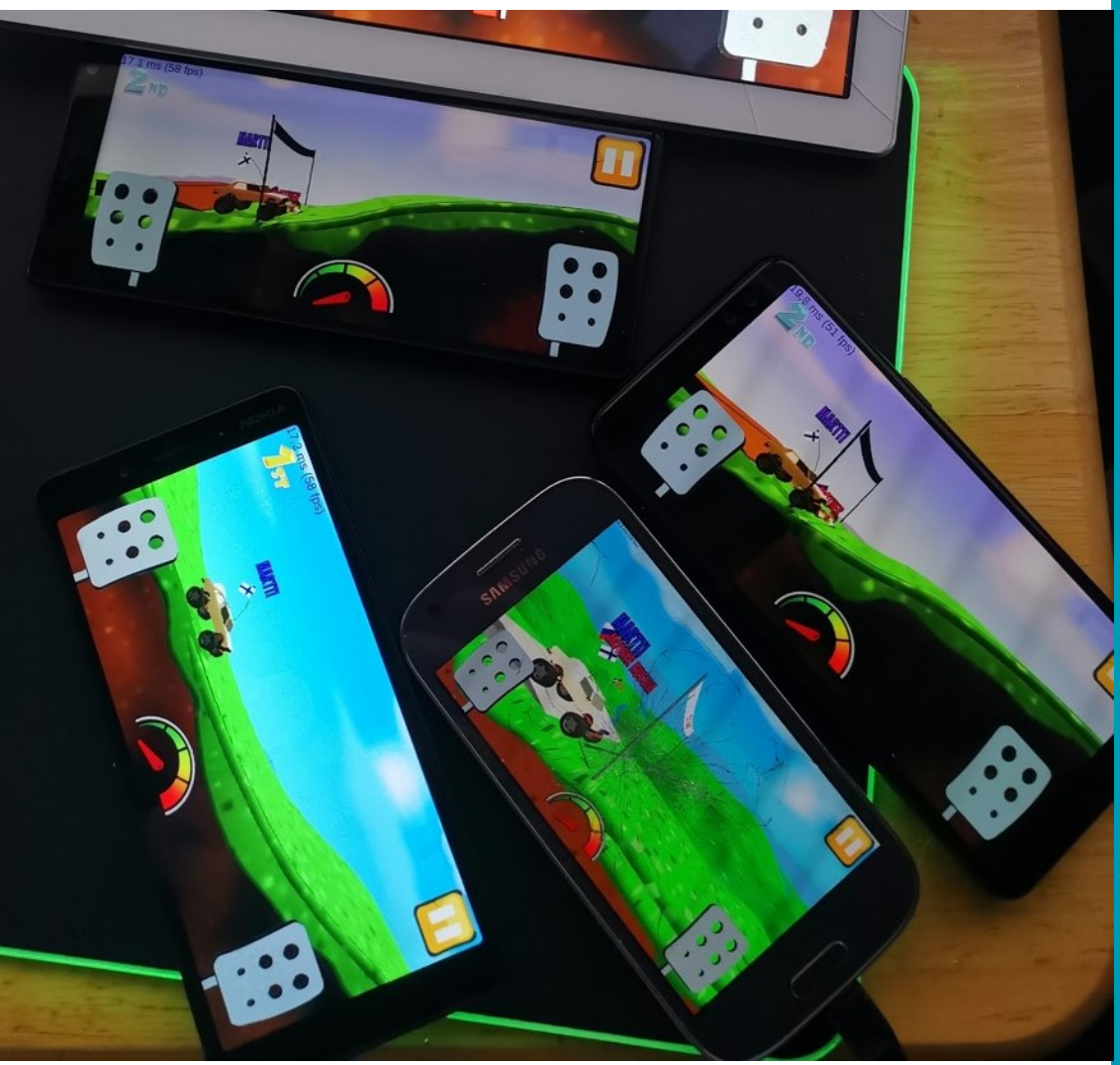
Jere Puusa

3D-grafiikan optimointi mobiililaitteelle



Tradenomi (AMK)

Tietojenkäsittely

Kevät 2022

KAMK • University of Applied Sciences

KAMK • University of Applied Sciences

Tiivistelmä

Tekijä: Puusa Jere

Työn nimi: 3D-grafiikan optimointi mobiililaitteelle

Tutkintonimike: Tradenomi (AMK), tietojenkäsittely

Asiasanat: 3D-mallinnus, optimointi, Unity, mobiilipelit

Tämän opinnäytetyön tavoitteena oli kerätä tietoa ja esitellä, kuinka tärkeää 3D-mallien optimointi on pelimoottorissa käyttöä varten ja miten toteuttaa sitä. Työ keskittyy mobiililaitteille optimointiin ja käyttää esimerkkinä itse kehitettyä mobiilipeliä. Internetistä on haastavaa löytää yksittäistä, tiivistä tekstiä, joka keskittyisi kertomaan optimoinnista, joten 3D-grafiikan alkajille voi olla vaikeaa löytää aiheesta tietoa. Tämä teksti voi toivottavasti auttaa tässä pulmassa.

Työ pohjautuu mobiilipeliin ja sen kehityksen aikana tehtyihin havaintoihin sekä internetin tietolähteisiin. Työssä selitetään 3D-mallinnuksen perusteet ja käsitteitä, jotta työtä voivat lukea myös 3D-mallinnuksen aloittelijat ja asiaan perehtymättömät. Työ sisältää useita tapoja optimoida malleja sekä tekstuureja, jotta niillä päästäisiin mahdollisimman hyvään suorituskykyyn. Optimoinnin käytännöt pätevät mobiilipelikehityksen lisäksi myös esim. konsolipelien ja fotorealistisen tyylisuunnan pelien kehittämiseen, sillä optimoidut mallit tuottavat tulosta alustasta huolimatta ja parempi suorituskyky on aina arvostettua. Asiaa käsitellään tekstin kautta ja havainnollistetaan kuvien avulla. Työssä esitellään mobiilipeli, jossa on käytetty esitettyjä optimointitapoja ja -perusteita. Työ päättyy testiin, jossa vertaillaan optimoitujen ja optimoimattomien mallien eroja suorituskykyyn ja pohditaan tuloksien eroja.

Työssä testataan optimoidun ja optimoimattoman mallin eroja suorituskykyyn. Tämä näyttää lukijalle, miksi optimointi on tärkeä taito oppia 3D-graafikkona. Tuloksista voi nähdä huomattavan eron kahden eri skenen välillä, mutta lopputulokset voisivat olla paremmin kirjattuja; mm. prosessorin käyttöä ei tallennettu ja testaukset suoritettiin pöytätietokoneella eikä mobiililaitteella. Paremman ja kattavamman testauksen voisi suorittaa vielä useilla mobiililaitteilla, mutta tämän työn testistä voi silti nähdä eron optimoinnin eduista. Työ voisi sisältää myös muita tapoja optimoida sekä peliskeneä Unityssä, että etenkin tapoja optimoida 3D-mallinnusohjelman sisällä, kuten UV-kartan ja mallin saumojen optimointia. Toivon, että opinnäytetyö voi auttaa aloittelevia 3D-graafikoita pääsemään nopeammin alkuun ja välttämään tekemään tekemästä joitain virheitä, kuten liian suuriresoluutioisia malleja tai tekstuureja sekä vähentämään materiaalien käyttöä.

KAMK • University of Applied Sciences

Abstract

Author: Puusa Jere

Title of the Publication: 3D Graphics Optimization for Mobile Platforms

Degree Title: Bachelor of Business Administration, Business Information Technology

Keywords: 3D-modelling, optimization, Unity, mobile games

The goal of this Bachelor's thesis was to collect information on optimizing 3D-models and to show how important it is in game engines, and how to go about doing it. The work focuses on optimization for mobile devices and uses a self-developed mobile game as an example. It is difficult to find a single, compact text on the internet that focuses solely on optimizing graphics, so it may be difficult for a beginner to find information on the subject. This text aims to partially address that issue.

This work is based on a self-developed mobile game, and observations made and lessons learned during its development, as well as sources from the internet. The basics and some terminology of 3D graphics are explained in the text to make it readable for beginners and newcomers to 3D modelling. The work includes multiple ways to optimize models and textures to maximize their performance in a game. The principles of optimization apply not only to mobile game development, but also, for example, to developing console games or games with photorealistic art styles, because improved performance is always appreciated regardless of the target platform. The subjects are explained through text, paired with image examples. The work ends with an introduction of a self-developed mobile game and a test in which a scene inside the game is compared with an intentionally unoptimized version of the same scene. The results of this test are listed and analyzed, and conclusions are drawn of how beneficial optimization is.

The work tests the performance impact of having optimized models versus unoptimized models in a scene in a video game. The results of this test show why optimization is an important skill to learn as a 3D artist. The results show a noticeable difference between the performance of two nearly identical-looking scenes inside the Unity game engine, although the test could have benefited from having more metrics measured such as CPU usage. The test could benefit from being tested on more devices, as it was performed on a desktop PC rather than a mobile device, however even in this limited test one can see the benefits of optimization. This thesis could also benefit from including other ways to optimize game engine scenes and 3D models, such as by optimizing UV maps or seam placements on models. Hopefully this thesis can help beginner 3D artists to get started faster and to avoid making certain mistakes shown here, such as making models or textures with too large resolutions, or minimizing the number of materials used.

Alkusanat

Tämän opinnäytetyön tavoitteena oli luoda aloittelevalle 3D-graafikolle helposti ymmärrettävä teksti optimoinnista. Aloittelijana en löytänyt kattavaa tietopakettia, jolla luoda pohjaa taidoilleni, joten halusin luoda jotain millä päästä alkuun mallintajana. 3D-mallinnus on luovaa ja hauskaa tekemistä, joka on helppo aloittaa, kunhan tietää perusteet. 3D-mallinnukseen on monta tapaa sekä ohjelmistoa, joista ilmaisohjelmat ovat erittäin hyvä tapa aloittaa ja toimivat usein myös ammattilaisten käytössä. Pelikäyttöön valmiin mallin tekeminen on oma haasteensa, joka luo omat rajoitteensa. Kun tietää nämä rajoitteet, voi 3D-mallin luonnin aloittaa oikeilla tavoilla, jolloin mallista tulee pelivalmis ilman myöhempiä jatkotoimenpiteitä mallin parantelua varten. Näin graafikko voi säästää aikaa ja vaivaa.

Haluan kiittää ammattikorkeakoulun henkilökuntaa erinomaisesta opetuksesta viime vuosina. Samoin kiitokset Markku Hietamäelle hienosta työstä ohjelmoijana peliprojektissamme ja avusta Unity-pelimoottorin opettelussa sekä tämän työn kansikuvan ottamisesta. Kiitokset kuuluvat myös Janne Joensuulle hänen 3D-alan sanastostaan, siitä oli suuri apu kääntäessä termejä suomen kielelle.

Sisällys

Symboliluettelo

CPU	Tietokoneen keskusprosessoriyksikkö, vastaa laskennoista.
GPU	Tietokoneen näytönohjain, vastaa kuvan piirtämisestä.
LOD	Level of detail. Yksinkertaistettu versio mallista, joka on tarkoitus olla näkyvillä vain kauempaa tai muissa tietyissä oloissa. Voi olla käsin tai automaatiolla tehty.
Materiaali	Tahkolle piirrettävän värityksen tiedon omaava kokonaisuus. Yleensä säilöö dataa käytettävästä tekstuurista ja varjostimesta.
Modifikaattori	Työkalu Blender-ohjelmistossa, jolla mallia voi muokata parametrejä säätämällä modifikaattorin tavan mukaisesti. Esimerkiksi peili-modifikaattori luo mallista peilatun kopion käyttäjän valitseman X, Y ja/tai Z akselin mukaisesti. Subdivision-modifikaattori puolestaan lisää geometriaa leikkaamalla jokaisen pinnan halki ja lisäämällä siihen verteksin, luoden kaarevampia, yksityiskohtaisempia muotoja.
PBR	Physically based rendering. Tosielämän valaistusta matkiva renderöintitapa 3D-grafiikassa. Vaatii useiden päällekkäisten tekstuurien ja monimutkaisten materiaalien käyttöä.
Renderöinti	Kuvan piirtäminen ruudulle grafiikkaprosessorin kautta.
(UV-)Sauma	3D-mallin reunoihin tehtävä merkintä, joka osoittaa 3D-ohjelmalle, missä mallin tulisi jakaantua UV-saarekkeisiin teksturointia varten.
Skene	Pelimoottorin sisäinen 3D-avaruuden sisältävä tallennettava tiedosto, joka voi sisältää 2D sekä 3D-kappaleita ja jossa pelin sisäinen koodi pyörittää pelin logiikkaa.
Särmä	Kahden verteksin välinen viiva 3D-ulottuvuudessa. Osa 3D-mallia.

Tahko	Kolmen tai useamman verteksin väliin piirretty 3D-ulottuvuudessa sijaitseva geometrinen monikulmio. 3D-mallin näkyvät pinnat koostuvat näistä.
Tekstuuri	Tahkoon piirrettävä kuvatiedosto.
Tekstuuriatlas	Kuvatiedosto, johon on koottu useita tekstuureita vähentämään ladattavien tiedostojen määrää.
Topologia	Geometrisen mallin osien jakautuminen, tahkojen rakenne.
UV-kartta	2D-visualisointi kolmiulotteisen mallin pinnoista.
Varjostin	Koodi, joka kertoo GPU:lle miten mallin pinta valaistustaan ja väritetään.
Verteksi	3D-ulottuvuudessa sijaitseva piste. Osa 3D-mallia.

1 Johdanto

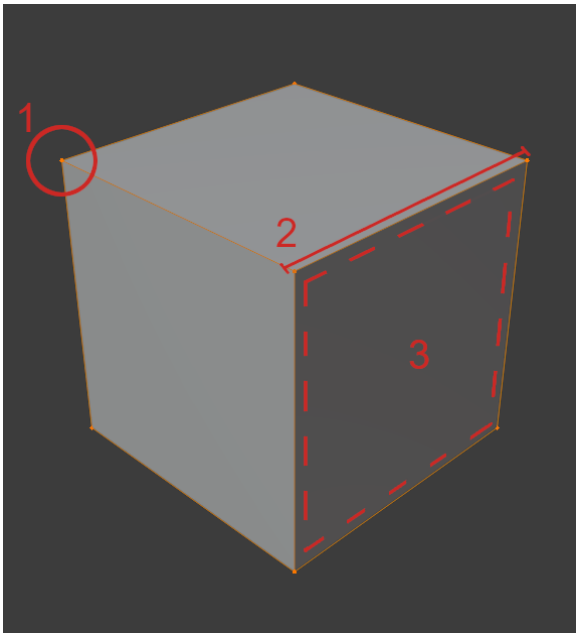
Tämän opinnäytetyön aiheena on 3D-pelin kehitys ja grafiikka mobiililaitteille. Työssä käytettään tapausesimerkkinä Android-mobiilialustoille kehitettyä videopeliä nimeltä Eggspress Delivery, joka on ollut kehityksessä kesästä 2020 lähtien. Tässä työssä esitetyt optimointitekniikat koskevat osittain pelkästään kyseisen pelin käyttämää Unity-pelimoottoria ja sen käyttöä, mutta suurin osa koskee 3D-mallinnusta sekä kolmiulotteisen pelin kehitystä yleisellä tasolla. Työssä esitellyt tavat optimoida peliä mobiilialustalle eivät ole uusia kehityksiä alalla, vaan kooste yleisesti tunnetuista sekä itse opituista tavoista päästä mahdollisimman hyvään suorituskykyyn mobiililaitteilla. Työ on suunnattu madaltamaan kynnystä aloitteleville peligraafikoille jakamalla näitä tapoja.

Pelin kehityksen alkuvaiheessa tulee päättää, miltä lopullinen peli tulee näyttämään. Tämä taiteen tyylilajin valinta vaikuttaa lopullisen ulkonäön lisäksi peligraafikoiden työn määrään ja tämän työn kannalta tärkeimmin, suorituskykyyn. Valitsemalla tyylittely realismin sijaan voidaan pärjätä pienempiresoluutioisilla tekstuurikartoilla sekä vähemmillä määrillä tekstuurikarttoja ja materiaaleja. Päätöksillä varhaisessa suunnitteluvaiheessa voidaan vaikuttaa myöhempään lopputulokseen ilman, että aiemmin valmistettuja asioita tarvitsee muuttaa tai korjata. Esimerkiksi päättämällä käyttää tekstuuriatlaksia jo heti alusta ei myöhemmin tarvitse korjata jokaiseen malliin uudestaan materiaaleja, UV-kartoitusta, sekä värjäystä.

Työ perustuu lähinnä omiin pelinkehityksen aikana tapahtuneisiin havaintoihin sekä internetin tietolähteisiin. Internetistä löytyy hajautetusti tietoa grafiikan optimoinnista graafikoille. Siihen verrattuna ohjelmoijille löytyy tietoa ohjelmien optimoinnista huomattavasti enemmän, joka ei pääasiassa ole tärkeää tietoa graafikolle. Työn tavoite on kerätä näitä tietoja graafikoille ja havainnoida kuvin ja tilastoin sekä testin avulla menetelmien tehokkuutta ja optimoinnin tärkeyttä.

2 3D-mallinnus ja pelimoottori

Tietokoneilla piirretty kolmiulotteinen grafiikka perustuu geometrisiin muotoihin, jotka koostuvat pisteistä 3D-avaruudessa. Näitä pisteitä kutsutaan vertekseiksi. Kaksi verteksiä yhdistämällä luodaan viiva, jota kutsutaan särmäksi, ja vähintään kolme särmää yhdistämällä saadaan luotua pinta, toiselta nimeltään tahko. Näitä tahkoja luomalla ja yhdistelemällä saadaan luotua 3D-malli. Tahkoksi lasketaan mikä tahansa monikulmio (eli polygoni), mutta tahko, jossa on kolme särmää, on nimeltään kolmio. Kolmiolla on englannin kielestä juontuva lempinimi tri. [1, s. 8.] Kuvassa 1 voi nähdä yksinkertaisen kuution, johon on merkitty nämä kolme geometristä perusmuotoa.

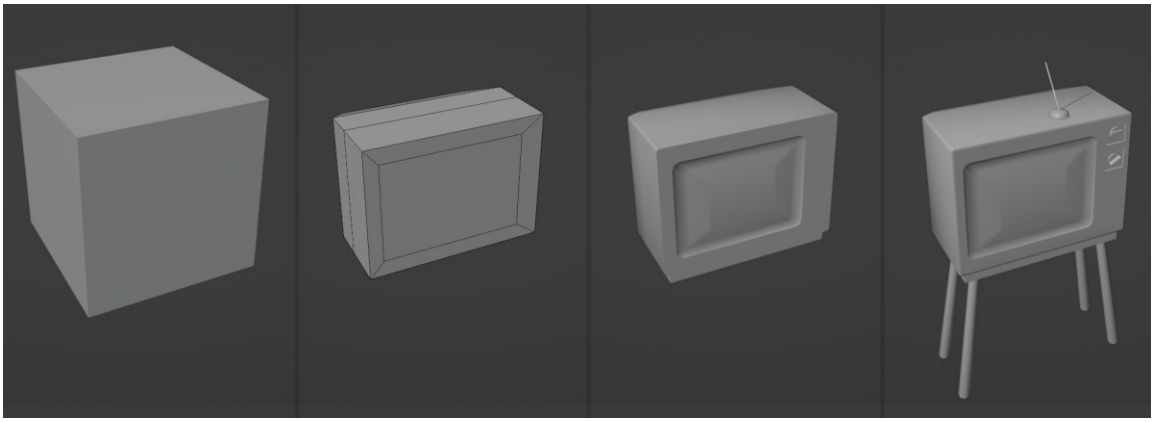


Kuva 1. Havainnollistus 3D-mallin osista. Numero 1 on verteksi, numero 2 on särmä, numero 3 on tahko.

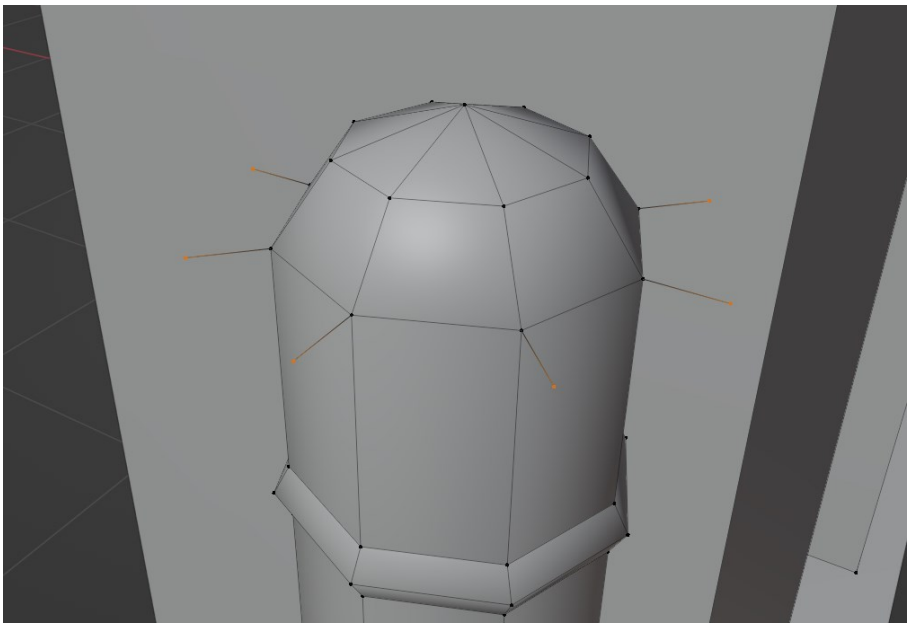
Kolmio on yksinkertaisin muoto, johon tahko voidaan jakaa, ja se ei voi olla epätasainen. Trigonometria on myös nopea laskutoimitus tietokoneelle. Näistä syistä kolmioita käytetään useimmiten 3D-mallinnuksessa laskemaan mallin monimutkaisuus [1, s. 8]. Esimerkiksi vaikka kuutiossa on 6 tahkoa, koska jokainen neliö voidaan puolittaa kolmioksi, niin sanottaisiin mallin monimutkaisuudeksi 12 kolmiota.

2.1 3D-mallinnus

Mallintaminen aloitetaan usein kuutiolla, tai muulla yksinkertaisella kappaleella, jota muotoillaan eri tavoilla. Tapoja ovat esim. osion venyttäminen, neliöiden jakaminen, tahkojen leikkaaminen, sekä verteksien ja reunojen lisääminen ja yhdistely. Tätä kuutiolla tai muulla peruskappaleella aloittamista kutsutaan laatikkomallintamiseksi (box modeling), kuvasta 2 voi nähdä esimerkin tästä tavasta. Muita tapoja ovat mm. veistäminen, NURBS-käyrien avulla mallintaminen [1, s. 8] tai 3D-skannaaminen. Mallintaessa on hyvä kiinnittää huomioita paitsi mallin taiteelliseen visuaaliseen ilmeeseen ja muotoon, myös hyviin optimointikäytänteisiin, joiden avulla laskea lopullisen mallin kolmioiden määrää. Tämän voi tehdä esimerkiksi poistamalla piilossa olevia tahkoja tai ylimääräisiä verteksejä, jotka eivät ole osana kolmiota (kuva 3). Myös lähellä toisiaan olevat verteksit voidaan joissain tapauksissa yhdistää ilman, että visuaalinen laatu kärsii [1, s. 9].



Kuva 2. Esimerkki laatikkomallinnuksella toteutetun TV-mallin vaiheista.

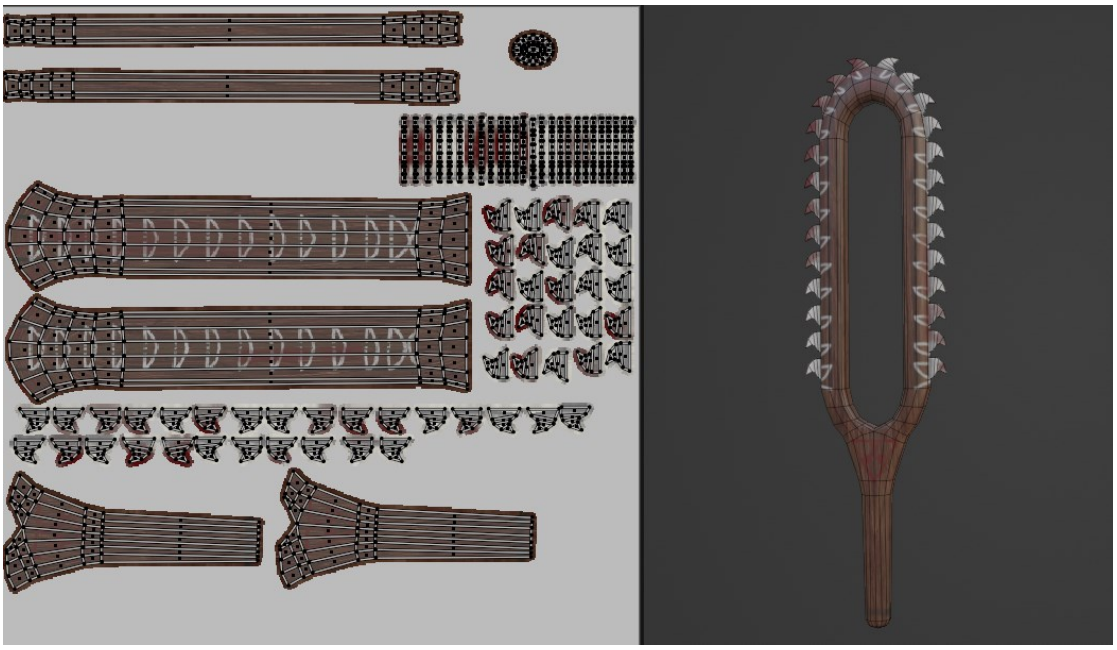


Kuva 3. Esimerkki mallista, jossa on jäänyt ylimääräisiä verteksejä, jotka voidaan poistaa. Oranssilla merkityt verteksit eivät ole yhdistettynä tahkoiksi, joten niitä ei voi nähdä pelimoottorin sisällä.

Mallinnettaessa on hyvä käyttää pääasiassa nelikulmaisia tahkoja. Myös kolmioita voi käyttää tietyissä solmukohdissa. Kolmioita tulee silti välttää, samoin kuin tahkoja, jotka koostuvat useammasta kuin neljästä särmästä, koska 3D-mallinnusohjelmat eivät voi jakaa tällaisia tahkoja automaattisesti pienempiin osiin suoralla- eli silmukkaleikkauksella.

2.2 UV-kartoitus

Mallinnuksen yksi vaihe on UV-kartoitus, jossa mallin tahkoista heijastetaan kaksiulotteinen versio niin sanottuun UV-karttaan. Tätä UV-karttaa käytetään määrittämään mille alueille heijastetaan tekstuuritiedoston pikselit. Tahkot asetellaan niin sanotuissa saarekkeissa yleensä mahdollisimman tiiviisti ilman, että tahkot ovat päällekkäin. Joissain teksturoinnin tavoissa, kuten listatekstuurien tai tekstuuriatlaksien käytössä, nämä säännöt eivät päde ja päällekkäisyydet ovat tarkoituksellisia. Kuvassa 4 on esimerkki UV-kartasta mallineen.



Kuva 4. Esimerkki 3D-mallista ja sen UV-kartasta. Kuvasta voi nähdä, miten jokainen mallin tahko on heijastettu UV-karttaan. UV-karttaan on jäänyt tyhjää tilaa, sillä mallin tekselitiheys on haluttu pitää tarkkana. Tekselitiheys kertoo, kuinka paljon pikseleitä halutaan näkyä tietyllä neliösenttimetrillä. Tässä mallissa on haluttu näyttää 5,12 pikseliä tekstuuria jokaista neliösenttimetriä kohden, jonka vuoksi UV-karttaan sekä tekstuurikarttaan on jäänyt paljon ylimääräistä tilaa. Tätä tilaa olisi voitu hyödyntää, mikäli malli olisi suurempi, yksityiskohtaisempi, tai mikäli tekselitiheys olisi pienempi. Yhteen tekstuuriin voitaisiin myös yhdistää useita malleja. Mallin eri osilla voi olla eri tavoitetiheydet ja tahkot voivat myös poiketa hieman halutusta tiheydestä, mikäli halutaan hyödyntää tyhjää UV-kartan tilaa. Tärkeää on, että tekselitiheyden erot eivät ole liian huomattavia lopullisessa mallissa.

2.3 Tekstuurit ja materiaalit

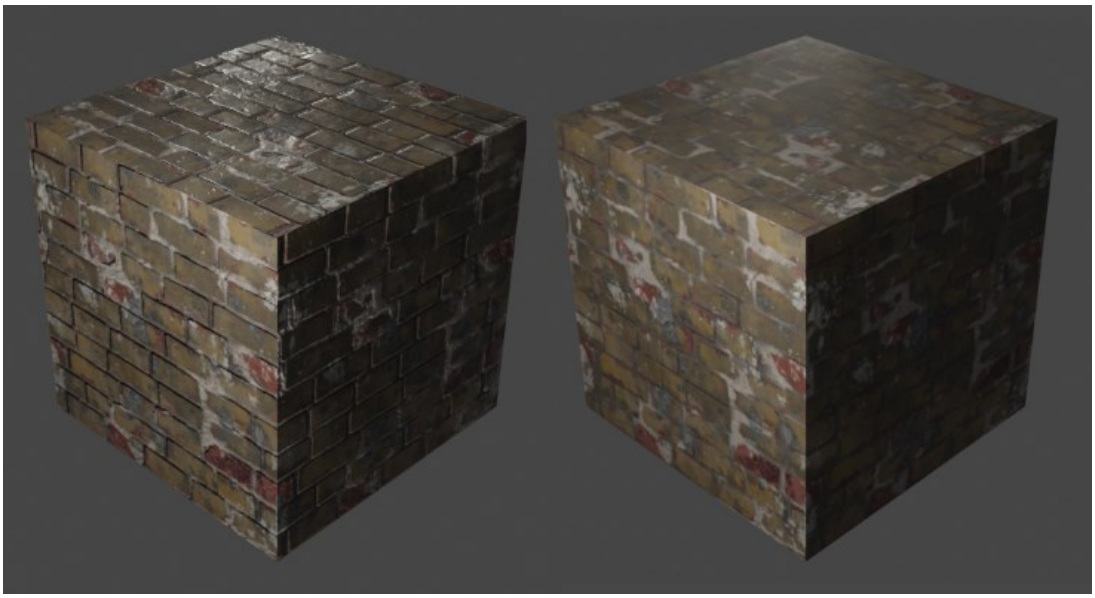
Tekstuuriksi, tai tekstuurikartaksi, kutsutaan bittikarttakuvaa, jota esitetään 3D-mallin pinnalla. Nämä tekstuurit esittävät pinnan ominaisuuksia, riippuen tekstuurin käyttötarkoituksesta. Pelimoottori lukee tekstuureja osana materiaaleja. Materiaalit ovat komponentteja, jotka sisältävät tietoa siitä, miltä pinnan tulisi näyttää, miten moottorin tulisi piirtää pintoja ja miten tulkita sille syötettyjä tekstuureja ja sävyttimiä [2, s. 137]. Materiaalit voivat myös toimia proseduraalisesti

luotuna tai käsin syötetyin arvoin, jolloin tekstuureja ei ole pakko käyttää tai niitä voi käyttää yhdessä muiden syötettyjen arvojen kanssa.

Physically based rendering, lyhennettynä PBR, on renderöintitekniikka, joka pyrkii fotorealistiseen kuvanlaatuun matkimalla tosielämän valaistuksen ja pintojen fyysisiä ominaisuuksia. PBR on pelialalla vakituisessa käytössä yleisesti lisääntyneen laitteiden suorituskyvyn ansiosta. [3.] [4.]

PBR:ssä 3D-mallin pinnoille piirretään päällekkäin useita tekstuurikarttoja, jotka vaihtelevat PBR-tavan mukaan. Esimerkiksi metallic-/roughness -tavassa pinnalle voidaan luoda pohjavärikartta, joka sisältää pinnan näkyvät värit. Tämän pohjavärin valaistusta voidaan reaaliajassa muokata sen ympäristön valaistuksen perusteella, jolloin se saadaan näyttämään enemmän fotorealistiselta. Pohjavärikartan lisäksi pinnalle voidaan antaa normaalikartta, jonka on tarkoitus saada tasainen pinta näyttämään syvältä halutuissa kohdissa. Nämä molemmat kartat ovat yleisiä muissakin PBR-tavoissa, kuten specular/glossiness. Metallic-/roughness -tavassa pinnan heijastuvuus määritellään eri karttojen avulla kuin specular/glossiness, mutta yleisellä tasolla lopputulos voi näyttää hyvin samalta. [5.] PBR:n käytön vaikutuksen piirrettävään pintaan voi havaita kuvassa 5.

PBR:än käyttö pelissä vie kuitenkin enemmän muistia kuin ilman, sillä PBR-tekniikassa sama pinta vaatii useita tekstuurikarttoja piirrettäväksi. Tästä esimerkkinä pinta, joka ei muuten vaatisi kuin pohjavärin näyttääkseen oikealta, vaatisi metallic/roughness-tavan PBR:ssä pohjaväritekstuurin lisäksi ainakin metallisuus-, karkeus- sekä normaalikartta-tekstuurit. Tällöin saman pinnan väritys ja valaisu veisi monta kertaa enemmän välimuistia tekstuurien määrän vuoksi, ja renderöinti veisi lisätehoja grafiikkasuorittimelta. Mobiililaitteilla on usein puute välimuistista, minkä vuoksi PBR:n käyttö on harvinaista kyseisillä laitteilla. Joitain PBR:n tapoja, kuten ambient occlusionia, voidaan imitoida ”leipomalla”, eli piirtämällä päälle, kyseisen tekstuurin dataa pohjavärikarttaan. Tämä leivottu data ei reagoi valaistukseen samalla lailla, mutta voi näyttää joissain tilanteissa paremmalta kuin jättää tekstuurikartta leipomatta.

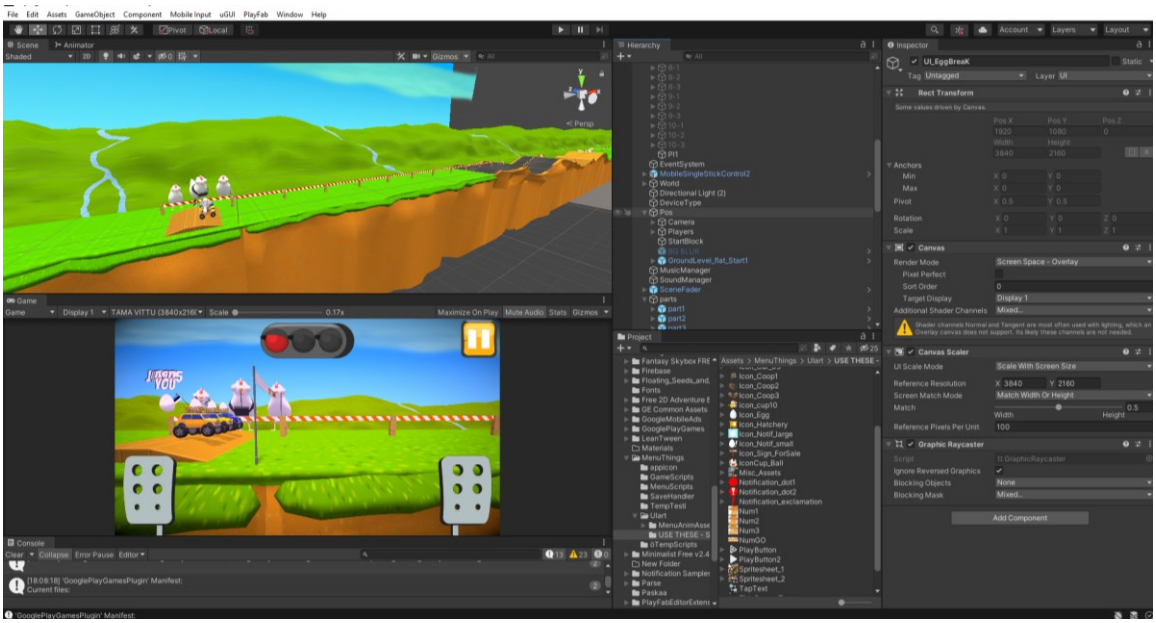


Kuva 5. Vasemmalla valaistu 3D-malli, jossa käytössä PBR-kartat pohjaväri, normaalikartta sekä kiiltävyys. Oikealla sama malli samalla valaistuksella, mutta käytössä pelkkä pohjaväritekstuuri. Pelkkä pohjaväritekstuuri on kooltaan 0,839 megatavua. Normaalitekstuuri vie puolestaan muistia 22,3 megatavua. Käytetyt tekstuurit ovat resoluutioltaan 2048 pikseliä leveät ja korkeat, ja ovat pakattu PNG-tiedostomuotoon, paitsi pohjaväri on JPEG-tiedostomuodossa. Vasen malli on visuaalisesti miellyttävämpi ja fotorealistisempi, mutta vie huomattavasti enemmän muistia ja heikentää suorituskykyä. Valitsemalla pelille ei-fotorealistinen tyyli, voidaan helpommin pärjätä pelkillä pohjaväritekstuureilla. Tällöin realistisen valaistuksen puute ei haittaa, vaan on osa tyyliä.

2.4 Pelimoottori

Pelimoottoriksi kutsutaan ohjelmistoa, joka on kehitetty erityisesti pelin kehittämiseen. Pelimoottorit sisältävät tyypillisesti mm. järjestelmiä tekoälyä, äänentoistoa ja animaatioita varten sekä moottorit fysiikan, renderöinnin ja valaistuksen laskentaa sekä lopullista koodin kääntämistä ja pelin pakkaamista varten. Usein myös analytiikkatyökalut ja joskus jopa ohjelmointiin tarkoitetut ohjelmankehitysympäristöt ovat osa pelimoottorin kehitysohjelmistoa. Näin pelin kehitystä aloittaessa kehittäjien ei tarvitse luoda näitä järjestelmiä tyhjältä pöydältä, vaan he voivat käyttää valmiita ominaisuuksia ja muokata niitä tarvittaessa pelin vaatimusten mukaan. [6, kpl 1.]

Kaksi pelialalla yleisessä käytössä olevaa pelimoottoria ovat Unity sekä Unreal Engine. Kuvassa 6 näkyy peli avattuna Unity-pelimoottorin käyttöliittymään.



Kuva 6. Kuvakaappaus Unity-pelimoottorin version 2021.1.2f1 käyttöliittymästä, jossa on avattuna Eggspress Delivery -pelin kehitysnäkymä.

2.5 Mallien ja tekstuurien rajoitteita

Mallien ja tekstuurien koissa on rajansa, missä graafikon on kannattavaa pysytellä. Pelin tyylisuunnasta sekä alustasta (PC-, konsoli- vai mobiilipeli) riippuu, kuinka yksityiskohtainen malli voi olla. Pelin kehityksen alkuvaiheessa, eli esituotannossa, kehittäjätiimin johtava graafikko päättää alustavasti mallien yksityiskohtaisuudesta, ja graafikot toimivat näissä annetuissa puitteissa. Näitä rajoitteita voidaan myöhemmin myös tuotannon aikana muokata. Mikäli tapahtuu suuria muutoksia, saattavat aiemmat mallit vaatia muokkauksia ja lisätä työtaakkaa, joten esituotannossa on hyvä olla tarkkana näitä rajoitteita laatiessa. Tiukemmat rajoitteet pienentävät asennetun pelin kokoa, vähentävät muistin käyttöä, nostavat kuvataajuutta sekä vähentävät lataamisen viemän ajan pituutta.

Konsolipeliä kehittäessä hahmon kolmiomäärä voi olla, hahmon tarkoitusperän mukaan, 10 000–100 000 kolmiota, ja peliympäristön taustalavasteena toimiva malli eli proppi voi sisältää 1000–

20 000 kolmiota. Mobiililaitteet ovat prosessointiteholtaan, välimuistiltaan, tallennustilaltaan sekä muilta ominaisuuksilta pelikonsoleita sekä pöytätietokoneita heikompia ja hitaampia, siksi niille kehitettävät pelit ovat rajoitetumpia yksityiskohtaisuudeltaan. Mobiilipelin hahmot voivat koostua n. 1000–10 000 ja propit 100–2 000 kolmiosta. Tekstuurien tulee olla mahdollisimman pieniä, halutusta yksityiskohtien määrästä riippuen. Nämä summat ovat kuitenkin vain viitteellisiä, ja oikea määrä riippuu useasta tekijästä, kuten taiteen tyylisuunnasta, mallien koosta ja määrästä, mallien käyttötarkoituksesta (esim. onko malli vain taustalavaste vai keskeinen osa peliympäristöä) ja tuleeko malli animoida. [7; 8.]

2.6 Testaaminen

Pelin testaaminen on kehitysvaiheen osa, jossa versiota pelistä testataan laadun varmistamiseksi ja uusien parannusehdotusten ja ideoiden löytämiseksi. Peliä kehittäessä on erittäin hyvä testata sitä jo varhaisessa vaiheessa. Testaamisella on monta tarkoitusta: mm. etsiä vikoja, joita korjata, tai kokeilla uusia ominaisuuksia, jotka ovat prototyyppivaiheessa ja kehittäjät eivät ole päättäneet, jäävätkö ominaisuudet osaksi peliä. Pelin testaamisen voi aloittaa heti ensimmäisen prototyypin valmistuttua. Testaamiseen on monta tapaa, kuten datan tallentaminen ja analysointi. Käyttäjillä testaaminen on toinen tapa. [9.]

Testaaminen on tärkeä osa kehitystä. Sen avulla havaitaan mahdolliset virheet, joita kehittäessä lähes aina tulee. Nämä virheet voivat johtaa niiden määrästä ja laadusta riippuen pelin laadun huononemiseen. Huonolaatuinen peli saa huonon maineen, ei myy hyvin ja voi vaikuttaa kehittäjän ja julkaisijan maineeseen ja tulevaan menekkiin. Jotkin kaupat ja alustat myös ylläpitävät kriteereitä, jotka tulee täyttää, jotta peli voidaan julkaista kyseisellä alustalla. Tämä vaatii testausta laadun varmistamiseksi ja ajan tuhlaamisen välttämiseksi. [10, kpl 3.]

3 Optimointi

Yksi tapa mitata suorituskykyä on laskea pelin kuvataajuus, englanniksi frames per second (lyhennettynä ”FPS”), jolla tarkoitetaan lukua, kuinka monta kertaa sekunnissa ruudun kuva päivittyy. Useimmat näytöt pystyvät päivittämään kuvansa 60 kertaa sekunnissa, jolloin niiden virkistystaajuuden sanotaan olevan 60 hertsiä. Tällaisella näytöllä pelattavan pelin optimaalinen FPS on myös 60, sillä synkronoimalla näytön virkistystaajuus sekä pelin kuvataajuus voidaan välttää teknisien ongelmien esiintymistä. [11.]

Ruudun virkistystaajuuden ollessa 60 hertsiä, optimaalisin aika jokaisen ruudun piirtämiseen on noin 16,7 millisekuntia. Tämä luku johtuu siitä, että se on millisekunneissa lähin pyöristetty luku, joka tulee jaettaessa sekunti 60 osaan: $\frac{1}{60} = 0{,}16666\ldots$ S. Millisekunteja ruudun päivitysten välillä käytetään pelinkehityksessä tapana laskea pelin suorituskyky, varsinkin optimoidessa. Joissain peleissä, missä pelaajan reaktioajan maksimointi on tärkeää, on hyvä jättää kuvataajuus ja virkistystaajuus lukitsematta. Tällöin ruudulla näkyy viimeisin mahdollinen kuva ja minimoidaan viivästys kuvan piirtämisen ja näyttämisen välillä. Joissain peleissä voi tosin esiintyä teknisiä ongelmia, kuten kuvan repeytymistä. [12.]

Paremmin optimoitu peli kuluttaa vähemmän akkua mobiililaitteella, on pienempi asennuskooltaan ja voi olla suositumpi eli saada paremmin latauksia sovelluskaupassa [2, s. 14; 13]. Optimoimalla pelin voi saada sen asennettavaksi vanhemmille laitteille. Luonnollisesti peli, jonka voi asentaa useammalle laitteelle, voi saada enemmän latauksia.

Kun peli ei pyri tyylillään fotorealismiin, visuaalinen ilme ei kärsi PBR-materiaalien puutteen vuoksi. Fotorealismiin pyrittäessä päädytään usein myös käyttämään huomattavasti enemmän kolmioita malleissa. Kun pinnat eivät vaadi yksityiskohtia, säästetään materiaalien ja tekstuurien määrän ja monimutkaisuuden lisäksi kolmiomäärässä. Yksityiskohtien vähyys voisi myös vähentää graafikoiden työtaakkaa, mutta graafikoiden tulee varmistaa, että heidän mallinsa sopivat pelin taiteen tyylin kanssa yhteen. Yhdeksi haasteeksi tosin ilmaantuu yksityiskohtien määrän ja suorituskyvyn tasapainottaminen – suurempi kolmiomäärä ja tekstuurikoko kulkevat käsi kädessä tietokoneohjelman pienenevän kuvataajuuden kanssa. Esimerkiksi niin sanottu low-poly-tyylinen peli voi olla käyttämättä PBR:ää, mutta näissäkin tyyleissä on eroja ja yksi low-poly-peli voi olla

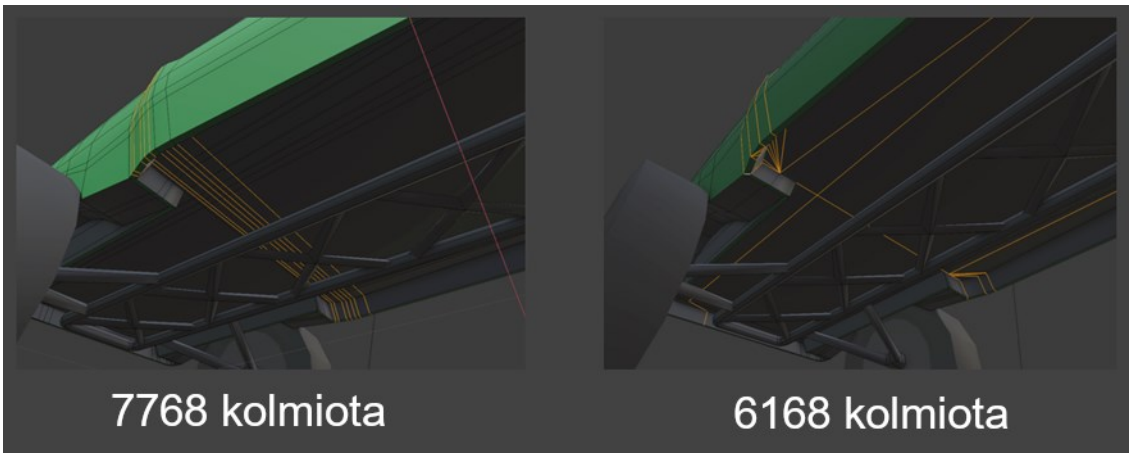
huomattavasti ”sarjakuvamaisempi” ja muodot voivat olla liioitellumpia kuin toisessa. Riippuu vahvasti siis tyylisuunnasta, kuinka tyyliteltyä jäljen tulee olla.

Mallien ja tekstuurien säädön lisäksi peliä voi optimoida muillakin tavoilla, kuten poistamalla käytöstä kuvan jälkikäsittelytoiminnot, esim. ylimääräiset heijastukset ja ympäristövarjostus. Materiaaleja luodessa tulee myös välttää läpinäkyvyyden käyttöä, sillä sen laskenta voi olla huomattavasti raskaampi GPU:lle. [14.] Partikkeliefektit käyttävät usein läpinäkyviä tekstuureita, kuten myös kasvien lehdet, ja voivat olla raskaita suorituskyvylle, joten niitä on käytettävä harkiten [15]. Tapoja optimoida on lukuisia ja niiden esiintyvyys ja hyöty vaihtelevat pelimoottorien välillä.

3.1 Mallien optimointi

3D-mallissa tulee olla mahdollisimman vähän verteksejä ja kolmioita, jotta pelin suorituskyky pysyy korkeana. Suurempi määrä johtaa suurempiin vaatimuksiin laitteistolta, mikä johtaa pienempään mahdolliseen käyttäjäkuntaan. Monimutkaisemman, yksityiskohtaisemman mallin käyttö kuluttaa enemmän akkua ja vaatii enemmän välimuistia. 3D-mallinnuksen suurimpia haasteita on tasapainottaa suorituskyky sekä visuaalinen laatu. [16.] Suuresta määrästä kolmioita muodostuvaa, yksityiskohtaisempaa mallia kutsutaan myös korkearesoluutioiseksi- tai korkeapolygoniseksi malliksi, joka englanniksi lyhyemmin tunnetaan nimellä high-poly model.

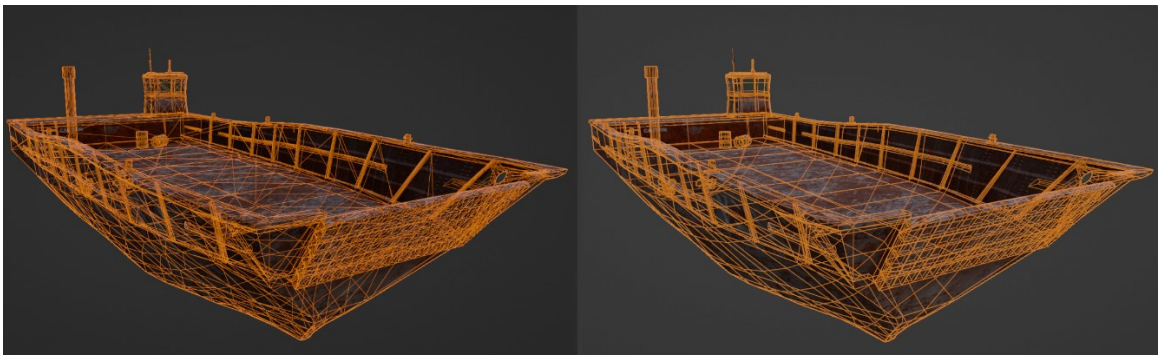
Mallien kolmiomäärän voi pitää matalana poistamalla ja yhdistämällä olemassa olevia reunoja mallista, kuten kuvassa 7, ja poistamalla ylimääräiseksi jääneitä tahkoja, kuten kuvassa 8. Tärkeää on pitää mallin ulkomuoto ja yleisilme samanlaisena. Pääasiassa tulee välttää poistamasta tahkoja ja reunoja, jotka vaikuttavat mallin ääriviivaan ja pääasialliseen muotoon. Yleensä malleista kannattaa poistaa tahkoja ensisijaisesti pinnoilta, jotka pelaaja näkee harvoin. [16.] Mallintaessa kannattaa välttää luomasta pitkiä kolmioita ja pysyä lähes tasareunaisissa kolmioissa. Näiden piirtäminen vie turhia resursseja GPU:lta. [15.]



Kuva 7. Vasemmalla alkuperäinen malli, oikealla sama malli, jota optimoitu vähentämällä kolmioiden määrää yhdistelemällä reunoja ja verteksejä.

Kuva 8. Esimerkki mallista, josta on poistettu mallintaessa tahattomasti luotuja tahkoja (poistetut tahkot merkitty punaisella oikealla). Nämä tahkot ovat muitten mallin osien peitossa, joten ne eivät näy lopullisessa mallissa. Mallista puuttuu myös pohja, sillä mallia ei ole tarkoitettu nähtäväksi alta.

Eräs toinen tapa optimoida mallia on käyttää level of detail -malleja, lyhennettynä LOD. Tässä tekniikassa otetaan olemassa oleva malli ja vähennetään sen kolmioiden määrää tekemällä pelkistetty kopio mallista, jota aiotaan käyttää vain, kun malli on niin kaukana, ettei joitain sen yksityiskohtia voi erottaa [16.]. LOD:eja voi olla useita, ja niiden nimeäminen aloitetaan 0:stä, jossa alin luku on yksityiskohtaisin malli. LOD:eja käytetään myös tiettyihin eri tarkoituksiin, esimerkiksi joissain pelimoottoreissa mallin varjoa varten voi olla oma LOD. Esimerkki LOD:sta on kuvassa 9.



Kuva 9. Vertailu mallin LOD 0:sta (vasemmalla) sekä versiosta, josta on vähennetty kolmioita, eli LOD 1:stä (oikealla). Mallin särmät näkyvät kuvassa oransseina viivoina havainnointia varten, niitä ei ole korostettu lopullisessa pelikäytössä. LOD 0 sisältää 7 242 ja LOD 1 sisältää 4 982 kolmiota. Oikeanpuoleinen malli on tarkoitettu nähtäväksi kauempaa, sillä siitä on poistettu pieniä yksityiskohtia, joita ei näe kaukaa. Tämä on tyypillinen LOD:n käyttötarkoitus.

LOD:ien lähtökohta on, että jos mallin kolmio vie vähintään pikselin ruudulta, niin se on optimaalinen. Tätä suurempi määrä kolmioita on turhaa, sillä tietyssä pisteessä ylimääräisten kolmioiden lisääminen ei tuo parannettua visuaalista ilmettä vaan heikentää suorituskykyä. Tästä syystä fotorealismiin pyrkiessä 3D-grafiikassa on suotavaa käyttää normaalikarttoja, jotka mainittiin kohdassa 2.3. Niiden avulla voidaan saavuttaa parempi kuvatarkkuus huomattavasti vähemmällä määrällä kolmioita alkuperäiseen, korkeapolygoniseen malliin verrattuna. Pienempi määrä kolmioita on kevyempi taakka grafiikkaprosessorille, joka vastaa ruudulle piirrettävien kuvien laskuista. Kevyempi taakka johtaa suurempaan nopeuteen kuvien piirrossa, eli FPS-luku nousee.

Mallin tahkot ja verteksit sisältävät dataa, joka kertoo mihin suuntaan kyseinen osa osoittaa. Tätä dataa kutsutaan normaaliksi ja se vaikuttaa miten mallin pinnat reagoivat valoon. Yleisesti ottaen pehmennettyä valaistusta käytettäessä tahkojen normaalit perustuvat ympäröivien verteksien normaalien keskiarvoon, ellei mallinnettaessa käytetä tekniikoita, joilla saadaan reunat näyttämään terävämmiltä. Tekniikan nimet ja käytänteet vaihtelevat ohjelman mukaan. Blender-mallinnusohjelmassa tätä kutsutaan mark sharp-tekniikaksi, jossa jokainen särmä, jonka halutaan näkyvän terävänä, merkitään erikseen. 3DS Max -ohjelmassa puolestaan valitaan ja ryhmitetään tahkot reunan terävyyden määrittämiseksi – näitä kutsutaan nimellä smoothing groups. Mallin suorituskykyä voi parantaa vähempi terävien särmien käyttö, sillä se vähentää verteksinormaalien määrää ja täten grafiikkaprosessorin laskennan taakkaa, samoin kuin vähempi UV-saumojen määrä. Tällöin tietokoneen tarvitsee lukea vähemmän dataa vertekseistä.

3.2 Tekstuurien optimointi

Tekstuurit ovat 3D-grafiikassa käytettyjä kuvatiedostoja, jotka ladataan laitteen välimuistiin ja piirretään mallien pinnoilla. On tärkeää, että näitä tiedostoja on mahdollisimman vähän ja että ne ovat mahdollisimman pienikokoisia ja että pikseleitä ei ole käytetty turhaan, jotta muistin käyttö pysyy alhaisena. Oikea tiedostomuoto auttaa vähentämään tiedostokokoa, kuten myös pakkausmuodon ja värisyvyyden valinta. [2, s. 147.]

Tekstuurit ovat kuvatiedostoja ja kuvatiedostoihin on olemassa sekä useita tiedostomuotoja sekä pakkausalgoritmeja. Jotkin pelit käyttävät pakkaamattomia- tai vähän pakkaavia kuvaformaatteja tekstuureissaan. Tämän takia kuvan laatu säilyy hyvänä, mutta tiedoston koko voi olla huomattavasti suurempi kuin muissa tiedostomuodoissa. Tekstuureille on useita eri tiedostoformaatteja ja pelimoottorissa voi valita tekstuureille erikseen pakkausalgoritmin. Näistä valitseminen riippuu käyttötarkoituksen mukaan. Tekstuurien kuvaresoluution on hyvä olla toisen potenssi, esimerkiksi 256x256 pikseliä eli 2^8, koska tällaisen resoluution omaavat tekstuuritiedostot vievät vähemmän muistia ja ne voidaan pakata paremmin [14].

Kun grafiikkaprosessori piirtää ruudulle kuvaa, se lukee käskyjä pelimoottorin renderöijästä. Pelimoottori lähettää käskyt siitä, mitä piirretään paketteina, joita kutsutaan piirtokutsuksi. Vähemmän piirtokutsuja tarkoittaa pienempää taakkaa grafiikkaprosessorille, tosin tähän pyrkiminen vaatii lisätyötä graafikolta. [16.] Tämä johtaa pelin suorituskyvyn paranemiseen, pienempään virrankulutukseen ja täten myös laitteen pidempään akunkestoon. Yksi tapa vähentää piirtokutsujen määrää on yhdistelemällä samaan kutsuun useita malleja, jotka käyttävät samaa tekstuuria ja materiaalia. Tätä kutsutaan ryhmittämiseksi. Tämän vuoksi on suorituskyvylle edullista käyttää samaa tekstuuria tai mallia useassa kohtaa peliskeneä. [15; 17.]

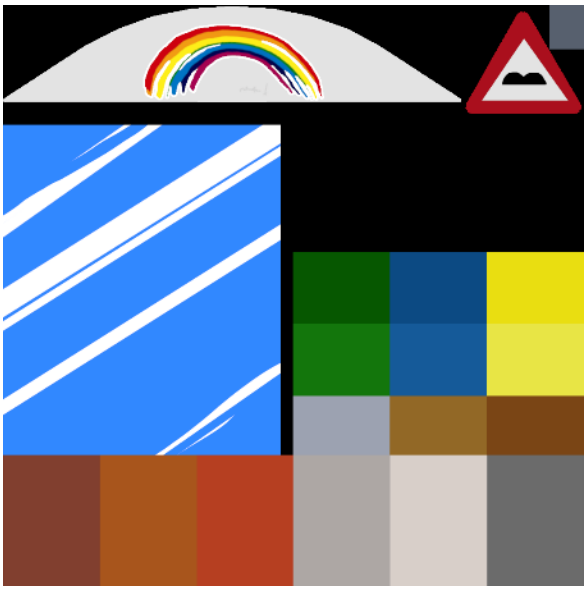
Yksi tapa optimoida vähentämällä tekstuurien määrää on käyttää tekstuuriatlasta. Tekstuuriatlas on kooste useista käytettävistä tekstuureista, jotka on yhdistetty siten, että yhtä atlasta voi käyttää useilla pinnoilla. Tekstuurien yhdistämisen tarkoitus on vähentää piirtokutsujen ryhmien määrää, mikä auttaa vähentämään laitteiston GPU:lle annettua taakkaa. [18, s. 1.]

Kuvassa 10 on esimerkki pelissä käytetystä tekstuuriatlaksesta. Kyseinen kuvatiedosto on käytössä useassa mallissa samaan aikaan. Tämä käytön jakaminen vähentää muistin tarvetta. Kysei-

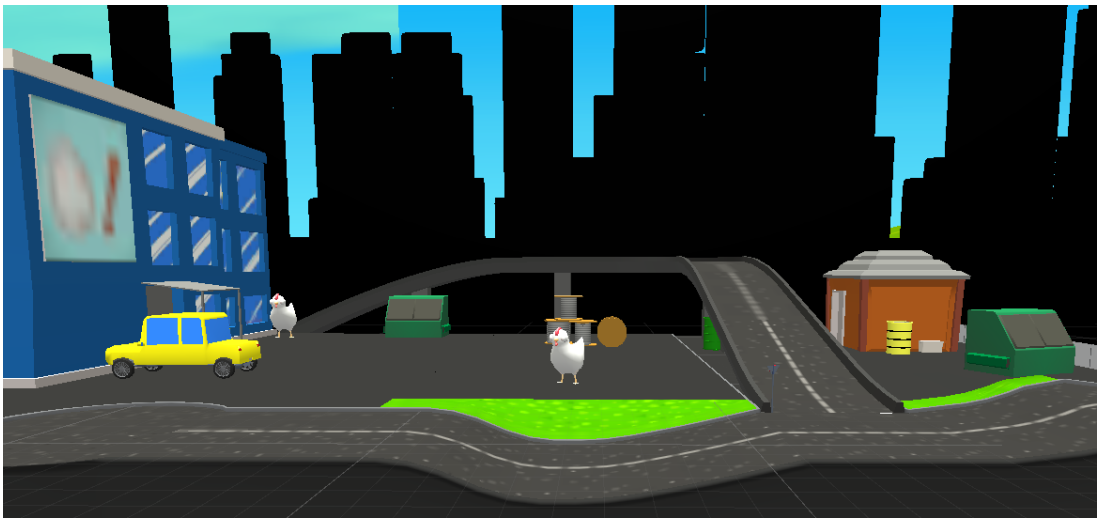
sen kuvatiedoston koko on tosin niin pieni, että pakkausalgoritmin käyttäminen hajottaisi ja sumentaisi värejä niin paljon, että lopullinen malli ei näyttäisi halutulta. Tämän vuoksi kuvaa ei pakata pelimoottorissa.

Kuva 10. Esimerkki pelin käyttämästä tekstuuriatlaksesta. Kuvan koko on 16x16 pikseliä ja vie 129 bittiä muistia. Pelin autot käyttävät kaikki tätä ja vastaavanlaisia tekstuureja pohjavärikarttana. Myös pelin taustalla olevat mallit käyttävät erilaisia tekstuuriatlaksia (Kuva 11; Kuva 12).

Samoin kuvassa 11 on tekstuuriatlas, joskin yksityiskohtaisempi ja täten suurempi resoluutioltaan. Tämä kuvatiedosto pakataan, jotta muistin käyttö vähenee. Pieni yksityiskohtien sumeneminen ei haittaa tässä tapauksessa suuremman resoluution ansiosta. Myös kuvan 11 tekstuuri on käytössä useissa 3D-malleissa kuvassa 12.



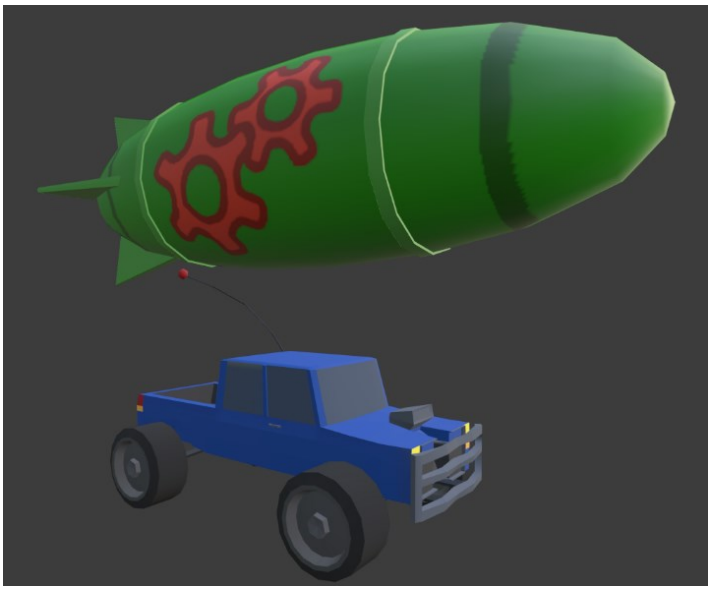
Kuva 11. Pelin taustalla käytetty tekstuuriatlas



Kuva 12. Esimerkki pelin sisäisestä taustasta, jossa on käytössä mm. kuvien 10 ja 11 tekstuuriatlakset. Kuvakaappaus Unity-pelimoottorista.

4 ”Eggspress delivery” -mobiilipelin kehittäminen ja optimointi

Kehitimme pienen ryhmän kesken mobiilipeliä. Pelin kehityksen alussa ideana oli luoda sivustapäin kuvattu ajopeli, jossa olisi kahdenlaisia kohtia pelikentässä: perinteistä autoilla ajoa sekä kohtia, joissa autot leijuvat ja pelaaja väistää esteitä liikkumalla ylös- tai alaspäin ruudulla. Autojen katoilla olisi ilmapallot, joiden ansiosta ne voisivat leijua, kuten kuvassa 13. Tällä idealla alettiin kehittämään mobiilipeliä Unity-pelimoottorilla. Jo hyvin aikaisessa vaiheessa kehitystä kuitenkin keksittiin, kuinka saadaan pelistä hauskempi: sijoittamalla avolava-auton takaosaan huomattavan kokoinen pallo ja vaihtamalla pelin tavoitteeksi kyseisen objektin kyydissä pitämisen kesken kilpa-ajon. Pelin keskeinen idea vaihtui tähän ja alettiin kehittämään videopeliä, jossa pelaajat eivät saa pudottaa kananmunaa kauko-ohjattavan autonsa kyydistä kilpaillessa kolmen tekoäly-pelaajan kanssa. Pelin nimeksi valikoitui Eggspress Delivery. Kuvassa 14 näkee tämän lopullisen peli-idean.



Kuva 13. Demonstraatio alkuperäisestä peli-ideasta.

Kuva 14. Kuvakaappaus lopullisesta peli-ideasta.

4.1 Grafiikan ja mekaniikkojen esittely

Pelin grafiikat ja tyylisuunta ovat yksinkertaisia. Tarkoituksena oli luoda peli, jolla on mahdollisimman hyvä suorituskyky vanhemmillakin laitteilla. Vanhin Android-versio, jota pelimme tukee, on vuonna 2013 julkaistu 4.4. PBR:n käyttö varaa välimuistia enemmän käyttöönsä kuin 3D-peli, joka ei käytä PBR:ää. Tästä syystä Eggspress Delivery ei käytä PBR-tekniikoita, vaan pelin 3D-malleissa käytetään tekstuurina ainoastaan perusvärikarttaa. Käytämme LOD:eja siten, että päävalikossa näkyvät autot ovat alkuperäisiä eli LOD 0. Pelin sisäiset mallit eli ne, jota pelaajat oikeasti liikuttavat, on puolestaan yksinkertaistettu poistamalla takaa tahkoja, joita pelaajan kuvakulmasta ei voi nähdä, kuten kuvassa 15. Samoin mallista on vähennetty kolmioita yksinkertaistamalla joitain piirteitä, kuten vähentämällä sivupeilien ja pohjan monimutkaisuutta. Auton pohja on harvoin näkyvillä eikä pitkää aikaa kerralla, joten sen geometriaa voi yksinkertaistaa ilman, että pelin ilme kärsii.

Kuva 15. Vasemmalla alkuperäinen malli (LOD 0) ja oikealla pelaamista varten optimoitu versio. Kun pelatessa näkyy vain auton oikea kylki, voi vasen puuttua.

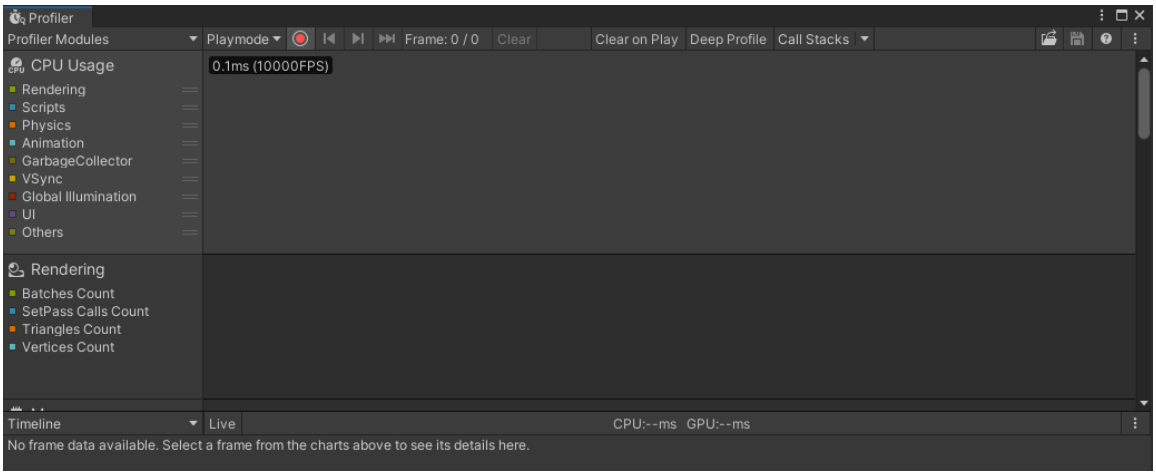
Pelin pääpelimekaniikkana toimii yksinkertainen sivusta kuvattu kilpa-ajo. Pääpelimekaniikan valmistuttua pelin kehitys on keskittynyt sisällön luomiseen. Olemme luoneet lisää autoja, värejä, uusia pelikenttiä sekä uusineet käyttöliittymän. Pelistä haluttiin mielenkiintoisempi lisäämällä päämekaniikan lisäksi toinen niin sanottu minipeli. Tämä yksinkertainen peli on käytännössä pelaajille tapa tienata pelin sisäistä valuuttaa, jota käyttää helpottamaan pääpeliä parantamalla autojen suorituskykyä. Samalla tavoite oli luoda tapa saada pelaajat palaamaan peliin usein keräämään nämä valuuttansa. Minipelissä ei ole muuta tehtävää kuin kerätä valuuttaa odottamalla ajastimen valmistuvan. Saaduilla pelinsisäisillä tuloilla voidaan parantaa minipelistä saatua valuutan määrää tai avata uusia pelikenttiä, autoja sekä muita lisäosia pääpeliin.

Koska olimme kokeneempia Unity-pelimoottorin käytössä, se valikoitui käyttöön Eggspress Delivery -pelin kehitykseen. Toinen syy oli, että Unreal Engine käyttää C++ -ohjelmointikieltä skriptien luomiseen, toisin kuin Unity, jossa käytössä on C# -kieli, josta tiimillämme oli enemmän kokemusta.

4.2 Testit

Pelimme ei ole tässä vaiheessa enää prototyyppi, vaan sisältää jo keskeiset suunnitellut elementit eli se on alfavaiheessa. Siksi pelimme testaaminen keskittyy teknisten vikojen löytämiseen enemmän kuin kohdeyleisön kanssa kokeiluun. Haluttiin testata, millaisen eron optimoitujen mallien käyttäminen luo peliin. Tähän käytettiin pelaamisen aikana tallennettua dataa.

Käytössä oli Unity-pelimoottorin sisäinen oma profiler-työkalu (Kuva 16). Profilerin on tarkoitus tallentaa ja näyttää dataa pelin tilastoista, kuten CPU:n, GPU:n, netin sekä välimuistin käytöstä. Nämä tilastot näkyvät käyrinä, joita voidaan tallentaa ja vertailla. Profilerilla ei valitettavasti voi mitata skenejen välisten latausten pituuksia ja niiden eroja. Profilerin tueksi ladattiin Unity-pelimoottoriin myös Profile Analyzer -niminen lisäosa.

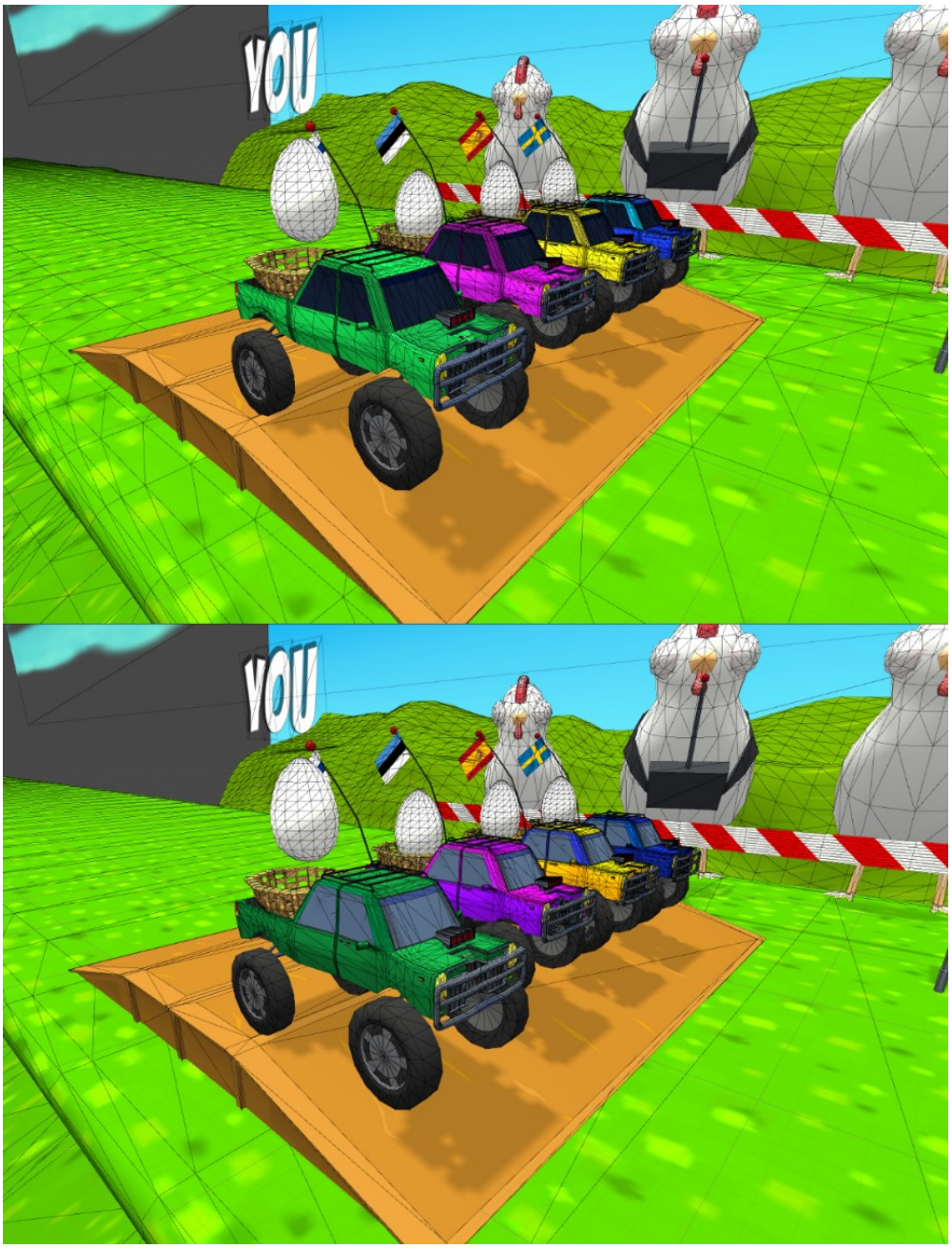


Kuva 16. Näkymä tyhjästä profiler-ikkunasta Unity-pelimoottorin editorin sisällä.

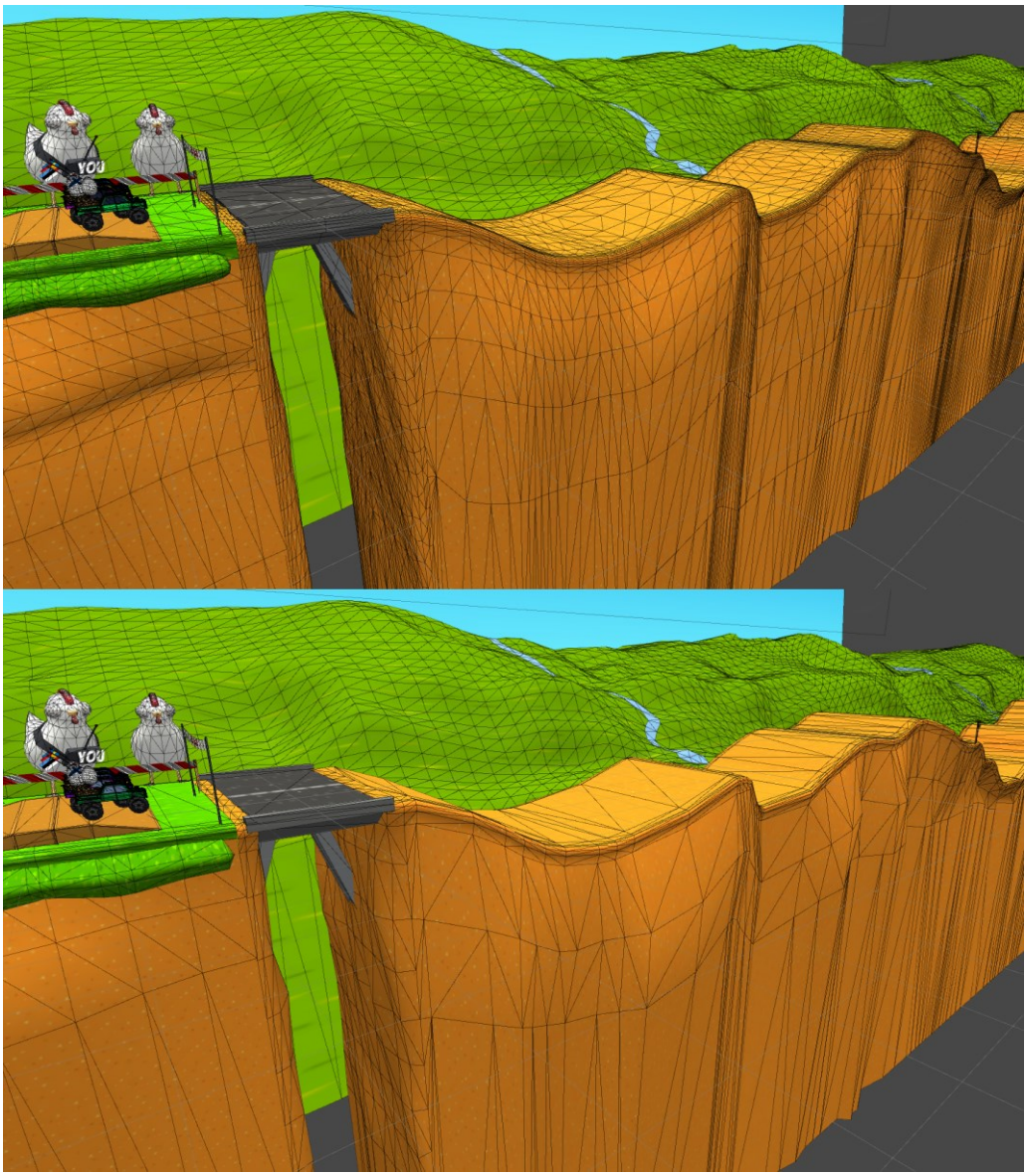
Koetta varten testattiin kahta skeneä Unity-pelimoottorin sisällä. Molempia skenejä testattiin 5 sekunnin ajan ja tulokset tallennettiin. Tallenteista etsittiin n. 10 millisekunnin kohta, jota vertailla. Näytteessä ei saanut olla yllättäviä ongelmia, kuten yksittäisiä hidastumisia pelin kuvataajuudessa tai tiedostojen latausnopeudessa. Näytteeseen haluttiin kaikki skenen mallit kerralla näkyviin, jotta materiaalien ja kolmioiden määrien ero olisi korkeimmillaan.

Yhdessä skenessä oli käytössä optimoituja malleja ja toista varten loin versiot malleista, joissa on käytetty huomattavasti enemmän kolmioita sekä yhdessä ylimääräisiä materiaaleja. Malleihin li-

sättiin kolmioita Blender-ohjelmassa aluksi leikkaamalla mallit keskeltä puoliksi ja sitten lisäämällä subdivision ja mirror -modifikaattorit. Ainoastaan autolle tätä ei tehty, vaan käytettiin vanhempaa versiota auton mallista, jossa kameralta piilossa olevan puolen tahkoja ei ollut poistettu eikä ylimääräisiä reunoja ja verteksejä yhdistelty. Autosta poistettiin käytöstä tekstuuriatlas ja kaikki tahkot värjättiin uusilla materiaaleilla, joissa jokainen väri oli oma materiaalinsa. Mallien yksityiskohtaiset erot voi nähdä kuvissa 17 sekä 18.



Kuva 17. Testissä käytettyjen huonosti optimoitujen (ylempi puoli) ja optimoitujen (alempi puoli) autojen mallien vertaus. Mallien kolmiomäärää on suurennettu, ilman että niiden visuaalinen laatu paranisi huomattavasti, joka on merkki huonosta optimoinnista. Autoissa on käytetty sama määrä värejä, mutta huonosti optimoidut mallit käyttävät jokaista eri väriä varten omaa materiaalia, esimerkiksi vihreä ja tumman vihreä ovat omat materiaalinsa. Tämä ei ole suorituskyvyn puolesta kannattavaa, mutta voi olla hieman nopeampaa graafikolle.



Kuva 18. Testissä käytettyjen huonosti optimoitujen (yllä) ja optimoitujen (alla) ympäristön mallien vertaus. Taustan malleja ei muutettu testissä, ainoastaan pelaamiseen liittyviä malleja eli autoja sekä kentän palasia muokattiin. Ympäristöt käyttävät keskenään samoja tekstuureita ja materiaaleja.

Testissä ei ole lisätty PBR-materiaaleja ja testattu niiden vaikutusta suorituskykyyn, koska kyseessä on mobiilipeli ja peliä ei ole suunniteltu PBR mielessä. Tämä voi vaikuttaa testin merkittävyyteen ja lopputulosten eron suuruuteen.

Testi suoritettiin tietokoneella, Unity-pelimoottorin editor-ohjelmassa. Tietokoneen prosessorina oli AMD Ryzen 1700 ylikellotettuna kellotaajuudelle 3,79Ghz. Näytönohjaimena toimi 2 kappaletta Nvidia GTX 960:ä SLI-konfiguraatiossa. Välimuistia oli 32 gigatavua DDR4:ää. Mallien väliset erot lukevat taulukossa 1.

	Kolmiomäärä (Optimoitu malli)	Kolmiomäärä (Huonosti optimoitu malli)	Materiaalien määrä (Optimoitu malli)	Materiaalien määrä (Huonosti optimoitu malli)
Auto	6168	7768	1	9
Karttapala 1	770	6160	2	2
Karttapala 1:n reuna	880	3536	2	2
Karttapala 2	1353	8244	3	3
Karttapala 3	946	6192	1	1

Taulukko 1. Testiin sisältyvien 3D-mallien erot.

4.3 Tuloksia testeistä

Testin tuloksena voidaan taulukosta 2 nähdä, kuinka optimoimalla mallit saatiin kevyemmin toimiva peliskene. Muistin käyttö, tekstuurien määrä sekä ensimmäisen ruudun kolmiomäärä laskivat huomattavasti. Vaikka materiaalien suuri määrä epäoptimoidussa skenessä ei vaikuttanut materiaalien viemän muistin määrään, uusiokäyttämällä materiaaleja ryhmitettyjen piirtokutsujen määrä laski huomattavasti. Tämä voi johtaa suorituskyvyn paranemiseen varsinkin vähemmän tehokkailla laitteilla.

Tulos ei ole erityisen tarkka vertailu optimoinnin eduista. Skenessä oli eroa vain 7 mallin verran, joten malleja olisi voinut lisätä ja muokata entisestään. Eggspress Delivery -pelin tapauksessa tämä ei kuitenkaan ole merkittävästi tulosten haitaksi, sillä pelissä näkyy harvoin tätä useampaa mallia kerralla. Tulokseen vaikuttaa monta muuttujaa. Taustalla pyöri muita prosesseja pelin lisäksi, varsinkin Unity-editointiohjelman pyörittäminen vei testaavalta laitteelta tehoja. Jotkin tulokset ovat myös lähes merkityksettömiä, esimerkiksi vaikka materiaalien viemä muisti laski 17,24 %, niin tämä johti vain muutaman kymmenen kilotavun laskuun – käytännössä merkityksetön summa nykylaitteilla, joista vanhemmillakin on vähintään satoja megatavuja välimuistia. Tulokseen vaikuttaa erityisesti se, että se toteutettiin kohtuullisen tehokkaalla pöytätietokoneella eikä mobiililaitteella, jolla tulokset voisivat olla hyvin erilaisia. Ero voi olla laitteesta riippuen huomattava.

	Hyvin optimoitu skene	Huonosti optimoitu skene	Ero prosentteina
Muistin käyttö (kokonaan)	0,70 Gt	0,81 Gt	-13,58 %
Muistin käyttö (pelkkä grafiikka)	45,3 Mt	46,0 Mt	-01,52 %
Tekstuurit, määrä	1046 kpl	1231 kpl	-15,02 %
Tekstuurit, muistin käyttö	178,4 Mt	199,4 Mt	-10,53 %

3D-mallit, määrä	76 kpl	78 kpl	-02,56 %
3D-mallit, muistin käyttö	12,5 Mt	13,9 Mt	-10,07 %
Materiaalit, määrä	129 kpl	144 kpl	-10,41 %
Materiaalit, muistin käyttö	315,8 Kt	381,6 Kt	-17,24 %
Ensimmäisen ruudun kolmiomäärä	101,7 tuhatta	151,4 tuhatta	-32,82 %
Ensimmäisen ruudun verteksien määrä	96,8 tuhatta	127,2 tuhatta	-23,89 %
Ensimmäisen ruudun piirtokutsujen ryhmien määrä	122	186	-34,40 %
Piirtokutsujen ryhmien määrä korkeimmillaan	142	218	-29,81 %

Taulukko 2. Tuloksia testeistä optimoidun ja ei-optimoidun skenen välillä. Taulukko jatkuu edelliseltä sivulta

Olisi parasta toteuttaa vastaava koe useilla mobiililaitteilla. Varsin hyvä olisi nähdä pelin suorituskyky vanhemmalla eli yli viisi vuotta vanhalla laitteella. Lisäksi testi toteutettiin 4K-näytöntarkkuudella (3840 x 2160 pikseliä), joka on resoluutio, jota useimmat mobiililaitteet eivät tue. Samaa testiä yritettiin myös Xiaomi Redmi Note 9 Android-puhelimella, mutta kyseisen puhelimen tehokkuus ei riittänyt ajamaan pelin kehittäjäversiota, jota Unity profiler lukisi samalla. Puhelimella peli lakkasi toimimasta heti, kun se avattiin. Täten testaus jäi vain tietokoneella toteutetuksi.

Huolimatta siitä, että testi ei ollut tarkin mahdollinen, voi tuloksista silti havaita, kuinka 3D-mallien optimointi on tärkeä tapa saada pelin suorituskykyä parannettua. 3D-graafikoiden on siis hyvin tärkeää osata optimoida mallinsa pelivalmiiseen kuntoon.

5 Pohdinta

Lopullinen testi saavutti halutut tulokset, joista voi nähdä eron optimoinnin hyödyistä: vähentämällä materiaalien sekä kolmioiden määrää onnistuttiin vähentämään mm. käytettyä muistia sekä piirtokutsujen määrää ja täten GPU:n käyttöä, jolloin peli tulee toimimaan paremmin myös suorituskyvyltään heikommilla mobiililaitteilla. Testi olisi voinut olla laajempi ja sisältää suuremman otannan testattavia laitteita ja malleja. Yritys Unityn profiler-työkalun käytöstä Android-puhelimella ei kuitenkaan onnistunut, vaan käytetyn puhelimen resurssit eivät riittäneet ajamaan pelin kehittäjäversiota Profiler-työkalun kanssa. Keskeisenä tuloksena voi kuitenkin onnistuneesti nähdä, että optimoimalla malleja ja tekstuureja voidaan hyötyä paremmasta suorituskyvystä.

Tämä työ onnistui tavoitteessaan kasata lyhyt katsaus optimoinnin tavoista ja hyödyistä. Tietoa optimoinnista, varsinkin 3D-mallinnusohjelman sisäisistä tavoista ja käytänteistä, voisi tosin olla vielä lisää. Työ pysyy mobiilipelien aihepiirissä, jonka vuoksi siinä ei käsitelty mm. high-poly mallinnusta ja tekstuurien leipomista, sekä muita asioita, joita graafikon on hyvä osata.

Tämä työ alkoi edeltävänä syksynä seminaarityönä, josta kirjoittaminen jatkui opinnäytetyöksi. Tapausesimerkin peli on ollut kehityksessä yli vuoden pidempään, mutta on ollut työpaikan vuoksi viimeistelemättä ja julkaisematta pidemmän aikaa. Pelin kehittäminen on jatkunut silloin tällöin, ja tämän opinnäytetyön kirjoittaminen on toiminut lisämotivaationa pelin viimeistelylle.

Työtä tehdessäni opin optimoinnista, normaaleista, 3D-grafiikasta, testaamisesta, termeistä sekä datan käsittelystä. Opinnäytetyön kirjoittamisessa ei ole ollut monia vastoinkäymisiä, suurimpana oli se, että pelin testaaminen ei onnistunut puhelimellani. Kirjoittaminen oli opettavainen prosessi ja olen työhön tyytyväinen. Vaikken pidä sitä sisällöllisesti alaa mullistavana, työ voi silti olla hyödyllinen lukijalle. Toivon tämän työn auttavan aloittelevia 3D-graafikkoja pääsemään alkuun ja välttämään joitain virheitä, joita itse tein aloittelijana. Optimointi on tärkeä tapa edistää peliä ja tapojen opiskelu kartuttaa taitoa graafikkona, josta on varmasti hyötyä graafikon uralla, sillä optimoitua tuotetta arvostetaan aina, ei huonoa suorituskykyä.

6 Yhteenveto

Optimointiin on monta tapaa ja graafikkona voi toimia lähes vapaalla kädellä. Kenties tärkeimpänä on muistaa pitää kolmiomäärä alhaisena, tekstuurit pieninä ja materiaalit vähäisinä joustamatta lopullisesta laadusta. Tekniikoita on monia, ja niistä kannattaa käyttää tilanteeseen sopivimpia mahdollisimman monta. Yleisesti voidaan suositella yhdistellä vierekkäisiä verteksejä ja särmiä, kunhan ääriviivat säilyvät pääosin samana. Mallin topologia on hyvä pitää siistinä, pääasiassa nelikulmaisten tahkojen verkkona.

Mallissa on suotavaa käyttää mahdollisimman vähän tekstuureja. Niitä voi yhdistellä tekstuuriatlaksiksi tai muuten keskenään, pitäen tekselitiheyttä silmällä. Samoin materiaalien määrä on pidettävänä matalana, jotta peli lähettää mahdollisimman vähän piirtokutsuja GPU:lle – mieluiten yksi materiaali per malli, jos mahdollista. Mallit voivat myös käyttää samoja materiaaleja keskenään. Malleissa on hyvä käyttää LOD:eja.

Peliskeneä on suositeltavaa testata. Kannattaa pitää silmällä varsinkin piirtokutsujen määrää, ja selvittää mitkä mallit niitä aiheuttavat ja miten vähentää piirtokutsuja. Eri pelimoottoreissa on vaihtelevat työkalut näitä varten.

3D-mallien optimointi on hyvin tärkeä osa 3D-graafikon työtä. On hyvä muistaa mobiililaitteille kehitettäessä, että laitteet vaihtelevat suorituskyvykkyydeltään, mutta oletuksena on pitää peli mahdollisimman kevyenä, jotta heikommankin pään laitteet voivat suoriutua pelin kelvollisesta ajamisesta. Ilman optimointia moni peli jäisi käyttäjiltä lataamatta huonon suorituskyvyn vuoksi tai ei toimisi ollenkaan. Kun graafikko osaa optimoida mallinsa, koko lopputiimi hyötyy - kaikki technical artistista ohjelmoijaan kiittävät.

Lähteet

1. David Franson, Eric Thomas. Game character design complete. Thomson Course Technology; 2007. [Viitattu 7.4.2022] Saatavilla: https://theswissbay.ch/pdf/Gentoomen%20Library/Game%20Development/Designing/Game%20Character%20Design%20Complete.pdf

2. Avisekhar Roy. Android Game Developer's Handbook. Packt Publishing; 2016. [Viitattu 23.11.2021]

3. Learn OpenGL. Theory. [Viitattu 12.10.2021] Saatavilla: https://learnopengl.com/PBR/Theory

4. Joe Wilson. Physically-Based Rendering, And You Can Too! [Viitattu 13.10.2021] Saatavilla: https://marmoset.co/posts/physically-based-rendering-and-you-can-too/

5. Adobe. The PBR Guide - Part 2. [Viitattu 13.10.2021] Saatavilla: https://substance3d.adobe.com/tutorials/courses/the-pbr-guide-part-2

6. Jason Gregory. Game Engine Architecture, Third edition. 2018. [Viitattu 19.10.2021] Saatavilla: https://books.google.fi/books?id=EwlpDwAAQBAJ&lpg=PT17&ots=Ebr69p8U98&dq=game%20engine&lr&hl=fi&pg=PT17#v=onepage&q&f=false

7. Wayne Maxell. How Many Polygons Should a Game Model Have. CG Obsession; 2019. [Viitattu 17.4.2022] Saatavilla https://cgobsession.com/how-many-polygons-should-a-game-model-have/

8. Wayne Maxell. What Game Texture Resolutions Should You Use. CG Obsession; 2019. [Viitattu 18.4.2022] Saatavilla: https://cgobsession.com/what-game-texture-resolutions-should-you-use/

9. Thomas Hamilton. Game Testing: Types & How to Test Mobile/Desktop Apps. 2021. [Viitattu 25.11.2021] Saatavilla: https://www.guru99.com/game-testing-mobile-desktop-apps.html

10. Chales P. Schultz, Robert D. Bryant. Game Testing: All in One, Third Edition. Mercury Learning and Information; 2017. [Viitattu 16.4.2022] Saatavilla: https://books.google.fi/books?hl=fi&lr=&id=OD6TDgAAQBAJ&oi=fnd&pg=PT18&dq=video+game+testing&ots=bLqu4rQKmK&sig=rfTu3HybZVLicOptjQBUh3ar7go&redir_esc=y#v=onepage&q&f=false

11. Sean Whaley. What is Frame Rate and Why is it Important to PC Gaming? 2018. [Viitattu 16.10.2021]. Saatavilla: https://www.hp.com/us-en/shop/tech-takes/what-is-frame-rate

12. Tim Schiesser. How Many FPS Do You Need? TechSpot; 2018. [Viitattu 15.4.2022] Saatavilla: https://www.techspot.com/article/1668-how-many-fps-do-you-need/

13. Appfigures. Are Smaller Mobile Games More Successful? 2016. [Viitattu 22.11.2021] Saatavilla: https://appfigures.com/resources/insights/are-smaller-mobile-games-more-successful

14. Epic Games. Performance Guidelines for Mobile Devices. [Viitattu 18.4.2022]. Saatavilla: https://docs.unrealengine.com/5.0/en-US/performance-guidelines-for-mobile-devices-in-unreal-engine/

15. Keith O'Conor. GPU Performance for Game Artists. 2017. [Viitattu 14.10.2021] Saatavilla: http://fragmentbuffer.com/gpu-performance-for-game-artists/

16. Arm Developer. Triangle and polygon usage. [Viitattu 22.11.2021] Saatavilla: https://developer.arm.com/documentation/102448/0100/Triangle-and-polygon-usage

17. Unity Documentation. Draw call batching. [Viitattu 19.10.2021] Saatavilla: https://docs.unity3d.com/Manual/DrawCallBatching.html

18. NVIDIA. SDK White paper: Improve Batching Using Texture Atlases. 2006. [Viitattu 23.11.2021] Saatavilla: http://download.nvidia.com/developer/NVTextureSuite/Atlas_Tools/Texture_Atlas_Whitepaper.pdf

Kansikuva: Markku Hietamäki