuja ominaisuuksia ilmaiseksi tai maksamalla. Play-painiketta painamalla kehittäjä voi kokeilla projektin aktiivisen tason pelaamista.

3.3.2 Unity

Unity on Unity Technologies:ien vuodesta 2005 asti käytössä oleva pelimottori. Unity on suosittu pelimoottori varsinkin aloittelevien kehittäjien keskuudessa helppokäyttöisyytensä sekä monipuolisuutensa ansiosta. Unity käyttää myös C# koodikieltä. Kuvassa 14 näytetään Unityn käyttöliittymää.

Kuva 14. Unityn käyttöliittymä

Unityn avattaessa keskelle ruutu avautuu näyttöikkuna, joka näyttää pelimoottorin projektin sisäisen maailman. Valmiiksi luodun projektin skene sisältää kameraobjektin sekä valaistusobjektin. Ikkunan sisällä yläpuolella sijaitsee näyttöikkunalle eri asetuksia näyttöikkunan katselun muokkaamiseksi, kuten rautalankamallin käyttö. Asetusten oikealla puolella on painikkeita, jotka muokkaavat näyttöikkunaa eri tavoin.

Näyttöikkunan yläpuolella on myös kolme painiketta: Scene-, Game ja Asset Store -painikkeet. Näyttöikkunan vasemmalla puolella sijaitsee Hierarchy-ikkuna. Hierarchy-ikkunasta näkyy aktiivisena olevan skenen nimi, sekä skenen sisällä sijaitsevat ominaisuudet. Hierarchy-ikkunan yläpuolelta löytyy Unityn versiot Move-, Rotate- ja Scale-työkaluista. Näiden lisäksi löytyy Hand-työkalu, Rect-työkalu sekä työkalu jolla voi suorittaa Move-, Rotate- sekä Scale-komentoja.

Näyttöikkunan alapuolella sijaitsee Project-ikkuna. Project-ikkunasta löytää projektin sisäiset ominaisuudet valituissa kansioissa Asset-osassa. Näyttöikkunan oikealla puolella sijaitse Inspector-ikkuna, joka näyttää aktiivisena valitun ominaisuuden asetukset.

4 CASE

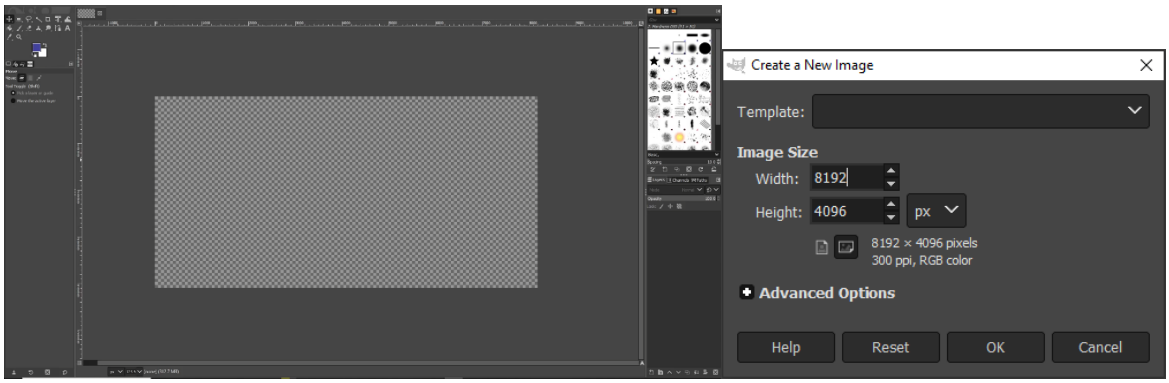
4.1 Tavoite

Projektin tavoitteena on luoda 3D-malli kaksikerroksisesta rivitalosta, joka kuluttaa mahdollisimman vähän tehoja. Malliin lisätään atlas-tekstuuriin pohjautuva materiaali sekä CB. Malli luotiin 3DS Max -ohjelmalla ja atlas-tekstuuri luotiin Gimp-ohjelman avulla.

Työssä keskitytään mallin ulkoseinien ulkonäköön. Talon sisätiloihin on tarkoituksena lisätä yksinkertaiset sisäseinät sekä lattiatekstuurit. Talon pohjalle tehdään myös talon perusta. Taloon lisätään myös yksityiskohtia, kuten vesirännit.

4.2 Tekstuurin luonti

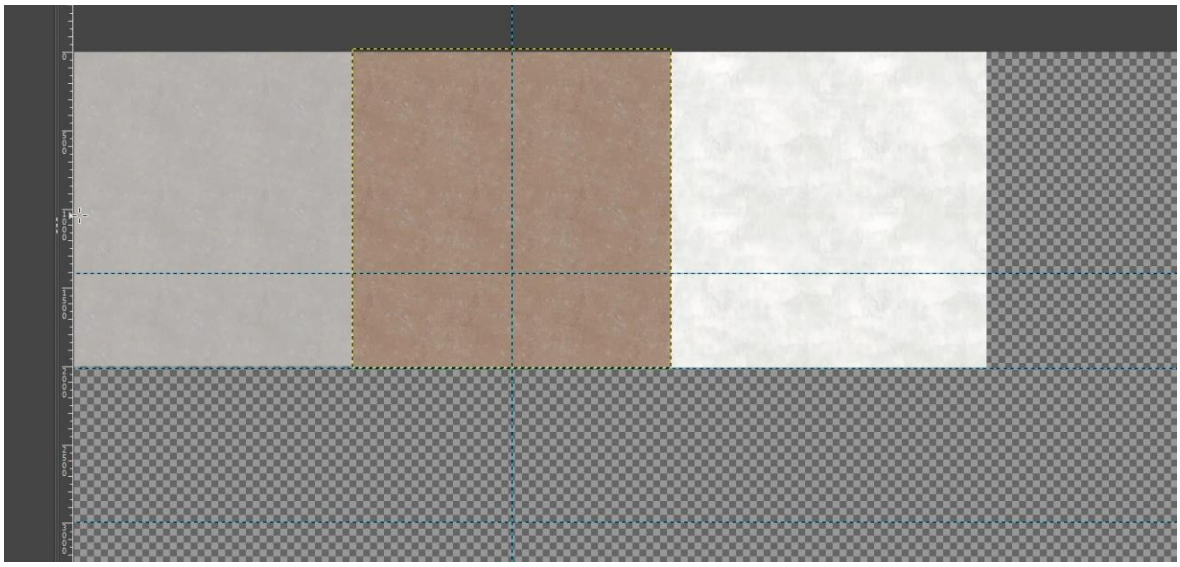
Gimp-ohjelmassa tekstuurikuvan kooksi valittiin 8192x4096 pikselin resoluutio. Molemmat arvot noudattavat power of two -ulottuvuuksia ja kuvan koko on tarpeeksi iso tarpeellisen tarkan tekstuurikuvien tuomiseen. Kuva viedään .png -formaattiin, minkä jälkeen kuva voidaan lisätä 3DS Maxin sisälle materiaaliin pohjakuvaksi. Kuvassa 15 näytetään tekstuuri ennen kuvien lisäystä sekä tekstuurin luontiasetukset.



Kuva 15. Tekstuurin pohja ja alkuasetukset

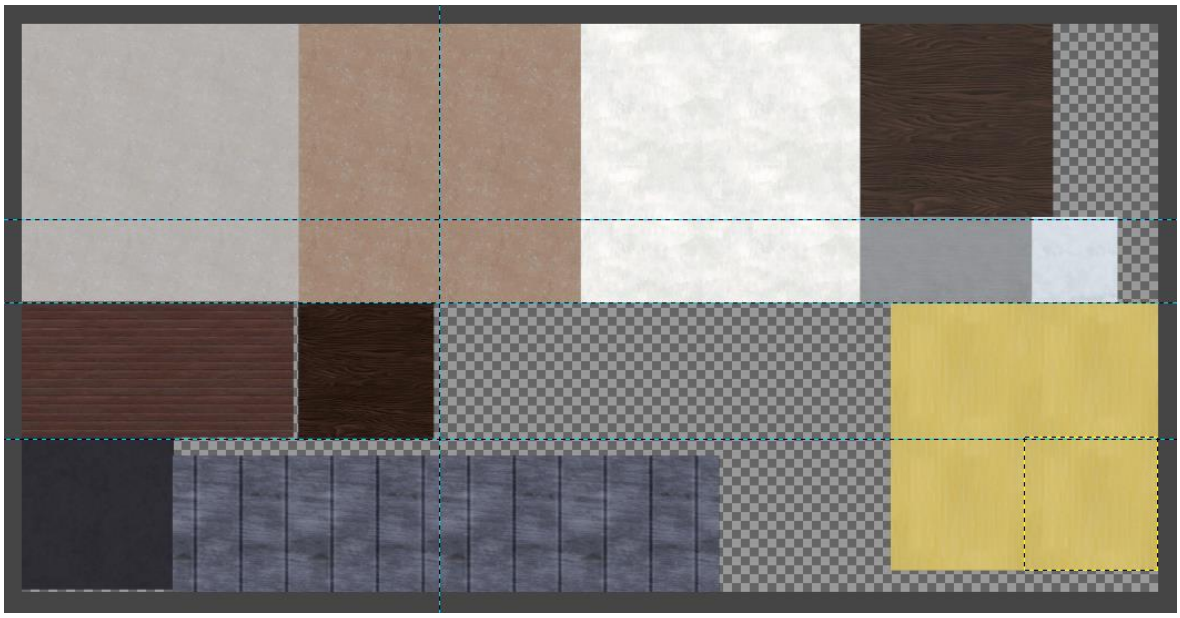
Seuraavaksi kuvapohjaan lisättiin talon seinässä käytettävä betonitekstuuri. Kuva on ladattu internetistä ja kuvalla on CC0 1.0 Universal -lisenssi, eli kuva on vapaasti käytettävissä sekä muokattavissa tekijänoikeuksien puolesta. Kuva monistettiin moneen kertaan luoden yhden suuremman kuvan. Kyseinen betonitekstuuri veikin suuren osan tekstuurin pohjakuvan pinta-alasta, mutta suuren pinta-alan ansiosta kuva tuottaa tarkemman näköisen tekstuurin.

Talomallin sivuseinään on suunniteltu eriväriset seinät. Värinä on mietitty ruskeaa väriä. Internettiä selaamalla ei löydetty halutun väristä betonitekstuuria, joten tekstuurin värit luotiin Gimp:illä. Edellisesti mainitusta betonitekstuurista otettiin kopio ja kopion väriasetuksia muokattiin. Kopiosta muokattiin color balance -työkalussa värin kohokohtia ja keskisävyjä niin, että betonitekstuuri sai ruskean sävyn. Tämän jälkeen betonitekstuurin kopion kirkkautta säädettiin levels-työkalulla hieman tummemmaksi. Tuloksena betonitekstuurin kopion väri muutettiin harmaasta ruskeaksi. Kuvassa 16 näytetään molemmat betonitekstuurit sekä mallin sisäseinien käyttämä tekstuuri.



Kuva 16. Tekstuurit tekstuuripohjassa.

Seuraavaksi tekstuuripohja vietiin .png -formaattiin ja .png -kuvana tekstuuri liitettiin 3DS Maxissa materiaaliin. Loput tekstuuripohjan tekstuureista luotiin edellä mainituilla tavoilla. Työn uudelleenvieminen päivittää myös mallinnusohjelman sisäisen materiaalin tekstuuripohjan. Kuvassa 17 näytetään lopullinen atlas-tekstuuri. Kuvassa olevat viivat ovat merkkausviivoja, jotka eivät lopullisessa kuvassa näy.

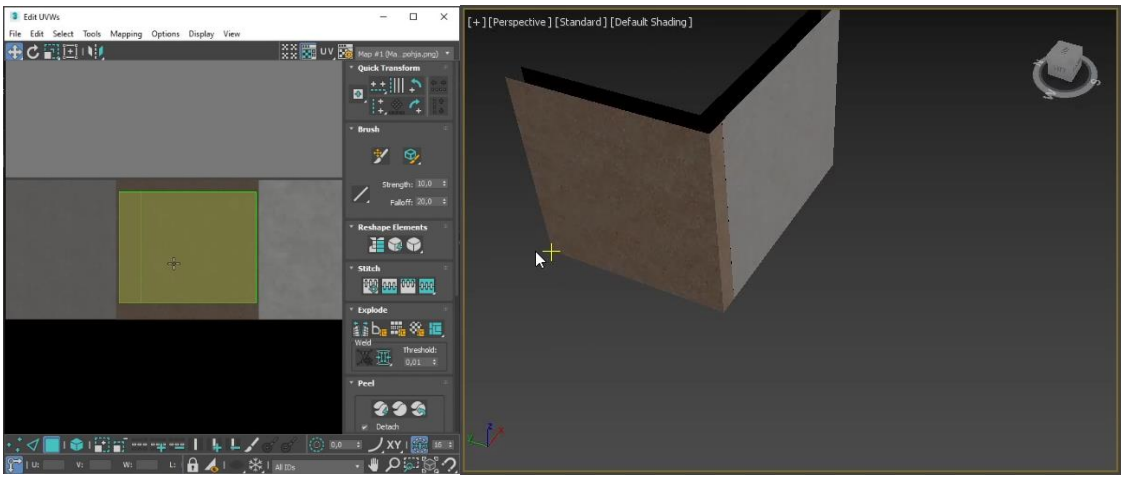


Kuva 17. Atlas-tekstuuri.

4.3 Mallin luonti

Uuden tiedoston aloituksen jälkeen lisättiin 3DS Maxiin laatikkomuoto. Laatikon sivujen määrittämisen jälkeen muodosta poistettiin ylimääräiset sivut, jotka eivät näy lopullisessa mallissa. Ylimääräisten sivujen poisto suoritettiin lisäämällä muotoon Edit Mesh -muuttuja, jolla voidaan valita muodosta yksittäisiä sivuja, -pisteitä tai -reunoja. Koska tämä laatikkomuoto on pohja ulkoseinälle, muotoon jäi sivujen poiston jälkeen vain kaksi vastakkaista pintaa. Mallin etu- ja takaseinälle tehtiin myös ikkuna- sekä oviaukot ProBoolean komennolla. Mallin vasemmat ja oikeat seinät ovat umpinaisia sekä niissä tullaan käyttämään eriväristä tekstuuria.

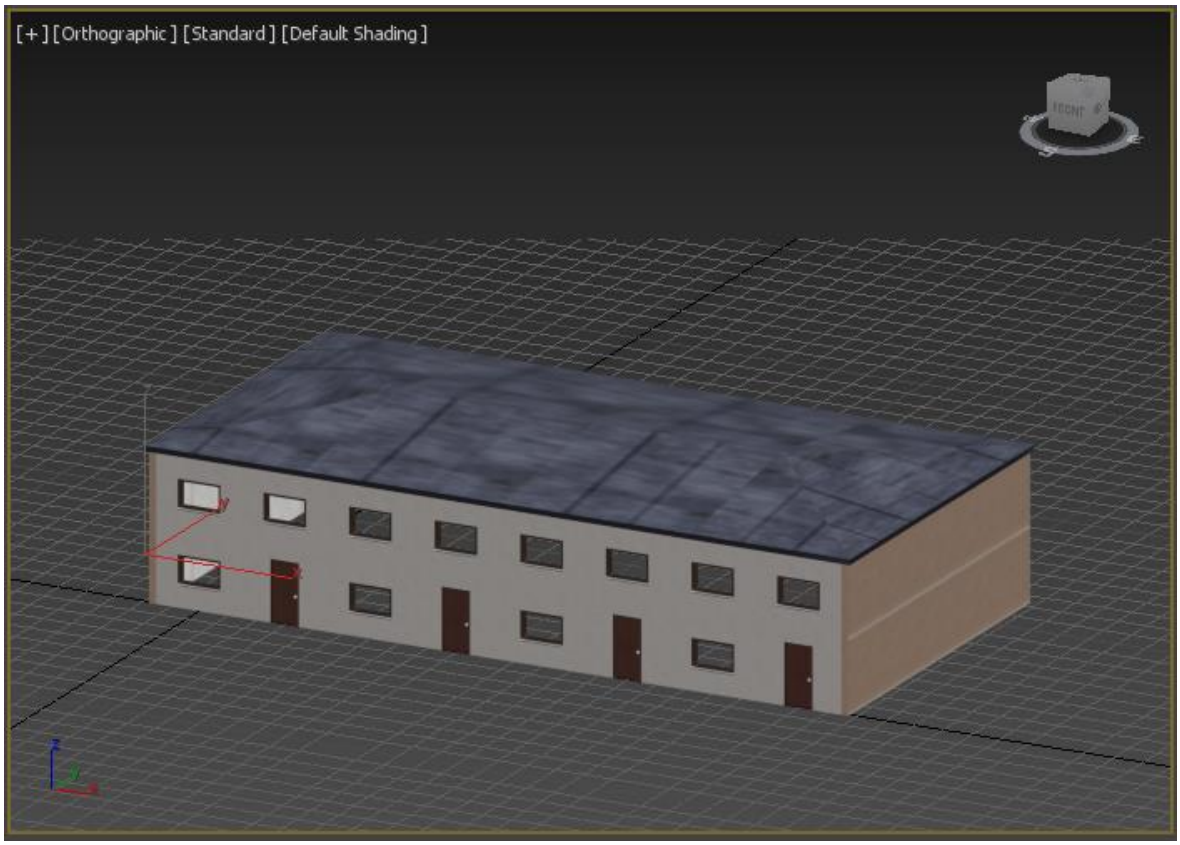
Ulkoseinien muotojen luomisen jälkeen tehtiin projektille materiaali. Materiaaliksi valittiin Standard Material ja bitmapiksi valittiin aikaisemmin luotu atlas-tekstuuri. Materiaalin tehtyä lisättiin ulkoseiniin materiaali sekä Unwrap UVW -muuntaja. Muuntajan avulla mallin sivuihin liitettiin oikeat tekstuurinosat atlas-tekstuurin kuva-alueella. Kuvassa 18 esitetään ensin sivuseinän pintojen sijainti materiaalin tekstuurissa Unwrap UVW -muuttujassa ja toiseksi etu- sekä sivuseinän ulkonäkö.



Kuva 18. Unwrap UVW ja ulkoseinät

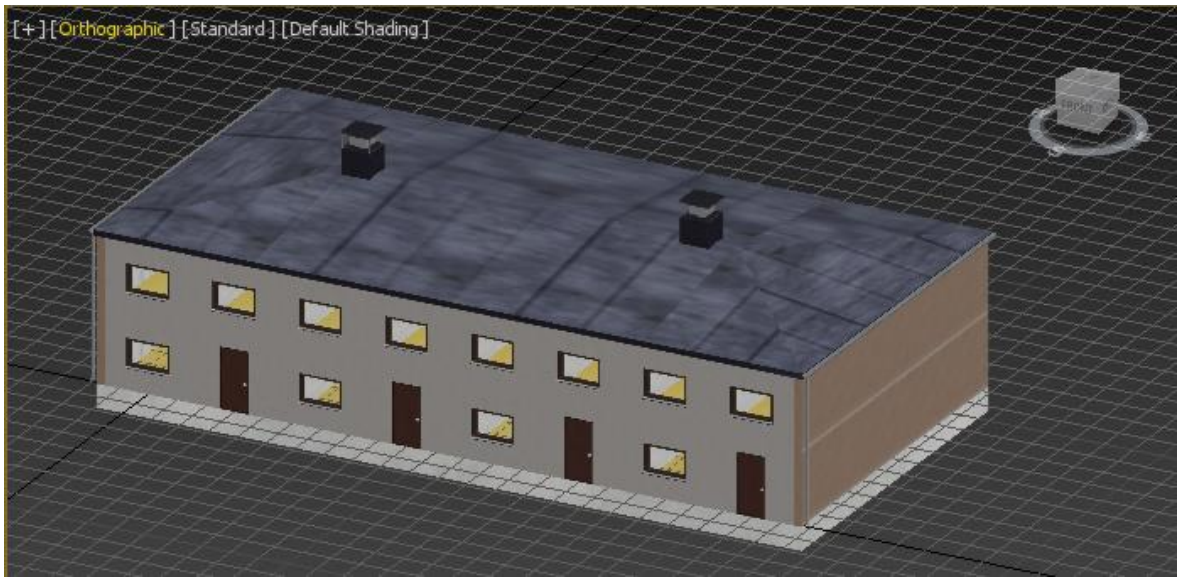
Ennen seinien monistamista ikkuna- ja oviaukon omaaviin seiniin tehtiin ikkunankarmet sekä ovenkarmet. Molemmat karmet luotiin pitkälti samalla periaatteella, kuin seinätkin: laatikkomuotoja oli vain enemmän. Oviin riitti neljä laatikkoa, jotka kiinnitettiin aktivoimalla Snap-asetus Move-työkalussa. Asetuksen avulla muotoja pystyi kiinnittämään yhteen esimerkiksi muotojen pisteiden mukaan. Ikkunankarmet luotiin samalla tavalla, mutta laatikoiden lukumäärää lisättiin. Ikkunankarmeihin lisätään myös alapelti, joka tehtiin valitsemalla reuna Edit Mesh -muuttujan avulla ja jatkamalla reunaa karmien ulkopuolelle. Lopuksi karmeihin lisätään materiaalit. Karmien luomisen jälkeen seinät sekä karmit monistetaan, ja liitetään yhteen Snap-asetuksen avulla.

Ulkoseinien luomisen jälkeen tehtiin mallille katto. Ensin lisättiin laatikkomuoto ulkoseinien päälle ”välikatoksi”. Seuraavaksi lisättiin välikaton päälle ”apulaatikoita”, joiden tarkoitus on avustaa Snap-asetuksen kanssa. Tämän jälkeen lisättiin katon muodostava laatikkomuoto. Laatikon ylimääräisten sivujen poistamisen jälkeen muodon kärkipisteitä siirrettiin apulaatikoiden kulmiin Edit Mesh -muuttujan avulla. Tuloksena laatikosta saatiin kolmion muotoinen kattolaatta. Kattolaatat monistettiin ja muokattiin siten, että laatoista muodostui katto. Lopuksi kattolaatat liitettiin yhdeksi muodoksi Collapse-toiminnolla ja kattoon lisättiin materiaali. Kuvassa 19 näytetään malli ovien- sekä katon lisäyksen jälkeen.



Kuva 19. Seinät ja katot

Talomalliin tehtiin tämän jälkeen perusta, sisätilat sekä yksityiskohdat. Yksityiskohtina toimivat talossa vesirännit sekä talon katon objektit. Loput osat tehtiin edellä mainituilla menetelmillä. Vain vesirännien teossa muistettiin, että rännin sisäosat täytyy tehdä kaksikerroksisena. Muuten rännin sisäosat näkyvät Unreal Enginen sisällä näkymättöminä. Mallin luomisen jälkeen malli vietiin Export-toiminnolla .fbx formaattiin. Kuvassa 20 näytetään lopullinen talomalli 3DS Maxissa.

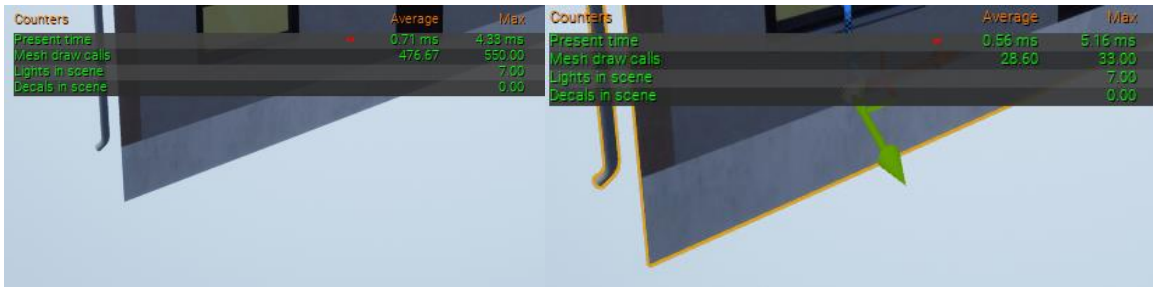


Kuva 20. Lopullinen malli

4.4 Mallin testaaminen

Mallin luomisen jälkeen aloitettiin uusi projekti Unreal Engine:ssä. Projektityypeistä valittiin First Person Shooter -pohja, jolla on valmiiksi ohjattava hahmo projektin alkaessa. Luodun mallin tiedosto vietiin projektin kansioon ja sitä kautta UE:n sisälle.

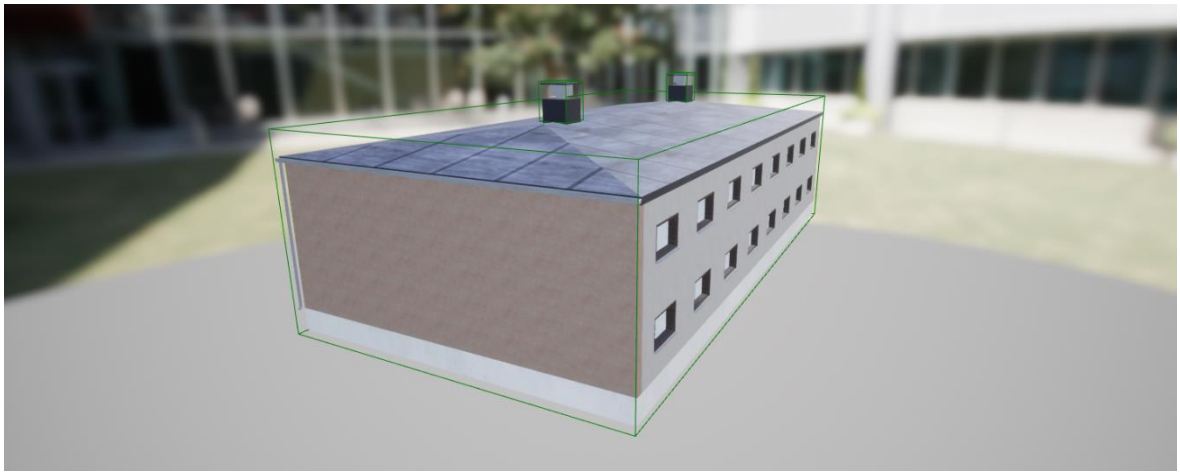
Ensimmäiseksi huomattiin, että 3DS Maxissa ei lopussa yhdistetty talon osia yhdeksi isoksi kappaleeksi. SceneRendering-tietoikkunan avulla nähtiin, että malli tuotti suuren määrän mesh draw calleja. Ongelma ratkaistiin käyttämällä 3DS Maxin sisällä Collapse-toimintoa mallin jokaiseen osaan, yhdistäen ne yhdeksi isoksi osaksi. Tämän jälkeen malli vietiin uudestaan Export-toiminnolla. Yhdistetty malli tuotti huomattavasti vähemmän mesh draw calleja UE:n sisällä. Kuvassa 21 näytetään mallin mesh draw callit. Vasen kuva näyttää draw callit ennen osien yhdistämistä ja oikea kuva näyttää draw callit yhdistämisen jälkeen.



Kuva 21. Mallien mesh draw call -arvot

Draw callien tarkastuksen jälkeen testataan, toimiiko mallin törmäystarkastus. First Person Shooter -esiasetuksessa on pelimaailmassa valmiiksi sijaitsevia muotoja, joissa on valmiiksi omat CB:it. Pallot sijoitettiin talomallin yläpuolelle, jonka jälkeen käynnistettiin projektin pelitila painamalla Play-painiketta. Muodot putosivat talon läpi, joten mallilla ei ole omaa CB:ia.

Muokkausnäkymässä avattiin Collision-valikosta Add Box Simplified Collision -asetus. Asetusta painamalla näkymään ilmestyi koko talomallin ympärille ilmestyvä suorakulmio, joka esitti syntynyttä CB:ia. Suorakulmion mittoja muutettiin hieman ja talon katon yksityiskohtiin lisättiin omat, pienemmät CB:it. Lopuksi malli tallennettiin Edit-tilassa. Uudessa testissä pudotetut muodot eivät tippuneet mallin läpi, vaan pysyvät mallin katolla. Kuvassa 22 näytetään malli Edit-tilassa CB:ien kanssa.



Kuva 22. Talomalli CB:in lisäämisen jälkeen.

3D-malli toimi hyvin. Mallin tuottamat mesh draw callit saatiin pidettyä vähäisinä ja mallin CB toimi pelimaailmassa. Materiaalin kanssa ei myöskään ollut ongelmia mallin siirtyessä 3DS Maxista Unreal Engineen.

5 Yhteenveto ja pohdinta

Tutkimuksen tavoitteena oli tutkia, miten pelinkehityksessä voi hyödyntää optimointia sekä miten 3D-mallia voidaan optimoida. Optimointi antaa kehittäjälle enemmän tilaa luoda suurempi määrä ominaisuuksia pelin sisälle, koska pelin yksittäinen ominaisuus kuluttaa laskentatehoa vähemmän. Optimaatiossa pitää myös muistaa, kuinka paljon pelin laskentatehoa halutaan vähentää pelin ulkonäön kustannuksella.

Projektin 3D-mallin luonnissa ei suuria ongelmia esiintynyt internetistä löytyneen aineiston avulla. Suurimmat ongelmat olivat ohjelmien kaatuminen sekä mallin tuominen Unreal Engineen. Ongelmia tuotti myös mallinnusohjelman sisällä vaihtoehtojen runsaus. Projektin 3D-osioon olisi myös voinut lisätä mallin, joka on luotu optimoimatta erojen löytämiseksi.

Materiaalien löytäminen oli myös suhteellisen helppoa löytyneen aineiston avulla. Toisaalta ilmaisesti saatavilla olevista kuvista ei yksilöllisiä saa, mutta talon mallinnukseen ne sopivat hyvin. Tekstuuripohjan teossa olisi mieluiten käytetty Adobe Photoshop -ohjelmaa, mutta ilmaiseksi saatavilla oleva GIMP-ohjelma toimi riittävän hyvin.

Projektin jatkokehityksessä keskittyisin tekemään talon sisätiloja yksityiskohtaisemmin. Huonekalujen lisääminen, portaiden sijoittaminen jahuoneiden tarkempi mallinnus olisivat oma haasteensa suorittaa mahdollisimman vähäisellä laskentatehon kulutuksella. Unreal Enginen sisälle voi myös luoda maailman, johon taloja voisi asettaa. Tällä voitaisiin testata monen eri talon mesh draw call -tuottoa.

Lähteet

Altvater, A. 2020. What is Code Profiling? Learn the 3 Types of Code Profilers. Stackify. Viitattu 4.6.2022. Saatavissa https://stackify.com/what-is-code-profiling/

Arm Developer. Triangle and polygon usage. Viitattu 29.4.2022. Saatavissa https://developer.arm.com/documentation/102695/0100/Triangle-and-polygon-usage

Arm Developer. Level of Detail – LOD. Viitattu 1.6.2022. Saatavissa https://developer.arm.com/documentation/102496/0100/Level-of-Detail---LOD

Evanson, N. 2020. How 3D Game Rendering Works: Lighting and Shadows. Techspot. Viitattu 18.5.2022. Saatavissa https://www.techspot.com/article/1998-how-to-3d-rendering-lighting-shadows/

Garney, B. & Preisz, E. 2011. Video Game Optimization. Saatavissa https://docplayer.net/169352662-Video-game-optimization.html

Game Dev Insider. Texturing and Materials. Viitattu 30.5.2022. Saatavissa https://gamedevinsider.com/making-games/game-artist/texturing-and-materials/

Hider, J. Real Time Rendering for Artists. Viitattu 4.5.2022. Saatavissa https://jesshiderue4.wordpress.com/real-time-rendering-an-overview-for-artists/

Kennedy, K. 2020. What is 3ds Max? – Simply Explained. All 3dp. Viitattu 10.5.2022. Saatavissa https://all3dp.com/2/what-is-3ds-max-simply-explained/

Lowrey, J. 2017. Basic Game Optimization techniques. Viitattu 3.5.2022. Saatavissa https://jarlowrey.com/blog/game-optimizations

Machkovech, S. 2019. Resident Evil 2 remake review: Beautiful, terrifying, and annoying. Ars Technica. Viitattu 20.5.2022. Saatavissa https://cdn.arstechnica.net/wp-content/uploads/2019/01/20190121210259_1-980x551.jpg

Merriam-Webster. Optimization. Viitattu 28.4.2022. Saatavissa https://www.merriam-webster.com/dictionary/optimization

Niklaus, L. What Is 3D modelling, And How Does It Work? Mount cg. Viitattu 9.5.2022. Saatavissa https://mountcg.com/what-is-3d-modeling/

Pan, M. 2017. GameEngineBook. Viitattu 29.4.2022. Saatavissa http://mikepan.com/GameEngineBook/text/08-Optimization.html

Reddit. 2020. Resident Evil 5 without the filter. Viitattu 20.5.2022. Saatavissa https://i.redd.it/llk754neji941.jpg

Sciutteri, M. 2016. Using a Texture Atlas to Optimize Your Game. Envato Tuts+. Viitattu 2.5.2022. Saatavissa https://gamedevelopment.tutsplus.com/articles/using-texture-atlas-in-order-to-optimize-your-game--cms-26783

Sloka-Frey, K. 2018. Code Optimizations for Game development: Basic Structures and Mindsets. Envato Tuts+. Viitattu 5.5.2022. Saatavissa https://gamedevelopment.tutsplus.com/tutorials/code-optimizations-for-game-development-basic-structures-and-mindsets--cms-30760

Torres Bonet, R. 2021. How to Use Occlusion Culling in Unity – The Sneaky Way. The Gamedev Guru. Viitattu 1.6.2022. Saatavissa https://thegamedev.guru/unity-performance/occlusion-culling-tutorial/

Unity Documentation. 2022. Optimizing draw calls. Viitattu 1.6.2022. Saatavissa https://docs.unity3d.com/2022.1/Documentation/Manual/optimizing-draw-calls.html

Unity Documentation. 2022. GPU Instancing. Viitattu 1.6.2022. Saatavissa https://docs.unity3d.com/Manual/GPUInstancing.html

Unreal Engine Documentation. Setting Up Collisions With Static Meshes. Viitattu 5.5.2022. Saatavissa https://docs.unrealengine.com/4.27/en-US/WorkingWithContent/Types/StaticMeshes/HowTo/SettingCollision/

Unreal Tips. 2019. What are Draw Calls. Viitattu 2.5.2022. Saatavissa https://unreal.tips/en/what-are-draw-calls/