Opinnäytetyö (AMK)

Tietojenkäsittely

2021

Tomi Parviainen

# 3D-MALLIEN SUORITUSKYVYN TESTAUS JA OPTIMOINTI ANDROID-MOBIILILAITTEILLE

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietojenkäsittely

2021 | 41

Tomi Parviainen

# 3D-MALLIEN SUORITUSKYVYN TESTAUS JA OPTIMOINTI ANDROID-MOBIILILAITTEILLE

Opinnäytetyön tavoitteena oli testata miten realististen 3D-mallien esittäminen Android-mobiililaitteilla vaikuttaa suorituskykyyn ja miten testattuja malleja voidaan optimoida.

Suorituskyvyn testaamista varten suunniteltiin ja luotiin testiohjelma Unreal Engine 4 - pelimoottorille. Suunnitteluvaiheessa päätettiin mitä testaamiseen tarvitaan ja miten testit tehdään. Testaamista varten päädyttiin käyttämään kolmea erityyppistä 3D-mallia, joista kaikista oli kolme esimerkkiä. Mallityypeiksi valittiin puun, ruohikon ja kiven mallit. Mallit päädyttiin luomaan ruudukkoon ja kuvaamaan ylhäältä päin, jolloin malleista saatiin ylhäältä päin kuvaavaa kameraa käyttäen näkyviin kerralla mahdollisimman paljon. Suorituskykytestissä mitattiin yhden ruudun piirtämiseen vaadittua aikaa ja tulokset vietiin taulukonlaskentaohjelmaan vertailua varten.

Tulosten perusteella aloitettiin optimointi puun malleista. Malleille löydettiin useita optimointitapoja joiden jälkeen päädyttiin tilanteeseen, jossa malleja voitiin harkitusti käyttää. Puiden mallin optimoinnin perusteella ruohon mallit testattiin uudestaan vaakatasossa ja niitä optimointiin tämän testin tuloksien perusteella. Ruohon mallia onnistuttiin optimoimaan vähentämällä ruohikon tiheyttä mallissa, samalla läpinäkyvien tekstuurien ja mallin käyttämien kolmioiden määrää vähentäen. Kivien mallien käyttäminen ja optimointi huomattiin helpoimmaksi ja siinä onnistuttiin nostamaan suorituskykyä yksinkertaistamalla mallia ja sen tekstuurikarttoja ilman suurta vaikutusta ulkonäköön.

Optimointien tulosten perusteella tehtiin vielä testitaso, jossa eri mallityyppejä ja optimointeja yhdisteltiin. Sama testi tehtiin vielä käyttäen optimoimattomia malleja, jonka jälkeen havaittiin selkeä kasvu suorituskyvyssä. Testitasossa havaittiin että tehdyistä optimoinneista oli hyötyä ja opinnäytetyön tavoitteessa on onnistuttu. Yhteisen testitason luomisessa käytettyjä optimointeja voi hyödyntää mobiilipeliä tehdessä.

ASIASANAT:

Videopelit, tietokonegrafiikka, kolmiulotteisuus

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Business Information Technology

2021 | 41

Tomi Parviainen

# TESTING AND OPTIMIZATION OF 3D-MODELS FOR ANDROID MOBILE DEVICES

The purpose of this thesis was to test how displaying realistic 3D models would affect performance on Android devices and how the tested models could be optimized.

For testing the performance, a testing program was planned and built using Unreal Engine 4. In the planning phase, it was first decided what was needed for the testing and how the tests would be conducted. Three different types of 3D models with three variations each were selectedto be used for testing. The chosen types of models were that of a tree, grass, and a rock. The models were generated in a square grid so when using a top-down camera, the greatest number of models could be shown at once. The performance test was measuring the time it would take to draw a single frame and the results were brought into a spreadsheet program for comparison.

Based on the results, the optimization was started from the tree model. Multiple working methods for optimization were found and based on the optimizations, the model type could deliberately be used. Based on the optimization of the tree model, the tests for the grass models were run again with a vertical camera and they were optimized based on these results. The grass model was optimized by reducing the number of total blades of grass in the model and at the same time simplifying the model and reducing the number of used triangles. The rock model was found to be the easiest to display and optimize and performance was improved by simplifying the model and its textures without a significant decrease in quality.

Based on the optimizations, a shared testing level was created by mixing different model types and optimizations. The same test was run again by using the unoptimized models and from these results, a significant increase in performance was noticed. The optimizations used in creating the shared testing level can be utilized when creating a mobile game.

KEYWORDS:

Video games, computer graphics, three dimensional

# SISÄLTÖ



# KUVAT

# KUVIOT



# TAULUKOT

# KÄSITTEET

| | |
|---|---|
| 1K-resoluutio | 1024 x 1024 kuvapisteen koko. |
| 2K-resoluutio | 2048 x 2048 kuvapisteen koko. |
| 3D-renderointi | 3D-grafiikan piirtäminen ruudulle tai tiedostoon. Voidaan suorittaa etukäteen, kuten animaatioelokuvissa tai reaaliaikaisesti, kuten peleissä. |
| 512-resoluutio | 512 x 512 kuvapisteen koko. |
| CAD | Computer aided design. Tietokoneavusteinen suunnittelu, jossa sovelluksen avulla luodaan suunnitelmia, kuten arkkitehtuurisia piirroksia. |
| Fixed-function | Grafiikkarajapinnan tapa sallia kehittäjälle vaikuttaa piirrettävään grafiikkaan valmiiksi luotujen funktioiden avulla. |
| LOD | Level of Detail. 3D-mallin tarkkuustaso, jota voidaan vaihtaa tarvittaessa suorituskyvyn, tai mallin laadun parantamiseksi. |
| NPC | Non-Player Character. Pelihahmo, jota ohjaa pelaajan sijasta tietokone. |
| Pikseli | Pikseli on yksi näytöllä näkyvä kuvapiste. Esimerkiksi terävä-piirtonäyttö esittää kerralla 1920 x 1080 pikseliä. |
| Pipeline | Koostuu ohjelmistotuotannossa ketjusta prosessoinnin osia, joissa seuraavan osan syöte koostuu edellisen osan tulosteesta. |
| Programmable | Grafiikkarajapinnan tapa sallia kehittäjälle vaikuttaa piirrettävään grafiikkaan ohjelmoitavien varjostimien kautta. |
| Rajapinta | Mahdollistaa sovellusten yhtenäistämisen pyyntöjen avulla, joilla ohjelmasta toiseen voidaan siirtää tai noutaa tietoja. |

| | |
|---|---|
| Varjostin | Ohjelma, joka sallii grafiikka pipelinen muokkaamisen. Ohjelma on kirjoitettu käytetyn grafiikkarajapinnan tukemalla ohjelmointikielellä. |
| Verteksi | Yksi piste, josta 3D-malli rakennetaan. Verteksit yhdistetään ja niistä luodaan kolmioita, joissa voidaan esittää tekstuureja. |

# 1 JOHDANTO

Mobiilipelit ovat nykyään iso osa pelimarkkinoista. Isot yritykset ovatkin sijoittaneet niihin ostamalla mobiilipelejä kehittäviä studioita, parhaimmillaan yli miljardien dollarien kauppahinnoilla, kuten kävi ruotsalaiselle pelistudio Mojangille, kun Microsoft osti sen 2,5 miljardin dollarin hintaan (Samit Sarkar 2014). Laitteiden kehityksen myötä 3D-pelien tekeminen on tullut helpommaksi ja korkeamman ruudunpäivitysnopeuden käyttäminen on tullut mahdolliseksi. Laitteita on kuitenkin markkinoilla todella suuri määrä erilaisilla suorittimilla, joten 3D-pelillä on mahdollisuus tavoittaa suurempi määrä käyttäjiä, jos se toimii myös vähemmän tehokkailla laitteilla. Erilaisten ominaisuuksien suorituskykyvaatimusten vertailulla voidaan löytää suorituskykyasetuksia, joilla peli saadaan näyttämään paremmalta tehokkaammilla laitteilla, samalla kuitenkin mahdollistaen pelaaminen myös heikommilla laitteilla. Monet suositut 3D-pelit, kuten Minecraft suoriutuvat tästä käyttämällä tyyliteltyjä ja vähemmän realistisia grafiikoita, jotka vaativat laitteiden grafiikkapiireiltä vähemmän suorituskykyä. Toiset pelit, kuten realistisista grafiikkaa käyttävä Playerunknown’s Battlegrounds, antavat pelaajalle mahdollisuuden valita grafiikka-asetuksia itse.

Opinnäytetyön aiheena on mobiililaitteilla OpenGL ES 3.1 -rajapinnalla (https://www.khronos.org/opengles/) 3D-renderoitavan grafiikan suorituskykyyn vaikuttavien ominaisuuksien vertailu ja optimointi. OpenGL ES -ohjelmointirajapinnasta valittiin versio 3.1, koska testiohjelman luomiseen käytettävä Unreal Engine ei versiosta 2.44 eteenpäin tue aiempia versioita ja versiolla 3.1 tavoitetaan 75,46 % Android-laitteiden käyttäjistä (Google 2021).

Opinnäytetyön tarkoituksena on vertailla nykyaikaisia grafiikkaominaisuuksia luomalla Unreal Engine -pelimoottorilla suorituskykyä testaava peli, jolla saadaan tallennettua raportteja analysointia varten. Analyysien perusteella testissä käytettyjä malleja optimoidaan ja testejä ajetaan uudestaan, kunnes tuloksia ei enää saada parannettua.

Lopuksi tulosten perusteella tehdään testin malleja käyttäen pieni pelitaso, joka on tarkoitus saada toimimaan siten, että sen ruudunpäivitysnopeus olisi vähintään 30 ruutua sekunnissa, tavoitteena kuitenkin noin 40 ruutua sekunnissa. Pelitaso luodaan ja testataan lopuksi myös optimoimattomilla malleilla, jolloin nähdään, kuinka paljon tehtyjen optimointien tulokset vaikuttavat pelikäytössä.

Opinnäytetyössä käytetään joitain englanninkielisiä termejä, joille ei löydy yleisesti käytettyä vastinetta suomen kielestä.

# 2 3D-GRAFIIKKA JA VIDEOPELIT

3D-grafiikkaa hyväksikäyttäviä kohteita löytyy nykyään joka puolelta. Realisen 3D-grafiikan esittämisestä on hyötyä useilla erilaisilla aloilla, aina satojen miljoonien liikevaihtoa tekevien yrityksien videopeleistä ja elokuvista pienen arkkitehtitoimiston suunnitelmien tekemiseen ja esittämiseen.

Iso osa nykyisen 3D-renderoinnin perustekniikoista on peräisin Utahin yliopistosta, vuonna 1965 David Evansin perustamasta tietotekniikan laitoksesta. 1970 luvun alussa DARPA:n alaisuudessa toimiva ARPA alkoi rahoittaa 3D-renderoinnin kehitystä yliopistossa ja tämän rahoituksen ansiosta on kehitetty esimerkiksi Gouraund, Phong ja Blinn -varjostustekniikat, tekstuurikartat sekä kaarevien pintojen osittaminen, joka mahdollisti pyöreiden pintojen esittämisen. Yliopistosta näistä Dave Evansin vetämistä opinnoista valmistuneiden perustamiin yrityksiin kuuluu tietokonegrafiikan uranuurtajia kuten Pixar, SGI ja Adobe. (Gdc 2000)

3D-grafiikkaa käytettiin ensimmäisen kerran videopelissä Atarin vuonna 1980 julkaisemassa kolikkopelissä BattleZone. Pelin grafiikat oli luotu vihreän värisillä viivoilla piirretyistä vektorigrafiikoista mustaa taustaa vasten ja pelissä oli mahdollista kääntyä 360 -astetta ja liikkua eteen ja taaksepäin kolmiulotteisen näköisessä pelimaailmassa. Ensimmäinen kotitietokoneille julkaistu 3D-peli oli vuonna 1981 Sinclair ZX81 -tietokoneelle julkaistu Monster Maze, jossa pelaajan tarkoituksena oli liikkua sokkelossa ja vältellä sokkelossa samaan aikaan liikkuvaa vihollista. (Jordan 2021)

Noin 90-luvun puolivälissä 3D-pelit nousivat valtavirtaan kun Sony julkaisi vuonna 1994 alusta lähtien 3D-grafiikan esittämistä varten suunnitellun pelikonsolinsa, Sony Playstationin. Sony Playstationia myytiin maailmanlaajuisesti sen 12-vuotisen elinkaarensa aikana yli 100 miljoonaa kappaletta ja se oli valtava kaupallinen menestys (Guinness World Records, 2018). Pelikonsolista julkaistiin myös pienemmille kehittäjille tarkoitettu Net Yaroze niminen versio, joka mahdollisti pelien kehittämisen Playstation konsolille PC:n avulla. Version 750 dollarin hintalappu mahdollisti laitteen hankinnan myös yksittäisille kehittäjille (Ed Smith 2015). Net Yaroze ei kuitenkaan noussut suureen suosioon, vaan sillä julkaistiin vain alle sata peliä, joista vain muutama kaupallisesti, kuten vuonna 1998 julkaistu Devil Dice (Cassidy 2019).

Ensimmäinen mobiililaitteella virallisesti julkaistu peli oli Tetris, joka oli sisällytettynä Tanskalaisen puhelinvalmistajan, Hagenukin, vuonna 1994 julkaistuun MT-2000 matkapuhelimeen (Todorov 2014). Mobiilipelit matkapuhelimilla eivät kuitenkaan alkaneet nousta suosioon ennen kuin Nokia julkaisi vuonna 1997 matopelin Nokia 6110 puhelimen mukana (Angelos 2021). Kaksi vuotta myöhemmin julkaistiin Nokia 7110, joka oli ensimmäinen matkapuhelin, joka tuki WAP-protokollaa (http://www.openmobilealliance.org/wp/Affiliates/WAP.html), joka mahdollisti rajoitetun verkkoselaamisen ja pelien pelaamisen WAP-selaimen kautta (Wright 2016a). Myöhemmin sama protokolla mahdollisti mobiilipelien ja muun lisäsisällön ostamisen puhelimeen suoraan puhelinverkon tarjoajalta tai kolmannen osapuolen toimijalta. Sisältö ostettiin usein tekstiviestien avulla, jolloin ostaminen oli usein helppoa vain paikalliselta tarjoajalta. Tämä vaikeutti pienempien mobiilipelien löydettävyyttä huomattavasti. Laitteiden käyttämien ohjelmointikielien vaihtelevuus laitevalmistajien ja mallien kesken teki myös pelien kehittämisestä vaikeaa (Wright 2016b).

Vuonna 2007 moni asia muuttui, kun Apple julkaisi ensimmäisen iPhone älypuhelimensa, ja myöhemmin ennen iPhone 3G:n julkaisua App Storen, jossa kuka tahansa pystyi myymään omaa sovellustaan tai peliään. Kehittäjät ottivat kauppapaikan hyvin vastaan ja puolen vuoden kuluttua julkaisusta oli siellä jo yli 15 000 sovellusta, joita oli ladattu yhteensä 500 000 000 000 kertaa (Myslewski 2009). Kehitys tästä eteenpäin oli nopeaa ja 10 kuukautta myöhemmin kauppapaikalla oli jo yli 100 000 sovellusta, joita oli ladattu yli 2 000 000 000 kertaa (Chen 2009).

Vuonna 2006 mobiililaitteet alkoivat jo olla sen verran tehokkaita, että 3D-pelien luominen alkoi olla mahdollista. Teknologiaa käytettiin kuitenkin yleensä varsin rajatusti, kuten suositussa Fruit Ninja pelissä pelaajan tuhoamaksi tarkoitettujen hedelmien esittämiseen. Suosituimmat pelit kuten 2009 julkaistu Angry Birds käyttivät pelkkää 2D-grafiikkaa. Vuonna 2010 Epic Games julkaisi täysin 3D-grafiikkaa käyttäen luodun Infinity Blade pelinsä, joka menestyi mobiilimarkkinoilla varsin hyvin (Edge, 2013).

Nykyään mobiililaitteet ovat jo varsin tehokkaita, sisältäen usein vähintään neljäytimisen prosessorin ja ainakin prosessoriin sulautetun grafiikkasuorittimen, joka mahdollistaa nopean 2D- ja 3D-grafiikan piirron. Tämä on mahdollistanut esimerkiksi alun perin PC:llä menestykseksi nousseen 3D-räiskintäpelin, PUBG Mobilen, nousta myös yhdeksi maailman suosituimmaksi mobiilipeliksi. Peli on erityisen suosittu Intiassa, missä halpojen älypuhelimien hankkiminen oli mahdollista (Acharya 2019). Isot pelivalmistajat ovat

myös huomanneet mobiilipelien suosion ja tällä hetkellä 10:stä pelatuimmasta PC-pelistä, on 5:stä tehty myös älypuhelinversiot (Newzoo 2021).

## 2.1 OpenGL ES kehitys maailman käytetyimmäksi grafiikkarajapinnaksi

OpenGL ES:n historia alkaa vuodesta 2003 ja OpenGL versiosta 1.3 johon OpenGL ES versio 1.0 perustuu. OpenGL ES on on tarkoitettu kevyemmäksi versioksi OpenGL -rajapinnasta, jonka päätarkoituksena oli mahdollistaa 2D- ja 3D-grafiikan piirtäminen sulautetuilla järjestelmillä. Sulautetuissa järjestelmissä ei tähän aikaan yleensä ollut erillistä matematiikkarinnakkaissuoritinta tuplatarkkuuden liukulukujen laskemiseen ja niiden tukemisen sijaan tässä versiossa tuettiin vain kiintopistematematiikkaa, jossa binääripisteestä bittejä siirrettiin oikealle 16 kertaa suurempien lukujen laskemista varten ilman desimaalin käyttöä (Blythe 2020). OpenGL ES 1.0 perustuu fixed function pipelineen, jossa kehittäjä käyttää valmiiksi luotuja funktioita esimerkiksi kameroiden, valaistuksen ja verteksien esittämiseen.

OpenGL ES 2.0 oli markkinoiden ensimmäinen mobiililaitteille tehty grafiikkarajapinta, josta löytyi tuki ohjelmoitaville varjostimille sitä tukevilla grafiikkapiireillä. Ohjelmoitavat varjostimet korvasivat tekstuurikartat mallien tekstuurien esittämisessä ja tarjosivat mahdollisuuden esittää monimutkaisempia tekstuureja useita tekstuurikarttoja yhdistellen ja varjostinmatematiikkaa käyttäen. OpenGL ES 2.0 perustuu programmable pipelineen, jossa kehittäjälle ei enää tarjota valmiiksi luotuja funktioita vaan ne piti luoda itse. Tämä sallii ohjelmoitavien varjostimien avulla myös mahdollisuuden hallita paremmin muiden osien, kuten valaistuksen esittämistä (Khronos 2021).

OpenGL ES 3.0 -versiossa kehittäjille tarjottiin uusia mahdollisuuksia paremman näköisen grafiikan esittämiseen ja uusia mahdollisuuksia optimoida suorituskykyä. Suuri lisäys oli tuki mallien esittämiseen instansseina, joka mahdollistaa useampien samanlaisien mallien esittämisen erilaisia ominaisuuksia käyttäen ilman useampien piirtokutsujen esittämistä. Aikaisemmin kun tuhannen samanlaisen mallin esittäminen eri sijainneissa olisi vienyt tuhat piirtokutsua, instansseja käyttämällä piirtokutsuja olisi vain yksi. Android-laitteilla kehittämistä helpotti lisäys uusille ETC2/EAC pakkausmuodoille, joiden tukeminen oli pakollista OpenGL ES 3.0 tuen lisäämiseksi. Aiemmin tekstuurien pakkausmuodoille ei ollut yhtenäistä standardia, vaan käytännössä jokainen laitevalmistaja tuki oman tyyppisiä tekstuurejaan, joka aiheutti vaivaa rajapintaa käyttäville (Khronos 2021).

Nykyään OpenGL ES:n suosiota heikentää hieman tuen poistuminen Applen iOS-käyttöjärjestelmästä versiossa 12.0, jolloin Apple siirtyi tukemaan vain omaa iOS 8.0-versiossa julkaistua Metal-rajapintaansa. Android-laitteiden osuus älypuhelinmarkkinoista oli kuitenkin tammikuussa 2020 mitattuna ollut 71,93 % (O' Shea 2021), jolloin myös OpenGL ES-rajapintaa tukevien laitteiden osuus on vähintäänkin tämän verran, koska mukaan voidaan lukea myös iOS-laitteet, joita ei voida päivittää enää versiosta 12.0 ylöspäin. Modernit pelimoottorit, kuten Unreal Engine ja Unity tarjoavat mahdollisuuden kääntää luotu peli sekä OpenGL ES -rajapintaa varten Android-laitteille, sekä Metal-rajapinnalle iOS-laitteita varten.

Android-laitteilla tuki OpenGL ES -rajapinnasta on ollut jo versiosta 1.0 ja tähän lisättynä sitä tukevat iOS-laitteet tarkoittavat sitä, että rajapinnan osuus mobiililaitemarkkinoista on suurempi kuin millään muulla grafiikkarajapinnalla. Google ei ole ilmoittanut aikeista tuen poistamiseen, vaikka tuki uudemmalle Vulkan-rajapinnalle nykyään löytyykin.

OpenGL ja OpenGL ES -rajapintoja kehittävä Khronos konsortiumi on julkaissut seuraavan sukupolven rajapinnaksi tarkoitetun Vulkan-rajapinnan (https://www.vulkan.org/), joka on tällä hetkellä versionumerossa 1.2. Vulkan-rajapinnan etuina on esimerkiksi pienempi prosessorin käyttö grafiikkaa piirtäessä (Kapoulkine 2018) ja muutos ajettaessa laskettavista varjostimista esilaskettuihin, jolloin yhdellä kertaa voidaan käyttää suurempaa määrää erilaisia varjostimia ilman samanlaisia ongelmia suorituskyvyn kanssa kuin ajettaessa laskettavien varjostimien kanssa. Tämä opinnäytetyö käsittelee tarkoituksella OpenGL ES -rajapintaa, koska Vulkan -rajapinnan tuki laitteilla on rajatumpaa ja tukea uusimmalle 1.2 -versiolle ei löydy vähän vanhemmista laitteista, mistä esimerkiksi Intian laitekannasta yli kolmasosa koostuu (Statcounter 2021). Tuki Vulkanille pitäisi löytyä kaikista laitteista, jotka tukevat OpenGL ES 3.1 -rajapintaa, mutta käytännössä tämä tarkoittaa tukea Vulkan rajapinnan ensimmäiselle versiolle, joka ei oman kokemukseni mukaan toimi aivan ongelmitta. Pelimoottorien kuten Unreal Enginen suurina etuina on kuitenkin se, että saman pelin voi kääntää tukemaan molempia rajapintoja, jolloin pelaajalle voidaan antaa mahdollisuus halutessaan käyttää rajapintaa tai tuki sille voidaan kehittäjän puolesta lisätä myöhemmin varsin helposti.

## 2.2 Unreal Enginen historia

Unreal Enginen historia alkaa noin viitisen vuotta ensimmäisen mobiilipelin, Tanskassa valmistetun Hagenuk MT-2000 puhelimelle julkaistun Tetriksen jälkeen. Alun perin

vuoden 1998 pelin Unreal kehittämiseen luotu Unreal Engine tuli heti pelaajien käyttöön, kun se julkaistiin pelin mukana. Pelin mukana julkaistulla versiolla pystyi korvaamaan pelin sisältöä omilla ja pelin NPC-objektien käyttäytymistä pystyi muokkaamaan UnrealScript-ohjelmointikielellä. Myöhemmin ensimmäiseen versioon lisättiin vielä tuki verkkopelille Unreal Tournament -peliä varten (Angel 2016).

Ensimmäinen Unreal Enginen versiolla 2 julkaistu peli oli Yhdysvaltain armeijan kanssa yhdessä tuotettu ilmainen monen pelaajan räiskintäpeli, America's Army, vuonna 2002. Pelimoottoriin oli lisätty tuki partikkeleille, yksinkertaiselle 3D-mallien muokkaukselle, fysiikkamoottori ja Matinee-työkalu epäinteraktiivisten segmenttien, kuten välivideoiden, luomiseen. Työkalut julkaistiin vielä saman vuoden aikana pelaajien käyttöön pelin Unreal Tournament 2003 kanssa (Angel 2016).

Unreal Enginen versio 3 julkaistiin Epic Gamesin kehittämän pelin, Gears of War:n kanssa samaan aikaan. Versiomuutos toi mukanaan tuen DirectX-grafiikkakirjaston versioille 9 ja 10, muutoksen valaistuksen hallintaan ja materiaaleihin sekä uuden fysiikkamoottorin (Angel 2016). Versiossa julkaistiin myös nykyisen käytössä olevan Blueprint -järjestelmän esiversio, kismet. Kismet-järjestelmä mahdollisti pelikoodin luomisen visuaalisillä työkaluilla, joka antoi esimerkiksi tasonkehittäjille mahdollisuuden luoda pelilogiikkaa ilman ohjelmointikielen opettelemista (Bottomley 2012). Pelimoottori oli tässä versiossa jaettu kahteen eri versioon, peruskäyttäjille tarkoitettuun UDK (Unreal Development Kit) -versioon ja yrityksien lisensointiin tarkoitettuun Unreal Engine 3 -versioon. Pelkästään pelimoottorin lisensointikustannukset ja lisensointineuvottelujen vaatimat lakimiehet pitivät Unreal Engine 3:n pienien yrityksien saavuttamattomissa (Nutt 2014).

Vuonna 2014 Epic Games julkaisi pelimoottorin seuraavan suuren versiomuutoksen, version 4. Tim Sweeney oli aikasemmassa haastattelussa kertonut kuinka pelimoottorin lisensointi oli tehty vanhentuneesti ja pelimoottorin sisältävien mahdollisuuksien takia koko lisensointijärjestelmä haluttiin tehdä uudestaan. Pelimoottorin sai uusilla lisenssiehdoilla käyttöönsä kuka tahansa 19 dollarin kuukausimaksulla ja suostumalla maksamaan rojalteina 5 % peliin liittyvästä bruttoliikevaihdosta Epic Gamesille (Nutt 2014).

Unreal Enginen version 4 elinkaareen kuuluu vielä kaksi suurta lisensointimuutosta. Vuoden 2015 maaliskuussa poistettiin aiempi kuukausimaksu, jonka jälkeen moottorin käyttö oli ilmaista kaikille, kunhan 3 000 dollarin bruttoliikevaihdon rajaa ei ylitetty (Sweeney 2015). Vuoden 2020 kesäkuussa tehtiin viimeisin lisensointimuutos, kun bruttoliikevaihdon rajaa rojalteihin liittyen nostettiin miljoonaan dollariin (Epic Games 2020).

Pelimoottorin käyttö ei enää nykyään rajoitu pelkkiin peleihin. Unreal Engineä käytetään myös esimerkiksi arkkitehtuurisissa tarkoituksissa ja virtuaalisessa tuotannossa, josta hyvä esimerkki on Disneyn luoma televisiosarja The Mandalorian, jossa pelimoottoria käytettiin joidenkin kohtausten taustojen esittämiseen (Travis 2020). Arkkitehtuurisia tarkoituksia varten Epic Games tarjoaa Twinmotion-versiota pelimoottorista, joka on suunniteltu CAD-ohjelmilla tehtyjen arkkitehtuuristen suunnitelmien esittämiseen.

Pelimoottorin versiota 5 on esitelty jonkin verran ja ensimmäisen yleisön käyttöön tulevan esiversion pitäisi ilmestyä ladattavaksi vuonna 2021. Pelimoottorista ollaan esitelty mahdollisuutta esittää kerralla useita kertoja enemmän kolmioita, kuin nykyisellä versiolla ja realistisemman näköistä dynaamisen valaistuksen esittämistä. Lisäksi uuteen versioon sisältyy virallinen julkaisu kehitteillä olevasta Chaos-fysiikkamoottorista, jonka suurin lisäys on tarkempi ja nopeampi 3D-mallien tuhoamisen fysiikkojen laskeminen (Epic Games 2020).

## 2.3 Unreal Engine ja mobiilipelit

Yleiseen käyttöön tarkoitettu versio Unreal Engine 3 pelimoottorista, UDK (Unreal Development Kit), sai tuen mobiilipelien tekemiseen joulukuussa 2010, jolloin pelimoottoriin lisättiin tuki pelien kääntämiselle iOS-käyttöjärjestelmälle OpenGL ES 2 -rajapinnalle (Epic Games 2010). Ensimmäinen pelimoottoria käyttävä mobiilipeli oli Epic Gamesin tytäryhtiön, Chair Enterntainmentin, kehittämä peli Infinity Blade, joka oli luotu esittelemään pelimoottorin kykyjä mobiililaitteilla ja se esiteltiin ensimmäisen kerran yleisölle nimellä Project Sword Applen erikoistapahtumassa vuoden 2010 syyskuussa (Caps, Jobs & Mustard 2010). Joulukuun 2010 julkaisun jälkeen peli myi hyvin ja nousi nopeasti Applen App Store -kauppapaikan nopeimmin tuottavaksi ohjelmaksi, tuottaen yli miljoona dollaria ensimmäisten päivien aikana (Edge, 2013). Yrityskäyttöön lisensoitavasta Unreal Engine 3 -versiosta löytyi tällöin jo tuki sekä iOS, että Android -käyttöjärjestelmille ja tuen oli tarkoitus myös tulla myös UDK-versioon (Graft 2011). Tuki Androidille kääntämiseen jäi kuitenkin lisäämättä UDK:n julkiseen versioon ja se nähtiin yleisessä käytössä vasta Unreal Engine 4:n ensimmäisessä versiossa. Ensimmäinen Unreal Engine 3 -pelimoottoria käyttävä molempia alustoja tukeva mobiilipeli, Dungeon Defenders: First Wave, julkaistiin iOS laitteille 15 joulukuuta 2010 ja Android-laitteille kahdeksan päivää myöhemmin, 23 joulukuuta 2010 (Ziegler 2010).

Epic Games jakaa edelleen pelimoottorimarkkinat Unity-pelimoottorin kanssa, joka oli saanut tuen iOS-laitteille julkaisemiseen jo vuonna 2008. Unityn lisenssiehdot olivat jo pitkään olleet Unreal Engineä halvemmat sallien kehittäjien hankkia kehittäjälisenssi 199 dollarin kertamaksulla, joka mahdollisti pelien julkaisun PC- ja Mac -laitteille, niin että pelin alkuruutu sisälsi Unityn brändäyksen. Pelin julkaisemiseksi ilman brändäystä vaati 1 499 dollaria maksavan Pro-lisenssin ja mobiililaitteille julkaiseminen maksoi 1 499 dollaria per alusta. Vuonna 2009 kehittäjälisenssi muutettiin ilmaiseksi, mutta brändäämättömän version julkaisulisenssin hinta pysyi samana (Unity 2009). Epic Games taas tarjosi Unreal Engine 4:lle 19 dollarin kuukausimaksullista lisenssiä, joka sisälsi mahdollisuuden julkaista kaikille tuetuille alustoille ilman lisämaksua. Lisäksi lisenssi antoi kehittäjille riskittömän tavan pelien julkaisemiseksi, koska rojalteja ei tarvinnut maksaa ensimmäisestä 3 000 dollarin liikevaihdosta (Nutt 2014).

Epic Games myös tarjoaa hankkimiensa yritysten tuotteita ilmaiseksi Unreal Engine 4:n lisenssin mukana. Näihin kuuluu esimerkiksi Quixel yrityksen yli 15 000 realistisen kuvamittauksella luodun mallin kokoelma, joka on tarkoitettu sekä pelien, arkkitehtuurisen visualisoinnin, että virtuaalisen tuotannon käyttöön. Epic Games myös tarjoaa pelimoottorin kauppapaikalta kuukausittain vaihtuvia ilmaisia tuotteita, sekä lisää silloin tällöin uutta sisältöä pysyvästi ilmaisten tuotteiden listaan. Uusin kehittäjille tarjottava tuote on vielä testivaiheessa oleva Metahumans Creator, joka mahdollistaa realistisen ihmismallien luomisen graafisella käyttöliittymällä ilman 3D-mallinnusohjelmien käyttämistä.

# 3 UNREAL ENGINE JA OPENGL ES -RAJAPINTA MOBIILIPELIN LUOMISESSA

Unreal Engine 4 sisältää pelimoottorin lisäksi useita työkaluja, jotka on suunniteltu sekä ohjelmoijien, taiteilijoiden ja animaattorien käyttöön. Huomattavin yksittäinen työkalu on pelimoottorin blueprint-järjestelmä, joka mahdollistaa pelimekaanikoiden luomisen nopeasti ja ilman koodikielen osaamista solmupohjaisella visuaalisella editorilla. Järjestelmä antaa esimerkiksi tasosuunnittelijalle mahdollisuuden testata tasoon liittyviä mekaanikoita ilman ohjelmoijan työpanosta. Oliopohjaisten objektien ja luokkien tekemiseen perustuvassa visuaalisessa editorissa ohjelmointikielen komennot on korvattu solmuilla, joita yhdistämällä luodaan pelimoottorin ymmärtämiä komentosarjoja. Blueprint-järjestelmän taustalla on C++ -ohjelmointikielellä luotu pohja ja lisää toiminnallisuutta järjestelmään voikin lisätä ohjelmointikielellä tehdyillä lisäosilla (Epic Games 2021a).

Mallien tekstuurien esittämisessä käytetään ohjelmoitavia varjostimia, joita pelimoottori kutsuu materiaaleiksi. Materiaalit ohjelmoidaan blueprint-järjestelmään perustuvalla visuaalisella editorilla. Varjostimen tai materiaalin luomisessa voidaan käyttää useita eri tekstuurikarttoja, joilla voidaan määrittää eri asioita mallin ulkoasussa. Pakolliset tekstuurikartat riippuvat mallin tyypistä mutta käytännössä läpinäkymättömän mallin esittämiseen vaaditaan vain värikartta ja läpinäkyvän mallin esittämiseen vaaditaan värikartan lisäksi myös läpinäkyvyyskartta, joka kertoo pelimoottorille mistä kohtaa malli on läpinäkyvä. Lisäksi usein käytettyjä tekstuurikarttoja ovat normaalikartta, jolla voidaan määrittää mallin pinnan muotoja kolmioiden määrää lisäämättä, karkeuskartta, joka määrittää mallin pinnan kiiltävyyden ja metallikartta, joka määrittää mallin pinnan metallisuuden.

Unreal Engine sisältää katselutilan, joka yrittää emuloida Android-laitteiden käyttämää OpenGL ES 3.2 -grafiikkarajapintaa, kääntäen kaikki käytetyt varjostimet sen määritteiden mukaisesti. Katselumoodi mahdollistaa grafiikkojen testaamisen ilman tason ajamista suoraan laitteella, mutta koska katselutila on emuloitu, ei se täysin vastaa mobiililaitteella nähtävää grafiikkaa, joten laitteellakin testaamista tarvitaan. Kehittäessä voidaan kuitenkin säästää aikaa kun pahimmillaan kymmeniä minuutteja kestävää kääntöprosessia ei tarvitse suorittaa aina, kun peliä halutaan testata laitteella.

Pelin laitteella testaamista varten käännetään se ensin laitteelle sopivaan muotoon ja käyttämään laitteen tukemaa grafiikkarjapintaa. Android-laitteille kääntämisessä käytetään Android SDK -työkalua, jolla projektista luodaan apk-tiedostopäätteinen tiedosto laitteelle asennusta varten. Kääntämisessä käytetään lisäksi Android NDK -työkalua, joka mahdollistaa C++ -ohjelmointikielellä kirjoitetun koodin ajamisen Android-laitteilla. Pelien kääntäminen Android-järjestelmälle on mahdollista sekä Windows, Linux, että Mac -laitteilla, mutta iOS-järjestelmälle kääntämiseen vaaditaan Mac-laite (Epic Games 2021b).

Virheenhallinta suoraan laitteella tapahtuu IntelliJ Ideaan perustuvaa Android Studiota käyttämällä. Peli paketoidaan ensin pelimoottorin kautta .apk-paketiksi ja sen jälkeen se asennetaan laitteelle. Projekti avataan Android Studiolla, jonka jälkeen paketin Java- ja C++ -ohjelmointikielellä kirjoitettuihin luokkiin voi lisätä pysäytyspisteitä, jotka aktivoituvat kun yhteys liitettyyn Android-laitteeseen löytyy ja projekti on käynnistetty (Epic Games 2021c).

Mobiilipeliä luodessa täytyy tietää OpenGL ES -rajapinnan rajoitteet verrattuna muihin rajapintoihin. Melkein kaikki operaatiot materiaaleissa suoritetaan pikselivarjostimessa, jotka ajetaan jokaista esitettyä pikseliä varten ja tämä on hidasta verrattuna verteksivarjostimiin, jotka ajetaan vain jokaista mallin verteksiä varten. Varjostinmatematiikkaan liittyvät laskut voidaan hoitaa Custom UV -solmun kautta, joka ajetaan aina verteksivarjostimissa, joilla säästetään aikaa pikselivarjostimien laskemiseen verrattuna. Toisin kuin pelikonsoleilla ja tietokoneilla, kaikki tekstuurikarttoja manipuloivat komennot ovat mobiililaitteilla hitaita, mutta manipulaatio voidaan suorittaa ensin Custom UV -solmun kautta verteksivarjostimessa ja näiden laskujen tulos palauttaa tekstuurikartan tesktuurikoordinatin parametriksi. Esimerkki yleisesti käytetystä tekstuurimanipuloinnista joka hyötyy Custom UV-solmusta on tekstuurien koon skaalaus (Epic Games 2021d).

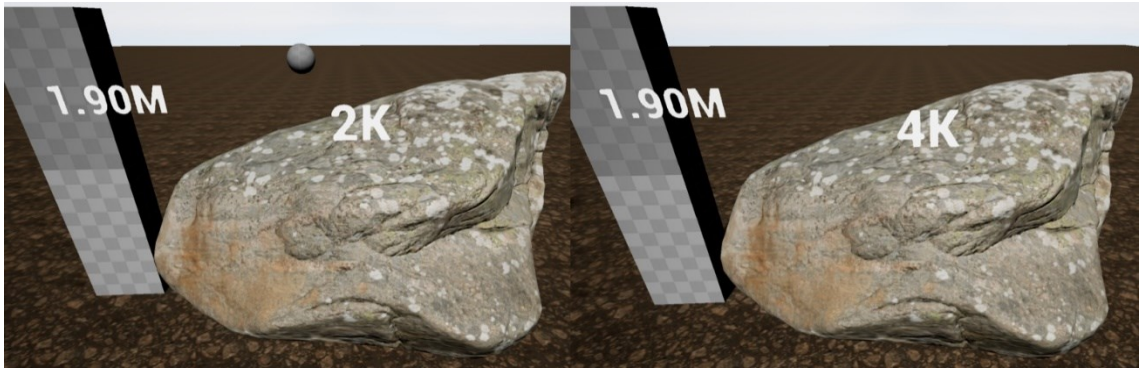
Pelinprojektin luominen voidaan aloittaa mallipohjan kanssa, mutta mobiilipeliä tehtäessä kannattaa kuitenkin aloittaa tyhjä projekti. Mallipohjat sisältävät useita valmiiksi luotuja blueprint-luokkia ja malleja kehittämistä helpottamaan, mutta ne vievät vain turhaa tilaa, jos niitä ei mobiilipelissä tarvita. Lisäksi Unreal Engine 4 sisältää useita kymmeniä lisäosia, joista voidaan poistaa käytöstä kaikki muut paitsi tarvittavat. Esimerkiksi Android-laitteille peliä luodessa voidaan huoletta ottaa käytöstä muiden laitelaustojen mediasisältöjä toistavat lisäosat.

# 4 SUORITUSKYKYTESTIN SUUNNITTELU JA TOTEUTUS UNREAL ENGINELLE

Monissa 3D-peleissä käytetään tyyliteltyä grafiikkatyyliä, jossa tarkoitus ei ole luoda mahdollisimman realistista kuvaa, jolloin suorituskykyä ei vaadita laitteelta niin paljon. Opinnäytetyössä aiotaan testata realistista grafiikkaa, johon sisältyy usein tarkoilla tekstuureilla varustettuja 3D-malleja sekä läpinäkyvyyttä käyttäviä tekstuureja esimerkiksi kasvuston esittämiseksi. Unreal Engine -lisenssiin kuuluu pääsy Quixel Megascans -kirjastoon, joka sisältää testin käyttöön hyvin sopivia kuvamittauksella luotuja malleja, joista löytyy useita versioita eri tekstuurien koolla ja mallien tarkkuudella. Malleista on mahdollista ladata myös versio, joka voidaan avata mallinnusohjelmassa, kuten avoimen lähdekoodin Blender-ohjelmassa, mallin muokkausta varten.

Testissä oli tarkoitus testata realistisen 3D-renderointia pelitarkoituksessa, joten testausta varten luotiin useita pelitasoja, joista jokainen oli oma testinsä. Testitasoja varten luotiin pohjataso, jossa maa oli luotu Landscape-objektia käyttämällä. Landscape-objektilla tason korkeuserot luodaan harmaakartan mukaan ja se on suositeltu tapa tehdä maa avoimen maailman tyyliseen tasoon. Testitasossa taso oli litteä ja kuvakulma oli ylhäältä päin, jolloin näkyviin saatiin kerralla mahdollisimman usea malli samaan aikaan. Testeissä kameran etäisyys malleihin pidettiin samana, jotta yksi malli veisi aina yhtä paljon ruudusta. Testissä käytettiin valaistuja malleja, jotta testitilanne olisi vastannut realistista peliä, jossa mallit jättävät varjoja maailmaan.

Testeissä käytettävässä HMD Nokia 6.1 -älypuhelimessa on tilaa tiedostoille ja sovelluksille järjestelmän lisäksi alle 10 gigatavua, joten tiedostokokoa ei sen puolesta tarvitse rajoittaa. Testiohjelma tulee kuitenkin asentaa aina testien ajamisen välissä, joten testien ajamisessa voidaan säästää aikaa pitämällä testiohjelman koko mahdollisuuksien mukaan mahdollisimman pienenä. Mikäli testiohjelmaan lisätään useita korkean tarkkuuden malleja, tarkoilla tekstuureilla, nousee tiedostokoko nopeasti. Koska pienemmällä ruudulla katsottaessa 2K- ja 4K -tarkkuuden tekstuurien ero on vaikeammin huomattavissa, käytettiin testissä korkeintaan 2K-tarkkuuden tekstuureja (Kuva 1).




Kuva 1. 2K- ja 4K -tarkkuuden tekstuurien vertailu.

Testattavien mallien määrä määritettiin etukäteen ja ne luotiin tasoon luomalla pelimuodon luokkaan funktio, joka luo neliön muotoisen ruudukon ja kutsuu tason luokkaa luomaan määritettyyn kohtaan halutun mallin. Jokaiselle testille luotiin oma taso, jolloin voitiin tason nimi tallentaa testiraportin tiedostonimeen ja nähdä helposti mitä raportissa on testattu. Testitasot luotiin siten, että tasossa tarvitsee vain parametreilla säätää mallien määrää, luotavia malleja, sekä niiden väliä toisistaan. Tällöin eri testitasojen luominen oli nopeasti mahdollista vain aikaisempia testitasoja kopioimalla. Malleja testatessa saman tyyppisiä malleja testattiin sekä yksittäin, että sekaisin. Mallien tyyppejä ei sekoitettu keskenään, jotta tuloksista voitiin selkeämmin nähdä mallityyppien erovaisuudet suorituskyvyssä ja myöhemmin voitiin helpommin kohdistaa optimointia niihin mallityyppeihin, jotka vaikuttivat eniten suorituskykyyn.

Käyttöliittymää varten mietittiin, mitä tietoja testiä ajettaessa tulisi näkyä. Testiä ajaessa tulisi ainakin näkyä se, että testi etenee eikä ole jäänyt jumiin. Tätä tarkoitusta varten ruudulle päätettiin lisätä reaaliaikaista tietoa sen hetkisestä ruudun piirtämiseen vaaditusta ajasta.

Ennen testien ajamisen aloitusta tuli Unreal Engine valmistella Android-laitteille kääntämistä varten. Android-projektin kääntämistä varten tulee tietokoneelta löytyä Android SDK, Android NDK ja Java JDK. Asennus suoritettiin lataamalla ja asentamalla Android Studio:n versio 3.5.3 ja pelimoottorin mukana tulevalla .bat-tiedostolla viimeisteltiin asennus. Projektin luomisen jälkeen tulee vain projektin asetuksista hyväksyä lisenssisopimukset ja määritettää projekti Android-järjestelmälle, jonka jälkeen projekti on mahdollista kääntää Android-järjestelmälle toimivaksi.

Testitasojen luominen alkoi uuden tyhjän projektin luomisella Unreal Enginessä ja asettamalla projektin laatumääritykseksi Scalable/2D. Tyhjä projekti luotiin, koska mikään

valmis mallipohja ei sisältänyt suoraan haluttuja järjestelmiä, joten ne voitiin luoda alusta alkaen itse. Scalable/2D asetuksella kerrottiin pelimoottorille, että oltiin tekemässä mobiililaitteelle sopivaa peliä tai 2D-grafiikkaan perustuvaa peliä. Kun uusi projekti oli luotu, käytiin läpi projektin lisäosa-listaus ja poistettiin käytöstä kaikki lisäosat, joiden toiminnallisuutta ei testiä luodessa tarvittu. Kaikki käyttöön otetut lisäosat pakataan mukaan pelitiedostoon vaikka niitä ei käyttäisi, joten tällä saatiin tiedostokokoa pidettyä pienempänä.

Koska tarkoituksena on käyttää Quixel Megascans -kirjaston malleja, asennettiin pelimoottoriin vielä lisäosa, jonka avulla mallit voi siirtää projektiin suoraan Quixel Bridge -ohjelmasta käyttöön valmiina. Asennus tapahtui Quixel Bridge -ohjelmasta pelimoottorin asennuskansion ja projektin kansion määrityksellä. Koska lisäosaa tarvittiin vain mallien lisäämiseen, voitiin sekin poistaa tämän jälkeen käytöstä.

Valitut mallit tuotiin pelimoottoriin Quixel Megascans -kirjastosta Quixel Bridge -ohjelmaa käyttäen, joka aikaisemmin asennetun lisäosan avulla luo malleille tekstuuritiedostot automaattisesti. Mallien esittämistä varten tarvittiin luokka, joka luo malleille ruudukon ja luo halutun määrän malleja näkyviin kerralla. Ohjelma jaettiin kahteen osaan, joissa malliruudukko luodaan haluttujen parametrien mukaan pelimuoto-luokassa ja itse mallien lisääminen ja parametrien asettaminen tehdään tason luokassa. Tämä tehtiin siksi, että pelimoottori vaatii muualla kuin tason luokassa tehtyjen Static Mesh Actor -luokkien olevan asetettu tukemaan liikkumista, kun normaalitilanteessa tällaiset tasoa täydentävät mallit olisi asetettu liikkuvuudeltaan paikallaan pysyviksi.

Mallit esitettiin neliön muotoisessa ruudukossa, joten ensin täytyi laskea tarvittava rivien määrä. Tarvittava rivien määrä saatiin ottamalla halutusta mallien määrästä neliöjuuri ja pyöristämällä luku ylöspäin, jolloin ruutuja luotiin vähintään haluttu määrä. Ylimääräisiä ruutuja tulee tällä laskukaavalla luotua, mikäli haluttu mallien määrä ei ole neliöluku, mutta laskemalla luotujen mallien määrä ja vertaamalla sitä nykyisen ruudun numeroon voitiin malleja luoda tasan haluttu määrä.

Ohjelman täytyi myös pystyä luomaan haluttuja malleja listasta vuorotellen. Tätä varten malleille luotiin array-objekti, johon halutut mallit voitiin lisätä, sekä indeksiarvo, johon lisättiin yksi tai se palautettiin nollaan riippuen viimeksi luodusta mallista. Indeksiarvoa käytettiin myös kertomaan missä indeksissä seuraavaksi luotava malli sijaitsee ja onnistuneen luonnin jälkeen arvoon joko lisättiin yksi, tai mikäli oltiin malliluettelon viimeisessä mallissa, palautetiin indeksi nollaan, jolloin seuraava malli luotaisiin taas listan alusta.

Testiraportit luotiin laitteelle käynnistämällä pelimoottoriin sisäänrakennettu tallennus konsolikomennolla ”stat StartFile” ja 10 s:n viiveen jälkeen lopettamalla tallennus komennolla ”stat StopFile”. Mallien luominen itsessään vaikutti väliaikasesti suorituskykyyn, kun malleja ja tekstuureja ladattiin, joten mallien luomisen jälkeen annettiin 5 s aikaa ennen kuin tallennuksen aloittava komento ajettiin.

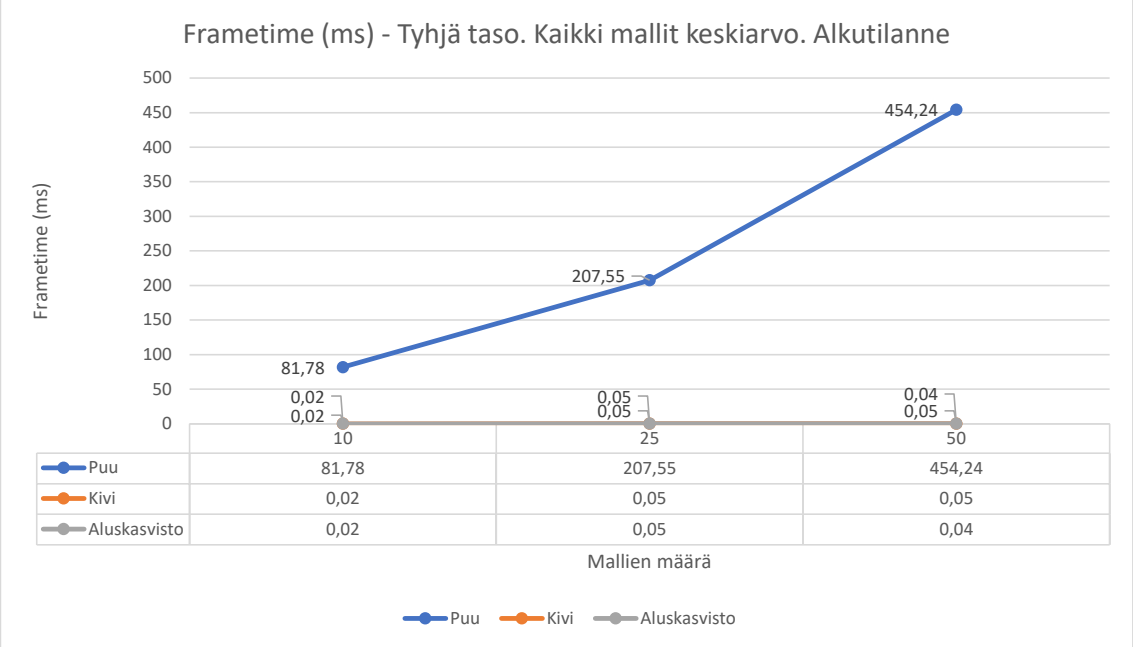
Testituloksia haluttiin verrata siten, että nähtäisiin eri mallityyppien ja määrien ero parhaiten, joten testi tehtiin myös tyhjällä tasolla, jotta tyhjän tason tuloksilla voitaisiin laskea pelkästään mallien vaikutus suorituskykyyn.

Testitasot eivät tulisi muuttumaan yhden testin aikana, joten testituloksesta voitaisiin ottaa yhden ruudun piirtämiseen vaaditun ajan keskiarvo. Tulokset kerättiin laskentataulukkoon Excelin avulla, jotta tuloksista saatiin helposti taulukoita ja kuvioita.

Testin ajamisen aikana ruudulla esitettiin reaaliaikaista tietoa ruudun piirtämiseen vaaditusta ajasta, jolloin testin mahdollinen jumiutuminen huomattaisiin helposti. Tällaista ongelmaa testeissä ei kuitenkaan lopulta havaittu.

# 5 3D-MALLIEN SUORITUSKYKYYN VAIKUTTAVAT OMINAISUUDET JA NIIDEN OPTIMOINTI

Testien ajamisen ja arvojen tallentamisen jälkeen voitiin taulukkolaskentaohjelmalla luoda arvoista taulukoita ja kuvioita. Testiraporteista koostettiin ensin kuvio, jolla voitiin vertailla yhden ruudun piirtämiseen vaaditun ajan keskiarvoa eri mallityypeillä, vähennettynä tyhjän tason piirtämiseen vaadittu aika, 17,33 ms. Kuviosta huomattiin heti, että suurin vaikutus suorituskykyyn oli testien mukaan puiden malleilla. (Kuvio 1).




Kuvio 1. Ruudun piirtämiseen käytetyn ajan keskiarvo eri mallityypeillä ja määrillä.

Testit ajettiin myös kahdella ja kolmella erinäköisellä mallilla yhtä aikaa, jotta nähtiin vaikuttiko useamman erinäköisen mallin esittäminen samaan aikaan suorituskykyyn. Testit eivät tuottaneet oletetun kaltaisia tuloksia, vaan useamman kaltaisia malleja käyttäessä suorituskyky saattoi vaihdella nähtävästi satunnaisesti, esimerkiksi puun mallien testissä yhdenlaisen mallin testi oli raskain, kahdenlaisen mallin testi oli kevyin ja kolmenlaisen testin tulos oli näiden väliltä. Malleja vertaillessa kuitenkin havaittiin, että suorituskykyero saattaisi johtua puun lehtien malleista, koska ensimmäisen tyyppisellä mallilla ne olivat lähimpänä toisiaan, toisen tyyppisellä kauimpana ja kolmannen tyyppisellä tältä väliltä.

Samanlainen tulos havaittiin ruohikon malleissa, jossa mallin koko näytti vaikuttavan suorituskykyyn. Kivien malleissa ei huomattavia eroja suorituskyvyssä havaittu käyttämällä useamman näköisiä malleja.

Mikäli halutaan että piirretty kuva vaikuttaisi sulavalta, tulee ruudunpäivitysnopeuden olla vähintään 24 ruutua sekunnissa. Vähimmäisnopeus yhden ruudun piirtämiseen tulisi siis olla 41,7 ms, koska yhteen sekuntiin eli tuhanteen millisekuntiin mahtuu 24 kappaletta 41,7 ms:ssa piirrettyä ruutua. Tyhjän tason käyttäessä noin 17,33 ms yhden ruudun piirtämiseen, kivien ja kasvien malleilla saatiin ruudunpäivitysnopeudet pidettyä lähes 60:ssä vielä 50 mallilla, joten suorituskyvyn optimointi aloitettiin yhteisen optimointiyrityksen jälkeen puiden mallista.

Yhteisenä optimointina kaikille mallityypeille koetettiin muuttaa mallien luomiskoodia siten, että yksittäisten mallien lisäämisen sijaan luotiin vain instansseja samasta mallista. Tällä saatiin vähennettyä tasojen käyttämiä piirtokutsuja huomattavasti, koska yhden instanssin lisääminen ei lisää enää ylimääräisiä piirtokutsuja. Tämä muutos saatiin aikaiseksi muutamalla mallien lisäyskoodia tason blueprint-luokasta. Luokassa luodaan yksi instansoitu versio halutusta mallista ja tästä versiosta luodaan instanssi jokaiseen haluttuun kohtaan. Testi ajettiin ensin käyttämällä vain yhden näköistä mallia jokaisesta mallityypistä ja koska suorituskyvyssä ei huomattu vaikutuksia, testiä ei ajettu enää uudestaan useamman näköisellä mallilla. Piirtokutsujen määrää onnistuttiin alentamaan huomattavasti, mutta sillä ei testien mukaan ollut vaikutusta suorituskykyyn.

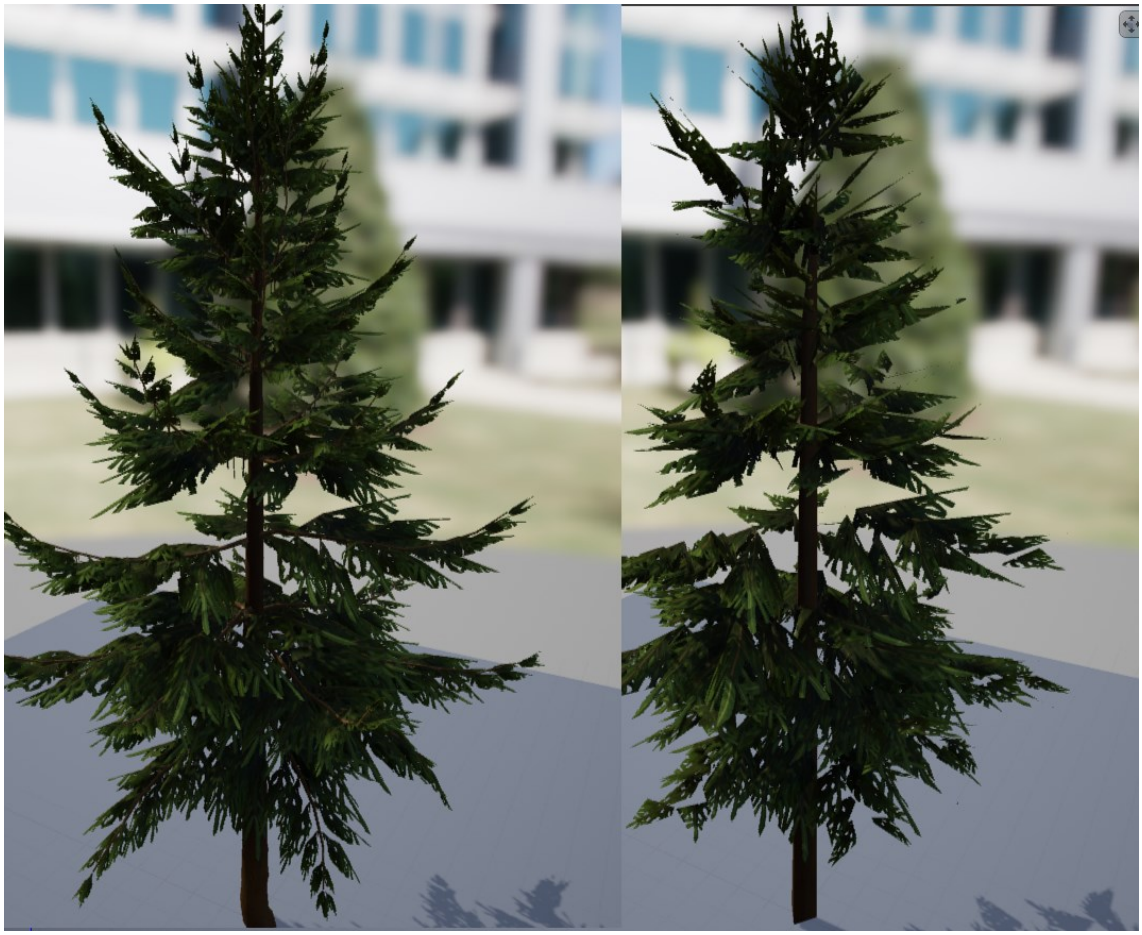
## 5.1 Puun mallin optimointi

Mallityyppejä verrattaessa huomattiin, että puiden mallit olivat yhdistelmä kahdesta muusta mallityypistä, sisältäen tarkemman mallin rungon ja oksien esittämiseen, sekä läpinäkyviä tekstuureja lehtien esittämiseen. Käytettäessä editoriin sisällytettyä katselumoodia, joka näyttää varjostimen monimutkaisuuden, huomattiin että läpinäkyvien tekstuurien monimutkaisuus on paljon suurempi läpinäkymättömään osaan puusta (Kuva 2).


Kuva 2. Puun mallin varjostimien monimutkaisuus.

Ruohikon mallien testiin verrattuna, kerralla ruudulla näkyvien läpinäkyvien tekstuurien osuus on puiden malleissa paljon suurempi ja kerralla toistensa läpi nähtyjä tekstuureja on enemmän, jolla oli testien perusteella suuri vaikutus suorituskykyyn.

Ensimmäisenä testattiin mallien yksinkertaistamista vähentämällä mallin kolmioiden määrää luomalla mallista pelimoottorin kanssa LOD-tasoja. Ensimmäinen taso oli täysi tarkkuus, jolloin ensimmäisen tyypin puussa on 9 790 kolmiota. Seuraava LOD-taso vähensi kolmioiden määrän 7 567:ään, mutta vaikka kolmioiden määrää väheni noin 22,7 %, vaikutus suorituskykyyn oli pieni, yhden ruudun piirtämiseen vaadittavan ajan laskien 10:n puun testissä yhteensä vain noin 10 ms. Seuraava taso oli noin 38 % vähennys täydestä tarkkuudesta ja saavutettu säästö yhden ruudun piirtämiseen vaadittusta ajasta oli 10 mallin testissä edellisestä tasosta vain noin 5 ms. Pelkästään LOD-tasojen määrää lisäämällä oli mahdollista viedä mallin tarkkuus todella alas, mutta tämä vaikutti mallin ulkonäköön, jolloin siirtymän mallien välillä huomasi helposti lehdistä (Kuva 3).



Kuva 3. Puun malli lehtien vääristymä LOD-tasoilla. Alkuperäinen vasemmalla.

Mallien esittämiseen matalalla tarkkuudella on kuitenkin toinenkin tapa. Lisäämällä malliin alimmaksi tasoksi pelkistetyn version mallista, jossa yksittäisien oksien ja lehtien malli on korvattu yksinkertaisella ristileikkaavalla mallilla, jota peittää vain yksi läpinäkyvä tekstuuri, joka on luotu ottamalla kuva puusta kahdeksasta eri kulmasta ja ylhäältä päin. Näkyvä voidaan halutessa määrittää pelaajan katselukulman mukaan ja tekstuurin kääntämiseen voidaan käyttää lisäksi ristihäivytystä, jolloin efektin huomaaminen on vaikeampaa. Läheltä tarkastellessa eron normaaliin malliin huomasi helposti, mutta varsin lyhytkin etäisyys kamerasta pienellä ruudulla mahdollisti useamman puun käyttämisen ilman suorituskyvyn suurta laskua. Tätä billboard-malliksi kutsuttua mallia käyttäessä laski puiden vaatima suorituskyky huomattavasti. Testatessa 10 mallilla ruudun piirtämiseen vaadittu aika oli noin 1 ms:n ja 50 mallin esittäminenkin onnistui noin 26,58 ms:ssa (Taulukko 1). Tämän tyyppinen malli soveltui hyvin kaukana kamerasta olevien puiden esittämiseen, varsinkin kun suorituskykyvaikutus väheni sen mukaan mitä vähemmän mallin läpinäkyvistä tekstuureista piirrettiin kerralla.

Taulukko 1. Puun malli billboard-tyyppisenä.

| Mallien määrä | Frametime (ms) – tyhjä taso. Alkuperäinen | Frametime (ms) – tyhjä taso. 36 kolmiota |
|---|---|---|
| 10 | 89,72 | 1,29 |
| 25 | 211,93 | 9,86 |
| 50 | 507,97 | 26,58 |

Seuraavaksi kokeiltiin tekstuurien tarkkuuden vähentämistä. Tarkkuuden vähentäminen vaikutti ulkoasuun varsinkin läheltä katsottuna, mutta pientä ruutua käyttäessä vaikutus oli vaikeampi havaita, kuin isolla monitorilla. Mallin tekstuurien kokoa testattiin alkutilanteessa käytettyjen 2K-tarkkuuden tekstuurien lisäksi 1K- ja 512 -tarkkuuden tekstuureilla. Testistä saatuja tuloksia tarkastellessa huomattiin, että tekstuurien tarkkuuden vaihtamisella ei saavuttettu merkittävää vaikutusta suorituskykyyn. (Taulukko 2).

Taulukko 2. Puun malli eri tarkkuuden tekstuureilla.

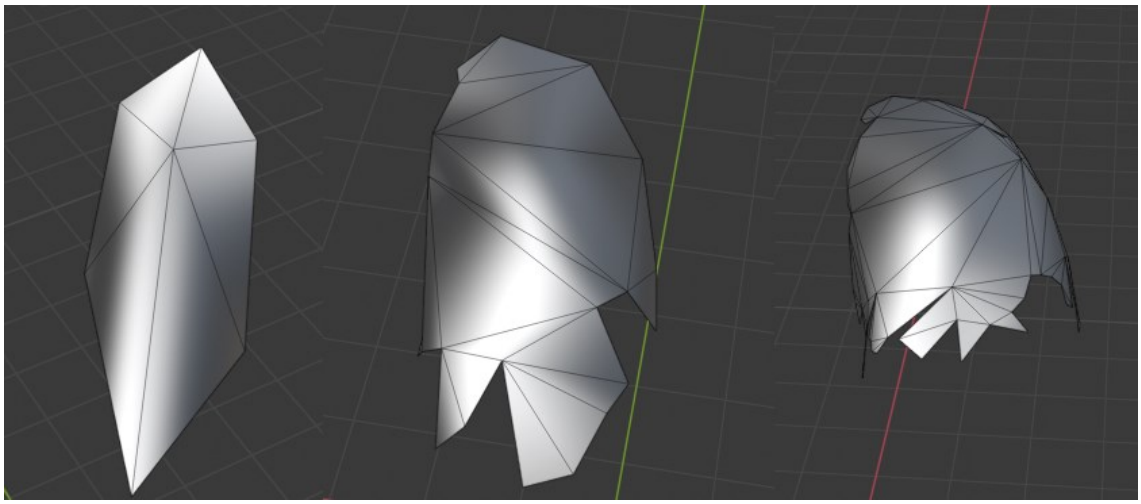
| Mallien määrä | Frametime (ms) – tyhjä taso 2K | Frametime (ms) – tyhjä taso 1K | Frametime (ms) – tyhjä taso 512 |
|---|---|---|---|
| 10 | 89,72 | 85,89 | 86,2 |
| 25 | 211,93 | 212,3 | 216,16 |
| 50 | 507,97 | 424,08 | 449,71 |

Puissa on alun perinkin käytetty vain kolmea tekstuurityyppiä, värejä esittävää tekstuurikarttaa, mustavalkoista läpinäkyvyyskarttaa ja pinnan muotoja esittävää normaalikarttaa. Vaikka normaalitilanteessa läpinäkyvää tekstuuria esittäessä vaaditaan kahden erillisen tekstuurikartan käyttämistä, voidaan tämä rajoitus kiertää muokkaamalla värejä esittävää tekstuurikarttaa ja rajaamalla sen läpinäkyvyys läpinäkyvyyskartan mukaisesti. Tekstuurikartan eri kanavien käyttäminen erikseen on mahdollista, jolloin läpinäkyvyys voidaan ilmaista tekstuurikartan läpinäkyvyyskanavan mukaan. Normaalikartan luomat yksityiskohdat olivat pienehköjä ja kartta ei ole välttämätön puun esittämiseen, joten se poistettiin käytöstä. Yhden tekstuurikartan käyttämisellä havaittiin selkeä ero suorituskyvyssä ja se todettiin toimivaksi optimointitavaksi. (Taulukko 3).

Taulukko 3. Puun malli vain yhdellä tekstuurikartalla

| Mallien määrä | Frametime (ms) – tyhjä taso. Alkuperäinen | Frametime (ms) – tyhjä taso. 1 tekstuurikartta |
|---|---|---|
| 10 | 89,72 | 34,92 |
| 25 | 211,93 | 93,9 |
| 50 | 507,97 | 194,45 |

Puiden mallien käyttämä kolmioiden määrä ei vaikuttanut testien perusteella suorituskykyyn yhtä paljon, kuin tekstuurien varjostimien monimutkaisuus. Tätä voitiin optimoinnissa yrittää käyttää hyödyksi rajaamalla lehtien tekstuuria esittävät mallit lähemmäs itse lehtien rajoja, jolloin tekstuurista täytyisi läpinäkyvää osaa esittää mahdollisimman vähän. Yrittämällä poistaa mahdollisimman paljon läpinäkyvää tekstuuria, kuitenkaan liian paljon kolmioita lisäämättä, päädyttiin ensimmäistä testiä varten nostamaan kolmioiden määrä yhden lehden mallissa 6:sta 18:aan. Tämä muutos nosti puun mallin kolmioiden määrän yhteensä 14 814 kolmioon. Lisäksi tehtiin vielä rajatumpi malli, jossa välitettiin vähemmän kolmioiden määrästä, kuin mallin tarkkuudesta. Koko puun mallin kolmioiden määräksi tulitätä mallia käyttämällä 51 866 kolmiota. (Kuva 4).



Kuva 4. Alkuperäinen, tarkempi ja tarkin lehtien malli.

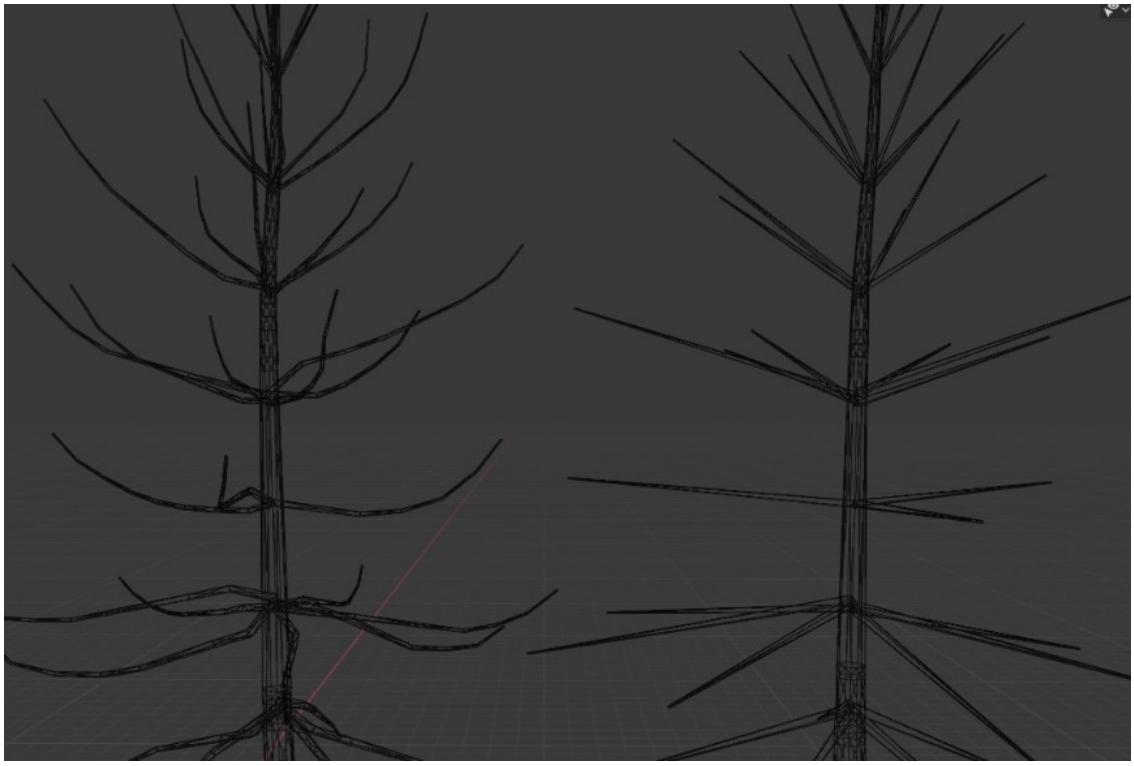
Testeissä huomattiin, että tarkimmalla mallilla suorituskyky laski huomattavasti, kun ruudun piirtämiseen vaadittu aika nousi noin puolitoistakertaiseksi. Puiden määrän lisääminen nosti ruudun piirtämiseen vaadittua aikaa samassa suhteessa lisättyjen puiden määrän kanssa. Vähemmänkin tarkalla mallilla ei testeissä havaittu kuin negatiivisia

vaikutuksia suorituskykyyn. Lehtien mallien tarkentamisella ei saatu siis halutunkaltaisia tuloksia suorituskyvyssä, joten optimointiyritystä ei jatkettu enempää. (Taulukko 4).

Taulukko 4. Puun malli tarkemmilla lehtien malleilla.

| Mallien määrä | Frametime (ms) – tyhjä taso. Alkuperäinen. 9790 kolmiota. | Frametime (ms) – tyhjä taso. 14814 kolmiota. | Frametime (ms) – tyhjä taso. 51866 kolmiota. |
|---|---|---|---|
| 10 | 107,05 | 120,45 | 159,20 |
| 25 | 229,26 | 308,5 | 412,61 |
| 50 | 525,3 | 645,49 | 805,28 |

Puiden malleja yritettiin vielä optimoida yksinkertaistamalla mallia mahdollisimman paljon. Itse rungosta ja oksista voitiin vähentää kolmioiden määrää, jolloin mallin ulkonäkö muuttui yksinkertaisemmaksi, mutta mobiililaitteilla vaikutus havaittiin pienempi kuin isolla ruudulla. Yksinkertaistamalla lisäksi lehtien mallia vain neliskulmion muotoiseksi, saatiin se käyttämään vain 2 kolmiota. Tällä saatiin koko puu käyttämään vain 1 884 kolmiota, joka oli 80.76 % vähemmän kolmioita kuin alkuperäisessä testatussa mallissa. Suurin osa itse mallin optimoinnista keskittyi oksiin ja niiden laadusta karsittiin huomattavasti (Kuva 5).



Kuva 5. Vasemmalla alkuperäinen puun malli, oikealla rankasti yksinkertaistettu.

Tällä optimoinnilla huomattiin puiden mallissa selkeä nousu suorituskyvyssä 10:n mallin testissä, suorituskyvyn noustessa lähes 40 % ja 50:n mallin testissä lähes 50 % (Taulukko 5). Kolmioiden agressiivinen vähentäminen mahdollisti yksittäisten puiden esittämisen pelaajan näkökentässä ja yhdistettynä billboard-tyyppiseen malliin, voitaisiin puita esittää taustallakin.

Taulukko 5. Alkuperäinen puun malli verrattuna 1 884 kolmion malliin.

| Mallien määrä | Frametime (ms) – tyhjä taso. 9 790 kolmiota | Frametime (ms) – tyhjä taso. 1 884 kolmiota |
|---|---|---|
| 10 | 0,04 | 10,91 |
| 25 | 0,07 | 41,41 |
| 50 | 0,05 | 101,68 |

Puiden optimointiyrityksistä huomattiin, että kerralla näkyvien täyslaatuisten puiden määrää tulee tasonsuunnittelussa rajoittaa ja mikäli puita esittää, kannattaa kiinnittää huomiota kerralla toisistaan läpi näkyvien tekstuurien määrään ja varjostimien

monimutkaisuuteen. Yhdistämällä LOD-tarkkuuksia vähemmän tarkkojen mallien kanssa voitaisin lähellä olevia puita kuitenkin pelaajalle esittää, varsinkin jos mallien sijainti suunniteltaisiin etukäteen, minimoiden kerralla näkyvien mallien määrä ja niiden lehtien mallien läpinäkyvyys toistensa kanssa. Käyttämällä billboard-tyyppistä mallia voitaisiin pelaajalle hyvin esittää useita kaukana näkyviä puita ilman suurempaa vaikutusta suorituskykyyn. Testien perusteella tiheiden, realistisien metsien esittäminen mobiililaitteilla olisi kuitenkin vaikeaa.

## 5.2 Ruohon mallin optimointi

Alkuperäistä testiä ajettaessa ruohikon mallien esittämisellä ei havaittu olevan kovin suuria suorituskykyvaikutuksia. Puiden mallien optimoinnista kuitenkin huomattiin, että läpinäkyvät tekstuurit vaikuttivat suorituskykyyn sitä enemmän, mitä niitä näkyi toistensa läpi. Tästä syystä aluskasviston testi ajettiin vielä uudestaan kääntämällä kamera vaakatasoon, jolloin mallit näkyvät mahdollisimman paljon toistensa läpi. Näin testaamalla huomattiin, että aluskavuston mallien suorituskykyvaatimus nousi erittäin paljon alkuperäiseen testiin verrattuna. Alkuperäisessä testissä 100:kaan mallilla ei ollut vielä kovin merkittävää vaikutusta suorituskykyyn, kun vaakatasossa testaamalla vaikutus on jo 228,83 ms:n vaadittu aika yhden ruudun piirtämiseen. Kun testi ajettiin 500 mallilla oli ruudun piirtämiseen vaadittu aika melkein 1 000 % korkeampi kuin alkuperäisessä testissä. Malleja tulisi siis optimoida paljon aikaisempaa arviota enemmän. (Taulukko 6).

Taulukko 6. Ruohon mallin testi pysty- ja vaakatasossa olevan kameran kanssa.

| Mallien määrä | Frametime (ms) – tyhjä taso. Pystytaso | Frametime (ms) – tyhjä taso. Vaakataso |
|---|---|---|
| 10 | 0,04 | 10,91 |
| 25 | 0,07 | 41,41 |
| 50 | 0,05 | 101,68 |
| 100 | 2,24 | 228,83 |
| 250 | 11,66 | 287,4 |
| 500 | 30,17 | 291,72 |

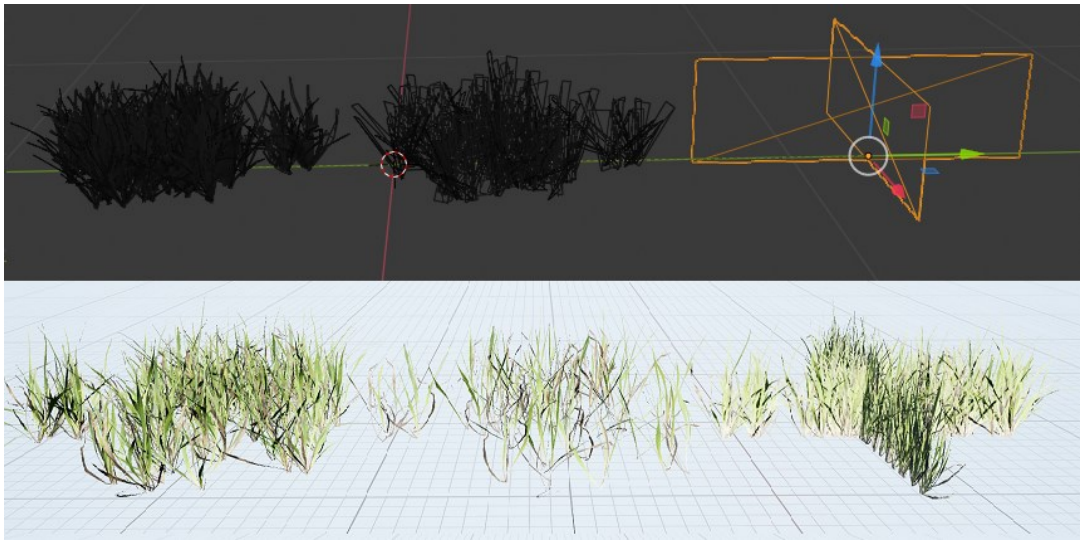
Malleissa käytettiin väri- ja läpinäkyvyyskarttojen lisäksi myös normaali- ja karkeuskarttoja. Pienemmän ruudun takia karkeuskartta voitiin korvata numeroarvolla ja

normaalikartan yksityiskohtia ei tarvittu, koska ne eivät kuitenkaan näkyneet juurikaan muutenkin pienikokoisena ruudulla näkyvässä mallissa. Vähentämällä käytettyjen tekstuurien määrää saavutettiin jo 10:llä mallilla noin 2 ms:n säästö ruudun piirtämiseen vaaditusta ajasta ja 500:lla mallilla jo noin 25 ms:n säästö. Itsessään optimointi ei riittänyt mallien esittämiseen, mutta sitä voitaisiin tarvittaessa yhdistää muihin optimointeihin (Taulukko 7).

Taulukko 7. Ruohon malli piirrettynä vain kahta tekstuurikarttaa käyttäen.

| Mallien määrä | Frametime (ms) - Vaakataso | Frametime (ms) – tyhjä taso. Vaakataso |
|---|---|---|
| 10 | 26,25 | 8,92 |
| 25 | 53,91 | 36,58 |
| 50 | 105,54 | 88,21 |
| 100 | 219,8 | 202,47 |
| 250 | 285,53 | 268,2 |
| 500 | 282,82 | 265,5 |

Mallit tuotiin Megascans-kirjastosta Quixel-lisäosan avulla jossa mallit oli tehty valmiiksi jo rajaten niitä mahdollisimman paljon, joten läpinäkyvyyttä on jo mahdollisimman vähän päällekkäin, joten samanlaista optimointia kuin puiden lehtien mallien kanssa ei kannattanut yrittää. Ruohikon mallia ei myöskään suoraan voitu samalla tavalla yksinkertaistaa, kuin lehtien malleja, koska malli on tehty yksittäisiä ruohonkorsia yhdistämällä. Ruohikon mallit koostuivat suhteellisen suuresta määrästä kolmioita mallien kokoon verrattuna, joten kolmioiden määrää vähennettiin. Kolmioiden määrän vähentäminen tapahtui ruohikon mallin tiheyttä vähentämällä, jolloin yksi malli näytti kerralla vähemmän ruohonkorsia. Ruohon mallista tehtiin myös puun mallin optimoinnissa käytetty billboard-tyyppinen malli, joka käytti mahdollisimman vähän kolmioita ja läpinäkyviä tekstuureja, mutta vaikutus ulkonäköön oli niin suuri että niitä ei voitaisi esittää pelaajan lähellä (Kuva 6).



Kuva 6. Ruohikon malli yksinkertaistettu billboardiksi asti. Alkuperäinen vasemmalla.

Suorituskyvyssä huomattiin selkeä ero mallien välillä. Yksinkertaisempia malleja voitiin esittää kerralla 10 ilman että suorituskyvyssä huomattiin juurikaan eroa ja bilboard malleissa suorituskyvyssä huomattiin vaikutus vasta 50:llä mallilla (Taulukko 8).

Taulukko 8. Ruohon malli eri kolmioiden määrällä käyttäen horisontaalista kameraa.

| Mallien määrä | Frametime (ms) – tyhjä taso. 3 500 kolmiota | Frametime (ms) – tyhjä taso. 438 kolmiota | Frametime (ms) – tyhjä taso. 4 kolmiota |
|---|---|---|---|
| 10 | 10,91 | 0,39 | 0,07 |
| 25 | 41,41 | 15,13 | 0,06 |
| 50 | 101,68 | 43,01 | 0,09 |
| 100 | 228,83 | 100,8 | 7,43 |
| 250 | 287,4 | 146,2 | 9,69 |
| 500 | 291,72 | 169,11 | 10,39 |

Ruohon malleja optimoidessa havaittiin samanlainen ongelma kuin puiden malleissa, jossa pelaajan lähellä oli vaikeaa esittää useita malleja samaan aikaan. Ongelma oli kuitenkin mallien koon takia pienempi, jolloin optimoituja malleja voitaisiin esittää noin 10 ja billboard-tyyppisiä malleita useita kymmeniä. Mallityypin käyttö oli siis helpompaa kuin puun mallilla, mutta samanlaisesta mallien sijainnin suunnittelusta olisi myös ruohon malleja käytettäessä hyötyä.

## 5.3 Kiven mallin optimointi

Kivien mallit tuotiin Megascans-kirjastosta Bridge-lisäosan kautta. Lisäosa loi malleille automaattisesti materiaalit, käyttäen tekstuurikarttoina väri-, karkeus- ja normaalikarttoja. Materiaalissa lisäksi annettiin käyttäjälle mahdollisuus säätää useita parametreja mallien yksilöimiseksi ja osa näistä, kuten tekstuurien skaalaus, vie enemmän suorituskykyä mobiililaitteilla kun tietokoneilla. Mallien esittämiseen ei käytännössä tarvita muita tekstuurikarttoja kuin värikartta, mutta normaalikartasta ja karkeuskartasta olisi mallin koosta riippuen enemmän tai vähemmän vaikutusta ulkonäköön. Testausta varten luotiin mallista versio joka käytti kolmea tekstuurikarttaa, väri-, karkeus- ja normaalikarttoja, ilman mahdollisuutta muokata materiaalia parametreilla. Tätä versiota testattaessa huomattiin pieni vaikutus suorituskykyyn yli sataa mallia käyttäessä ja koska automaattisesti luotuja parametreja ei testissä tarvittu, poistettiin ne käytöstä. (Taulukko 9).

Taulukko 9. Kiven mallien testi käyttäen pelkkiä tekstuurikarttoja.

| Mallien määrä | Frametime (ms) – tyhjä taso. Alkuperäinen | Frametime (ms) – tyhjä taso. Pelkät tekstuurikartat |
|---|---|---|
| 10 | 0,01 | 0,06 |
| 25 | 0,04 | 0,06 |
| 50 | 0,04 | 0,06 |
| 100 | 3,43 | 2,95 |
| 250 | 14,82 | 13,5 |
| 500 | 35,67 | 31,62 |

Kivien malleista vähennettiin myös kolmioiden määrää. Toisin kuin aiemmissa optimoinneissa, kivien yksinkertaisempi muoto helpotti kolmioiden määrän vähentämistä ilman yhtä suurta vaikutusta ulkoasuun, kuin aikaisempien mallityyppien kanssa. Vähentämällä kolmioiden määrää alkuperäisestä 5 134 kolmiosta 600 kolmioon, voitiin kerralla esittää 250 mallia ilman huomattavaa vaikutusta suorituskykyyn ja vielä 500 mallin testissäkin suorituskykyvaikutus oli vain noin 5 ms:n lisäruudun piirtämiseen vaadittuun aikaan. Kolmoiden määrä valittiin tarkastelemalla Blender-mallinnusohjelman decimate-muuntimen vaikutusta ulkonäköön. Decimate-muunnin vähentää mallin kolmioiden määrää, yrittäen samalla kuitenkin vaikuttaa mahdollisimman vähän mallin muotoon.

Muunninta käytettiin niin monta kertaa kunnes ulkonäössä havaittiin suurempi ero ja palattiin tätä edeltävään asetukseen. Mallia yksinkertaistettiin 500:n mallin testiä varten vielä ulkonäöstä välittämättä edelleen 150 kolmioon, jolla saatiin vielä noin kahden millisekunnin säästö yhden ruudun piirtämiseen vaaditusta ajasta (Taulukko 10)

Taulukko 10. Kiven mallin suorituskyky eri määrällä kolmioita.

| Mallien määrä | Frametime (ms) – Vaakataso. 5 134 kolmiota | Frametime (ms) – tyhjä taso. 600 kolmiota | Frametime (ms) – tyhjä taso. 150 kolmiota |
|---|---|---|---|
| 10 | 0,01 | 0,07 | 0,06 |
| 25 | 0,04 | 0,06 | 0,04 |
| 50 | 0,04 | 0,08 | 0,04 |
| 100 | 3,43 | 0,08 | 0,02 |
| 250 | 14,82 | 0,03 | 0,09 |
| 500 | 35,67 | 5,24 | 3,02 |

Kivien malleja voitaisiin näyttää pelaajalle useita satoja kerrallaan, varsinkin jos niiden esittämiseen käytettäisiin LOD-tasoja, jotka olisi määritetty käyttämään eri tarkkuuden versiota mallista riippuen kuinka suurena se näkyy pelaajan ruudulla. Lähellä voitiin esittää 50 mallia täydellä tarkkuudella ja käyttäen aiemmin luotuja yksinkertaistettuja malleja kauempana, voitaisiin kerrallaan pelaajalle esittää yli 250 kiven mallia kerrallaan.

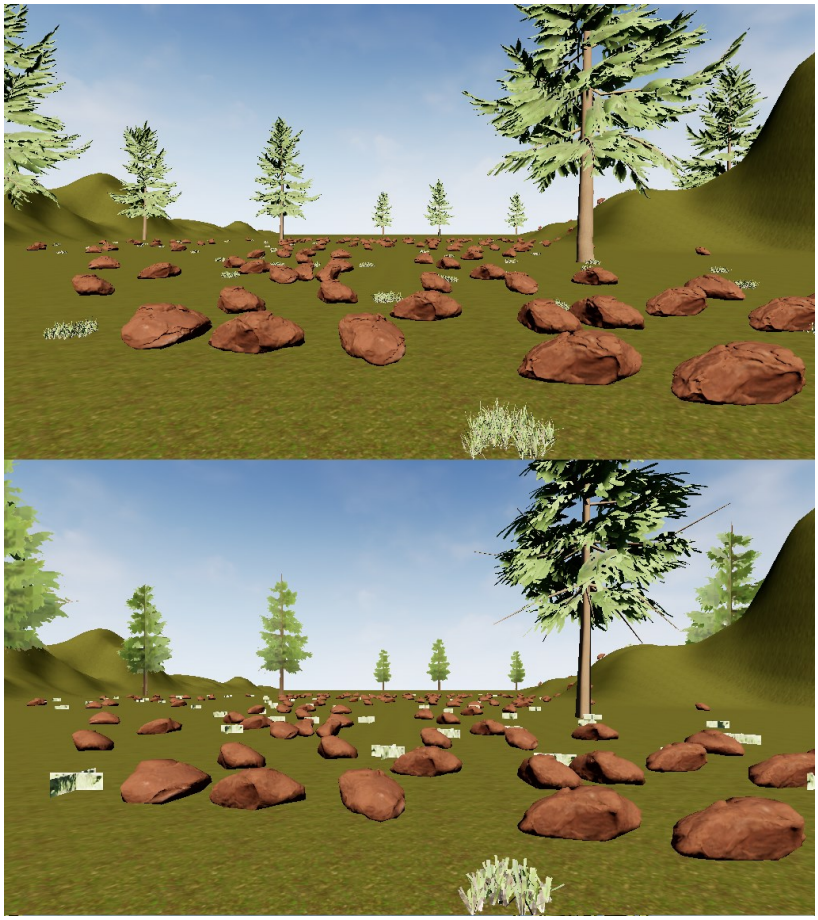
## 5.4 Yhteinen testitaso

Lopulta tehtyjen optimointien perusteella luotiin yhteinen testitaso, jossa käytettiin kaikkia kolmea mallityyppiä sekaisin. Tasossa yritettiin luoda pieni maisema, jonka pelaaja voisi mahdollisesti nähdä realistista 3D-peliä pelattaessa. Yhteisessä tasossa testatttiin suoraan ruudunpäivitysnopeutta, eli montako kertaa sekunnissa piirretään uusi kuva näytölle.

Yhteisessä testitasossa käytettiin sekaisin optimointitapoja, joista oli havaittu olevan vaikutusta suorituskykyyn. Puiden malleista käytettiin 1 884 kolmiota käyttävää versiota yhdellä tekstuurikartalla ja kauempana se muutettiin käyttämään billboard-tyyppistä mallia. Ruohon mallissa käytettiin lähellä 438 kolmion versiota ja pienen etäisyyden päässä

billboard-mallia. Kivien malleissa käytettiin lähellä 600 kolmion versiota yhdellä tekstuurikartalla ja kauempana kamerasta 150 kolmion versiota.

Eri tyyppisten mallien määrät päätettiin alun perin vertailemalla optimointien testiraportteja ja arvioimalla niiden perusteella montako mallia voisi kerralla mahdollisesti näyttää. Alkutilanteessa päädyttiin käyttämään 10 puun mallia, 50 ruohon mallia ja 200 kiven mallia. Testaamalla tällä määrällä huomattin, että ruudunpäivitysnopeus oli noin 30 ruutua sekunnissa, joka oli tavotteiden mukainen. Vaikka 30:n ruudunpäivitysnopeudella peli vaikuttaakin sulavalta, koetettiin sitä nostaa vielä vähän mallien määrää säätämällä. Testaamalla tasoa laitteessa mallien määrää vähitellen muuttamalla, päädyttiin lopulta käyttämään 8 puun mallia, 65 ruohon mallia ja 169 kiven mallia, jolla ruudunpäivitysnopeus oli hyvin pelaamiseen sopiva noin 40 ruutua sekunnissa. Sama pelitaso testattiin myös optimoimattomia malleja käyttäen, jotta nähtiin kuinka paljon optimoinneilla oltiin yhteensä vaikutettu suorituskykyyn (Kuva 7).



Kuva 7. Lopullinen optimoimaton ja optimoitu testitaso. Optimoimaton ylhäällä.

Testin perusteella havaittiin selkeä vaikutus suorituskykyyn optimoiduilla malleilla. Optimoidun tason pystyessä piirtämään noin 40 ruutua sekunnissa, pystyi optimoimaton taso piirtämään vain noin 15, joka ei riitä sulavan kuvan esittämiseen (Taulukko 11).

Taulukko 11. Yhteisen testitason ruudunpäivitysnopeudet

| Ajettu testitaso (Puu/ruoho/kivi) | Ruudunpäivitysnopeus |
|---|---|
| Optimoitu 10/50/200 | 29,8 – 30,2 |
| Optimoitu 8/65/169 | 39,6 – 40,3 |
| Optimoimaton 8/65/169 | 14,5 – 15,2 |

# 6 JOHTOPÄÄTÖKSET

Opinnäytetyön tavoitteena oli perehtyä 3D-grafiikkaan peleissä, sekä suunnitella ja toteuttaa Unreal Engine 4 -pelimoottorilla suorituskykytesti, jolla voitiin testata erilaisten 3D-mallien vaikutusta suorituskykyyn Android-mobiililaitteilla. Tarkoituksena oli lisäksi optimoida testattuja malleja, jotta niitä käyttämällä voitaisiin saavuttaa pelikelpoinen ruudunpäivitysnopeus mobiilipelissä.

Useita läpinäkyviä tekstuureja käyttävät mallit osoittautuivat testeissä eniten suorituskykyä vaativiksi. Useamman puun mallin esittäminen kerralla vähensi suorituskykyä huomattavasti ja mallit vaativat käyttäessä useita optimointeja. Sekä puiden, että ruohikon mallit käyttävät useita läpinäkyviä tekstuureja ja niiden käyttämä suorituskyky riippuukin paljon siitä, mistä kulmasta niitä katsotaan ja näkyykö niiden läpi muita läpinäkyviä tekstuureja. Tehokkain optimisaatio näissä malleissa oli billboard-tyyppisen mallin luominen, jolla on kuitenkin erittäin suuri vaikutus mallin ulkonäköön. Tätä vaikutusta pystyi vähentämään ristihäivytystä käyttämällä, mutta läheltä katsottuna sen huomasi aina helposti. Näitä malleja voitiin kuitenkin käyttää kaukana pelaajasta olevien puiden ja ruohikoiden esittämiseen, varsinkin pienemmällä ruudulla koska muutos ulkonäköön oli vaikeammin huomattavissa. Puiden ja ruohon malleissa hyvin toiminut optimisaatio oli myös tekstuurikarttojen yhdistäminen ja vähentäminen, jonka kanssa pelaajan lähellä näkyvien mallien määrää voitiin kasvattaa.

Kivien malleja voitiin pelaajan lähellä esittää optimoimattakin useita kymmeniä, mutta luomalla mallista yksinkertaistettuja versioita ja vaihtamalla näitä näkyviin riippuen siitä kuinka suurena malli pelaajan ruudulla näkyi, saatiin kerralla esitettyä useita satoja malleja. Malleista vähennettiin vielä käytetyt tekstuurikartat pelkkään värikarttaan, jolla säästettiin vielä muutamia prosenttyksiköitä suorituskyvyssä riippuen siitä, paljonko malleja näytettiin kerralla.

Yhteisen testitason perusteella havaittiin, että optimoinneilla saatiin luotua pelitaso, joka toimi hyvin pelikelpoisella ruudunpäivitysnopeudella. Yhteisessä pelitasossa käytettyjä optimointitapoja voikin tarvittaessa käyttää oman pelin optimointiin.

# LÄHTEET

Acharya, Abinash 2019: How did the PUBG game become famous in India?. Quora. Saatavissa: https://www.quora.com/How-did-the-PUBG-game-become-famous-in-India/answer/Abinash-Acharya-24. Viitattu 20.4.2021

Angel, Evan 4.11.2016: This is Unreal Engine. Hotgates. Saatavissa: https://hotgates.eu/this-is-unreal-engine/. Viitattu 20.4.2021

Angelos, Ayla 23.2.2021: The history of Snake: How the Nokia game defined a new era for the mobile industry. It's Nice Chat. Saatavilla: https://www.itsnicethat.com/features/taneli-armanto-the-history-of-snake-design-legacies-230221. Viitattu 23.4.2021

Blythe, David 22.1.2020: OpenGL ES Comon/Common-Lite Profile Specification. Khronos. Saatavissa: https://www.khronos.org/registry/OpenGL/specs/es/1.0/opengles_spec_1_0.pdf. Viitattu 27.3.2021

Bottomley, Pete 3.3.2012: UDK: Introduction to Kismet. World of Level Design. Saatavilla: https://www.worldofleveldesign.com/categories/wold-members-tutorials/petebottomley/udk-kismet-introduction.php. viitattu 23.4.2021

Cassidy 27.5.2019: An Introduction to "Yaroze a Day". Bad Game Hall of Fame. Saatavilla: https://www.badgamehalloffame.com/yaroze-a-day-00/. Viitattu: 17.5.2021

Chen, Brian 11.4.2009: Apple's App Store Hits Six Digits; How Many Apps Do You Need?. Wired. Saatavilla: https://www.wired.com/2009/11/appstore/. Viitattu 20.4.2021

Edge Staff 21.4.2013: The Making Of: Infinity Blade. Edge. Saatavissa: https://web.archive.org/web/20130708135203/http://www.edge-online.com/features/the-making-of-infinity-blade. Viitattu 23.4.2021

Epic Games 16.12.2010: Epic Games Releases December 2010 Unreal Development Kit Beta. Epic Games. Saatavilla: http://web.archive.org/web/20110202160005/http://www.udk.com/news-beta-dec2010. Viitattu 31.3.2021

Epic Games 15.6.2020: A First Look At Unreal Engine 5. Epic Games. Saatavissa: https://www.unrealengine.com/en-US/blog/a-first-look-at-unreal-engine-5. Viitattu 12.4.2021

Epic Games 26.5.2021a: Blueprint Visual Scripting. Epic Games. Saatavissa: https://docs.unrealengine.com/4.26/en-US/ProgrammingAndScripting/Blueprints/. Viitattu 29.5.2021

Epic Games 26.5.2021b: Setting Up Android SDK and NDK for Unreal. Epic Games. Saatavissa: https://docs.unrealengine.com/4.26/en-US/SharingAndReleasing/Mobile/Android/Setup/AndroidStudio/. Viitattu 29.5.2021

Epic Games 26.5.2021c: Android Debugging. Epic Games. Saatavissa: https://docs.unrealengine.com/4.26/en-US/SharingAndReleasing/Mobile/Android/AndroidDebugging/. Viitattu 29.5.2021

Epic Games 13.5.2021d: Customized UVs. Epic Games. Saatavissa: https://docs.unrealengine.com/en-US/RenderingAndGraphics/Materials/CustomizedUVs/index.html. Viitattu 18.5.2021

GDC 26.11.2000: Geometric design and computation. Utahin yliopisto. Saatavilla: https://www.cs.utah.edu/gdc/history/. Viitattu 7.4.2021

Google 14.1.2021: Distribution Dashboard. Google. Saatavissa: https://developer.android.com/about/dashboards. Viitattu 23.3.2021

Graft, Kris, 2011: Epic's Rein: 'UDK Will Eventually Come To Android'. Gamasutra. Saatavissa: https://gamasutra.com/view/news/33958/Epics_Rein_UDK_Will_Eventually_Come_To_Android.php. Viitattu 18.4.2021

Capps, Mike – Jobs, Steve – Mustard, Donald 9.9.2010: Epic Games & Chair Reveal "Project Sword" at Apple Special Event. Video. Epic Games, Youtube-videopalvelu. Saatavissa: https://www.youtube.com/watch?v=p2H8hmi8yrQ. Viitattu: 20.4.2021

Jordan, Justine, 11.1.2021: What's the First 3D Game in the World. Narrasoft. Saatavissa: https://narrasoft.com/what-is-the-first-3d-game-in-the-world/. Viitattu 19.5.2021.

Kapoulkine, Anthony 26.05.2018: Reducing Vulkan® API call overhead. Gpuopen. Saatavissa: https://gpuopen.com/learn/reducing-vulkan-api-call-overhead/. Viitattu 31.3.2021.

Khronos: OpenGL ES Overview. Khronos. Saatavissa: https://www.khronos.org/opengles/. Viitattu 4.5.2021

Myslewski, Rick 16.1.2009: iPhone App Store breezes past 500 million downloads. The Register. Saatavissa: https://www.theregister.com/2009/01/16/half_billion_iphone_apps/. Viitattu 20.4.2021

Newzoo 4.2021: Most Popular PC Games | Global. Newzoo. Saatavissa: https://newzoo.com/insights/rankings/top-20-pc-games/. Viitattu: 23.4.2021

Nutt, Christian 19.3.2014: Epic radically changes licensing model for Unreal Engine. Gamasutra. Saatavissa: https://www.gamasutra.com/view/news/213517/Epic_radically_changes_licensing_model_for_Unreal_Engine.php. Viitattu 12.4.2021

O'Dea, Simon 8.2.2021: Mobile operating systems' market share worldwide from January 2012 to January 2021. Statista. Saatavissa: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/. Viitattu 25.3.2021.

Plant, Mike 21.12.2018: Top 10 best-selling videogame consoles. Guinness World Records. Saatavissa: https://www.guinnessworldrecords.com/news/2018/12/top-10-best-selling-videogame-consoles-551938. Viitattu 7.4.2021

Sarkas, Samit 6.11.2014: Microsoft officially owns Minecraft and developer Mojang now. Polygon. Saatavissa: https://www.polygon.com/2014/11/6/7167349/microsoft-owns-minecraft-mojang-acquisition-closes. Viitattu 2.5.2021

Smith, Ed 29.7.2015: Doing It Together: Remembering PlayStation's Net Yaroze Console. Vice. Saatavissa: https://www.vice.com/en/article/mvxbky/doing-it-together-remembering-playstations-net-yaroze-console-935. Viitattu 2.5.2021

Statcounter 31.3.2021: Mobile & Tablet Android Versions Market Share India. Statcounter. Saatavissa: https://gs.statcounter.com/android-version-market-share/mobile-tablet/india. Viitattu 31.3.2021.

Sweeney, Tim 2.3.2015: If You Love Something, Set It Free. Epic Games. Saatavissa: https://www.unrealengine.com/en-US/blog/ue4-is-free. Viitattu 12.4.2021

Todorov, Nickolay 16.11.2014: This was the world's first cell phone with a game loaded on it. Phonearena. Saatavissa: https://www.phonearena.com/news/This-was-the-worlds-first-cellphone-with-a-game-loaded-on-it_id62920. Viitattu: 7.4.2021

Travis, Jeff 20.2.2020: Forging new paths for filmmakers on ”The Mandalorian”. Epic Games. Saatavissa: https://www.unrealengine.com/en-US/blog/forging-new-paths-for-filmmakers-on-the-mandalorian. Viitattu 20.4.2021

Unity 28.10.2009: Unity 2.6 Released and Now Free!. Unity. Saatavissa: https://unity.com/our-company/newsroom/unity-2-6-released-and-now-free. Viitattu 4.5.2021

Wright, Chris 14.3.2016a: A Brief History of Mobile Games: In the beginning, there was Snake. Pocketgamer. Saatavissa: https://www.pocketgamer.biz/feature/10619/a-brief-history-of-mobile-games-in-the-beginning-there-was-snake/. Viitattu: 23.4.2021

Wright, Chris 17.3.2016b: A Brief History of Mobile Games: 2002 - Wake up and smell the coffee. Pocketgamer. Saatavissa: https://www.pocketgamer.biz/feature/10705/a-brief-history-of-mobile-games-2002-wake-up-and-smell-the-coffee/. Viitattu 23.4.2021

Ziegler, Chris 17.12.2010: Dungeon Defenders: First Wave brings Unreal Engine to Android this month. Engadget. Saatavissa: https://www.engadget.com/2010-12-17-dungeon-defenders-first-wave-brings-unreal-engine-to-android-th.html. Viitattu 23.4.2021