Tampereen ammattikorkeakoulu

# Unity-projektin siirtäminen uuteen renderöintiputkeen

Jari-Pekka Salonen

OPINNÄYTETYÖ
Joulukuu 2020

Tietojenkäsittelyn tutkinto-ohjelma
Pelituotanto

**TIIVISTELMÄ**

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn tutkinto-ohjelma
Pelituotanto

SALONEN, JARI-PEKKA:
Unity-projektin siirtäminen uuteen renderöintiputkeen

Opinnäytetyö 37 sivua
Joulukuu 2020

---

Tässä opinnäytetyössä perehdyttiin 3D-grafiikan renderöintiin, sen toteutustapoihin Unity-pelimoottorissa ja toteutettiin tekijän kehitystiimin aiemmin kehittämään ja julkaisemaan Dante's Infernya -peliin uudet visuaaliset efektit. Työn tavoite oli vertailla Unityn renderöintiputkia eli graphics rendering pipelineja ja selvittää, miten pelin visuaaliset efektit voisi toteuttaa niissä.
Opinnäytetyön tarkoituksena tutkittiin renderöintiputkien ominaisuuksia ja valittiin pelille sopivin putki. Lisäksi suunniteltiin ja kehitettiin peliin uudelleen vanhat varjostimet sekä uusi varjostin valittua putkea käyttäen.

Renderöintiputkien vertailussa keskityttiin kehitystiimille sopivimpiin vaihtoehtoihin, jotka olivat High Definition Render Pipeline ja Universal Render Pipeline. Putkia vertailtiin Unityn dokumentaation perusteella. Valituksi tuli Universal Render Pipeline, koska se soveltui paremmin Dante's Infernyan teknisiin vaatimuksiin. Pelin varjostimet ja uudistukset suunniteltiin valittuun putkeen etsimällä internetistä esimerkkejä. Ne kaikki toteutettiin suunnitellusti muutamaa pientä poikkeusta lukuun ottamatta.

Työn lopussa valittu renderöintiputki todettiin pääpiirteiltään oikeaksi vaihtoehdoksi. Huomattiin kuitenkin, että tämä putki voi rajoittaa vaihtoehtoja efektien toteuttamisessa. Tästä syystä kehittäjien täytyy olla tietoisia projektinsa tarpeista ja valita renderöintiputki oikein. Varjostimista onnistuttiin tekemään paremmat kuin entiset, mutta kaikkia niiden vikoja ei onnistuttu korjaamaan. Varjostimia voi vielä jatkokehittää eteenpäin. Esimerkiksi esineiden reunustusefektiin voisi lisätä tuen läpinäkyville esineille.

---

Asiasanat: Unity, renderöinti, varjostin, URP, peli

## ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Game Development

SALONEN, JARI-PEKKA:
Moving a Unity Project into a New Rendering Pipeline

Bachelor's thesis 37 pages
December 2020

---

The purpose of this thesis was to compare the different graphics rendering pipelines available in the Unity game engine, and to determine how they could be used to develop visual effects for the author's development team's game, Dante's Infernya. In order to further develop the game, it was important to choose a pipeline and to develop new shaders using it.

The pipeline feature comparison focused on the High Definition Render Pipeline and the Universal Render Pipeline. The research was based on Unity's online documentation. Improved shaders were researched by finding examples of similar effects online.

The Universal Render Pipeline was found to be the correct pipeline for the game's technical requirements. The new shaders were found to be more performant than the old ones, but the chosen pipeline imposed some limitations on their design.

The results suggest that for Dante's Infernya, the Universal Render Pipeline is the correct pipeline in Unity. The work required to implement the game's design after choosing the pipeline implies that, in general, developers should know their game's needs so they can choose a suitable pipeline for their project. For further study, it would be interesting to approach the subject from an angle that uses the High Definition Render Pipeline instead.

---

Key words: Unity, rendering, shader, URP, game

**SISÄLLYS**

## LYHENTEET JA TERMIT

| | |
|---|---|
| HDR | Kuvan esitysmuoto, jossa on tavallisia näyttöjä laajempi kirkkaus |
| Materiaali | Hallitsee 3D-mallin ulkonäköä säilytetyillä resursseilla, kuten tekstuureilla |
| Node | Piste kaaviossa, joka muihin linkitettynä kuvastaa niiden välisiä suhteita |
| Primitiivi | 3D-mallin geometriaa ilmaiseva piste, viiva tai polygoni |
| Verteksi | 3D-tilassa sijaitseva piste, johon voi liittää tietoa, kuten värin |
| VR | Virtuaalitodellisuus |
| XR | Virtuaalitodellisuuden ja yhdistetyn todellisuuden vapaamuotoinen yhdistelmä |

# 1 JOHDANTO

Dante’s Infernya on 3D-tasohyppelypeli, jossa pelaajan ohjaama kissahahmo aiheuttaa kaaosta pudottamalla tavaroita lattialle samalla kun tekoälyn ohjaama ihmishahmo jahtaa sitä. Pelin kehitti ja netissä ilmaiseksi julkaisi nelihenkinen opiskelijatiimi. Pelin lämpimän vastaanoton seurauksena tiimi haluaa tutkia sen mahdollista jatkokehitystä isommaksi kokonaisuudeksi. Peli tehtiin Unity-pelimoottorilla ja nähdään aiheelliseksi kokeilla siirtää projekti pelimoottorin uudempaan versioon. Uusien työkalujen pitäisi helpottaa pienen tiimin työtaakkaa erityisesti pelin optimoinnissa, jotta se toimisi myös vanhemmilla laitteilla.

Yksi ratkaiseva valinta uuden peliprojektin luomisessa Unityllä on projektin renderöintiputki. Unityyn tullut Scriptable Render Pipeline -järjestelmä (SRP) antaa kehittäjien luoda oman renderöintiputken tai valita sellaisen annetuista mallikappaleista. Putken valinta vaikuttaa sitä seuraavaan kehitysprosessiin valtavasti. Esimerkiksi artistien työtehtävät voivat muuttua sen mukaan, mitä taideresursseja valittu putki tukee. Tämän takia valinta kannattaa tehdä hyvän tiedon pohjalta.

Opinnäytteen tavoitteena on vertailla Unityn uusia renderöintiputkia ja selvittää miten Dante’s Infernyan grafiikat ja efektit voisi toteuttaa valitussa uudessa ympäristössä. Tarkoituksena on tutkia renderöintiputkien ominaisuuksia ja valita niistä projektiin paras. On myös tarkoitus suunnitella ja sitten kehittää pelin varjostinefektit uusiksi valitun putken puitteissa. Lisäksi suunnitellaan ja toteutetaan graafisia uudistuksia alkuperäisen pelin parantamiseksi.

# 2 TAUSTATIEDOT

## 2.1 Reaaliajan renderöinti

Tietokonegrafiikassa kuvan tuottamista näytölle mallin (model) tai malliasetelman (scene) pohjalta kutsutaan renderöinniksi (Butterfield, Ekembe Ngondi & Kerr 2016). Videopeleissä käytetään reaaliajan renderöintiä. Akenine-Möller ym. (2018) määrittelevät reaaliajan renderöinnin jatkuvaksi kuvien näyttämiseksi tietokonenäytöllä nopeaan tahtiin. Pelaaja reagoi kuvaan ja antaa uuden syötteen, johon peli vastaa piirtämällä uuden kuvan. Tämä jatkuva kierto perinteisesti tapahtuu erittäin nopeasti, jotta käyttäjä ei huomaisi sitä. (Akenine-Möller ym. 2018, 1.)

Pelaajan peliin uppoutumisen takaamiseksi pelien täytyy käydä korkealla kuvataajuudella, jota mitataan ruutuina sekunnissa (frames per second tai FPS). Alhaisia nopeuksia, kuten elokuvissa yleisesti käytettyä 24 FPS, ei voi soveltaa interaktiivisiin medioihin. Tässä yhteydessä kuvien vaihtuminen on silmin nähtävissä, ja se koetaan häiritseväksi. Korkeammat renderöintinopeudet parantavat interaktiivisuuden tunnetta. Pelien yleiset kuvataajuudet ovat 30, 60 ja 72 ruutua sekunnissa. Alustasta riippuen vaatimukset muuttuvat, kuten esim. VR-pelien täytyy saavuttaa yleensä vähintään 90 FPS viiveen minimoimiseksi. (Akenine-Möller ym. 2018, 1.)

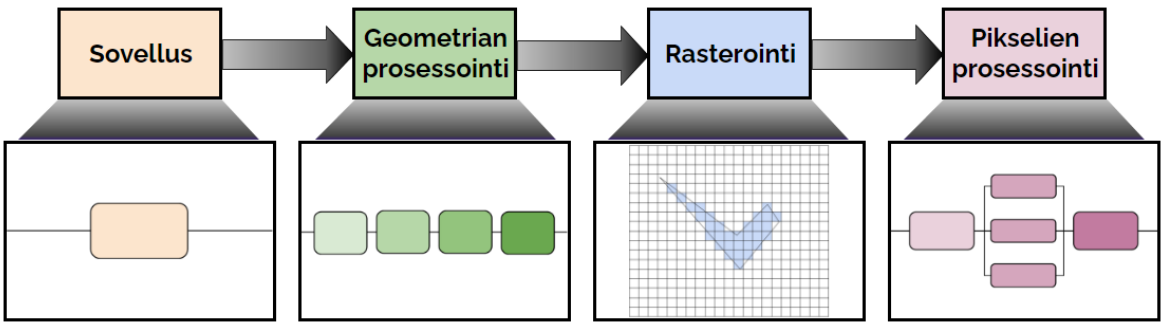
Renderöitävän kuvan käyttötarkoituksesta tai taustalla käytetystä moottorista riippumatta se rakennetaan tietokoneessa vaiheittain. Reaaliajan renderöinnissä on olennaista tietty johdonmukaisuus, jota kaikki eri renderöintiprosessin toteutukset seuraavat.

## 2.2 Renderöintiputki

Renderöintiprosessin käsite kattaa kaikki laskennalliset operaatiot, jotka muuttavat datan kuvaksi näytölle. Laitteistotasolla tietokoneessa prosessori ja näytönohjain ovat vastuussa näiden suorittamisesta. Prosessori tyypillisesti kerää datan, ja näytönohjain käsittelee sen kuvaksi. Prosessin vaiheet käydään läpi järjestelmällisesti, minkä takia Akenine-Möller ym. vertaavat sitä tehdastuotannon liukuhihnaan ja pikaruokalan keittiöön (Akenine-Möller ym. 2018, 12).

Renderöintiprosessin rakenteesta puhuttaessa käytetään termiä renderöintiputki (graphics rendering pipeline). Renderöintiputkelle on omanlainen toteutuksensa jokaisessa ohjelmaympäristössä, mutta prosessiin kuuluu tiettyjä universaaleja vaiheita, joiden takia korkealla tasolla putket muistuttavat toisiaan.

Akenine-Möller ym. (2018) esittämän karkean mallin mukaan renderöintiputken voi jakaa neljään pääosaan — sovellus, geometria, rasterointi ja pikselien prosessointi (kuvio 1). Tyypillisesti jokainen mallin vaihe itsessään sisältää useita alivaiheita tai alitehtäviä. Vaiheella on mallin mukaisesti oma vastuualue, mutta ei määritystä sen suorittamiselle. (Akenine-Möller ym. 2018, 12—13.)




KUVIO 1. Akenine-Möllerin ym. esittämä renderöintiputken rakenteen kuvaus korkealla tasolla (Akenine-Möller ym. 2018, 12)

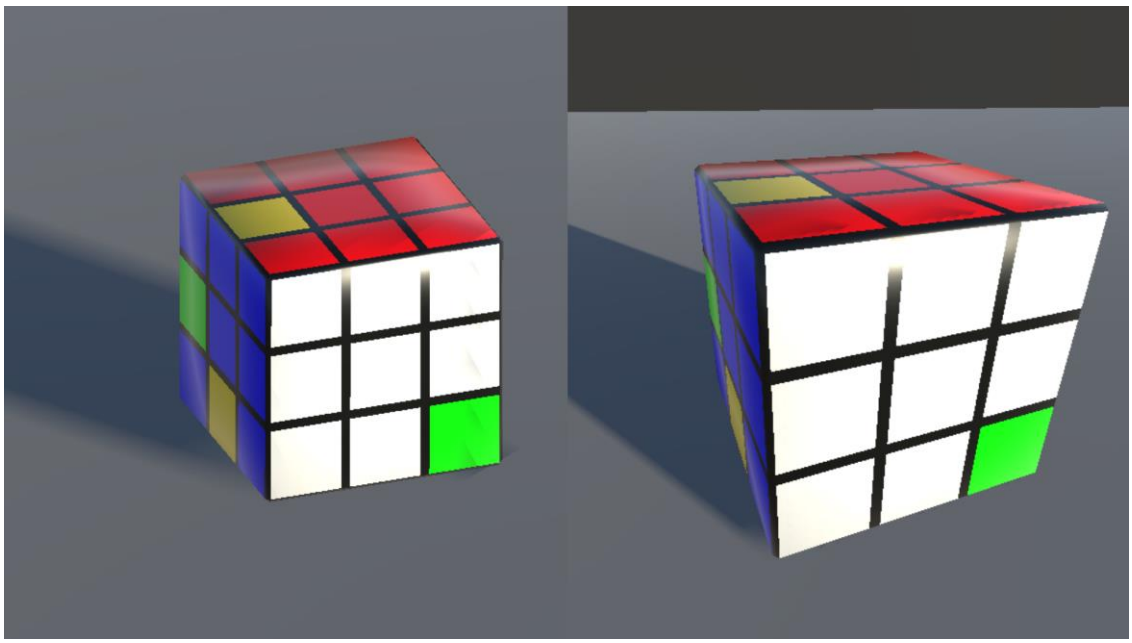
Sovellusvaihe tapahtuu prosessorissa, ja sen päätehtävä on koota myöhempien vaiheiden käyttämä data. Vaiheen lopussa sitä seuraavalle geometrian prosessoinnin vaiheelle lähetetään renderöitävät primitiivit. Tätä ennen kehittäjällä on täysi vapaus vaikuttaa vaiheen toteutukseen. Hän voi esimerkiksi sisällyttää renderöitäviä kolmioita vähentävän asetuksen, joka keventää

seuraavien vaiheiden kuormaa. Lisäksi sovellusvaiheessa kerätään tiedot tapahtumista, kuten pelaajan syötteet ja pelimaailman objektien yhteentörmäykset, jotka voivat vaikuttaa jatkotehtäviin. (Akenine-Möller ym. 2018, 13—14.)

Geometrian prosessoinnissa määritellään tehtävät, joissa käsitellään mallin verteksejä. Tätä ei voida suorittaa ilman sovellusvaiheen primitiivejä. Vaihe koostuu seuraavista askelista: verteksien varjostus, projisointi, leikkaus ja näytönkartoitus. Tästä vaiheesta on vastuussa näytönohjain. (Akenine-Möller ym. 2018, 14.)

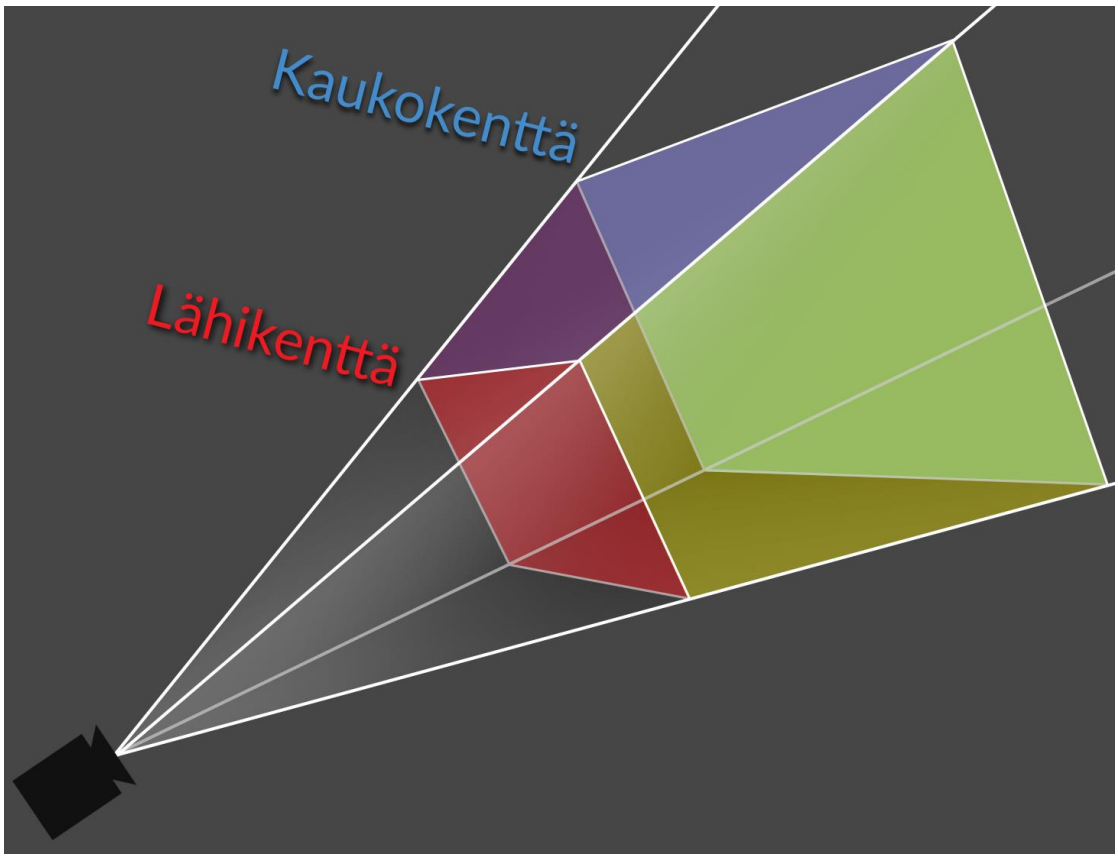
Verteksien varjostuksella viitataan mallin muodostavien koordinaattipisteiden asetteluun ja niiden värityksen muuttamiseen valon perusteella. Termi on vanhentunut, sillä nykyaikainen määritelmä kattaa kaikki verteksien käsittelyyn liittyvät valmistelut. Yksi mahdollinen käyttötarkoitus on animaatio — mallin verteksien sijaintia muuttamalla voidaan simuloida liikettä. (Akenine-Möller ym. 2018, 15.)

Projisoinnissa määritetään kameran näkemä osa pelimaailmasta. Kolmiulotteinen maailma muutetaan kaksiulotteiseen näytön tilaan. Tähän on useita eri tapoja, mutta yleisimmät ovat ortografinen projektio ja perspektiiviprojektio (kuva 1). Ortografinen projektio asettaa näkymän suoraan kameraa vasten, luoden litteän vaikutelman. Perspektiiviprojektio vuorostaan muuttaa kamerasta kauempana olevat objektit pienemmiksi simuloiden ihmissilmää. (Akenine-Möller ym. 2018, 16—17.)



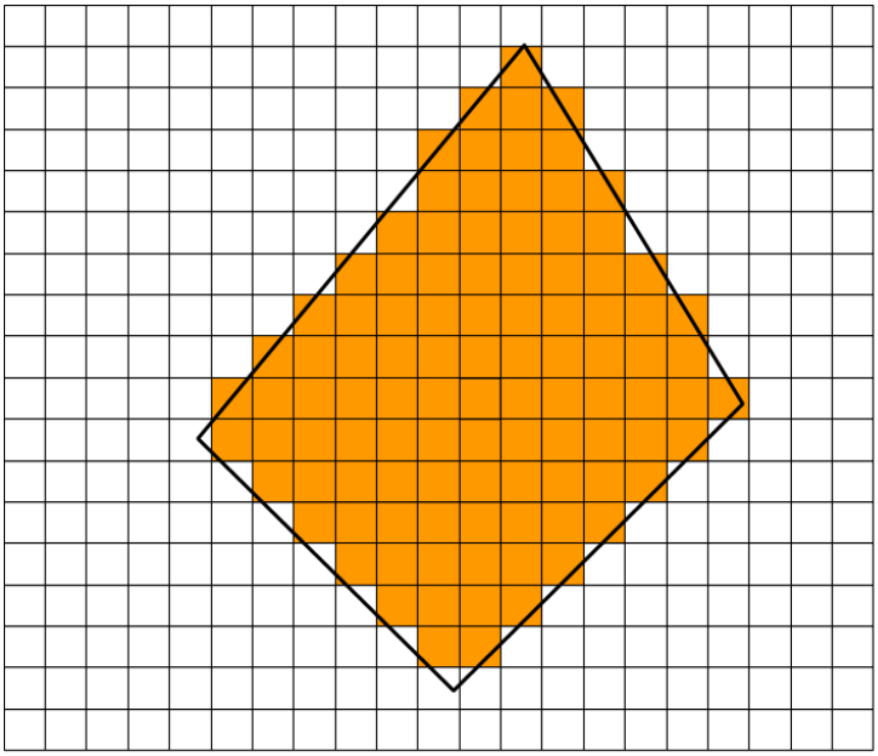
KUVA 1. Ortografinen projisointi (vasen) ja perspektiiviprojisointi (oikea)

Leikkauksessa jätetään pois mallien osat, jotka eivät sopineet projektioon. Täysin projektion ulkopuolelle jääneet mallit on jo jätetty pois, joten leikkauksen ei tarvitse käsitellä niitä. Samoin projektion sisälle kokonaan sopivat mallit jätetään huomioimatta ja annetaan sellaisenaan seuraavalle vaiheelle. Säilytettävä alue on kolmiulotteinen tila, joka määritetään kameran neljän sivun sekä kahden leikkaustason (clipping plane) avulla. Leikkaustasot ovat kameran lähi- ja kaukorajat (kuvio 2). Leikkaus suoritetaan malleille, jotka ovat vain osittain edellä mainitun tilan sisällä. Lopuksi näytönkartoitus ottaa edellisten vaiheiden rajaaman osan pelimaailmasta ja kääntää sen näkymän näyttökoordinaateiksi. (Akenine-Möller ym. 2018, 19—20.)




KUVIO 2. Kameran frustum-kartio, johon on asetettu leikkaustasot

Rasterointi on vaihe, jossa projisoinnista ja leikkaamisesta selvinneistä primitiiveistä etsitään näkyvät pikselit. Kaksiulotteisen näyttöavaruuden verteksit muutetaan pikseleiksi näytölle (kuvio 3). Yksinkertaisimmillaan jokaisen pikselin keskipisteelle etsitään sen kanssa päällekkäinen mallin primitiivi ja niille luodaan yhteys. Reunanpehmennys (anti-aliasing) voi monimutkaistaa tätä operaatiota lisäämällä pikselin vertailupisteitä. Käytetyn reunanpehmennystekniikan mukaan kuvasta voi tulla paljon sileämmän näköinen. Kaikki rasteroinnin löytämät pikselit lähetetään pikselien prosessoinnille. (Akenine-Möller ym. 2018, 21—22.)



KUVIO 3. Rasteroitu malli

Pikselien prosessointi suorittaa pikselien värityksen operaatiot. Vaihe jakautuu kahteen osaan: pikselien varjostukseen ja pikselien yhdistämiseen. Pikselien varjostuksessa ohjelmoijan antama varjostin (kuvattu luvussa 2.3) suorittaa pikseleille tarkoitetut laskentatehtävät. Yksi mahdollinen tehtävä on pintakuviointi, jossa tekstuuri asetetaan mallin päälle. Yksinkertaisimmillaan tulos voi olla oma väri joka pikselille. Suoritetut tehtävät riippuvat täysin siitä, mitä ohjelmoija haluaa varjostimella tehdä. (Akenine-Möller ym. 2018, 22—23; Halladay 2019, 3.)

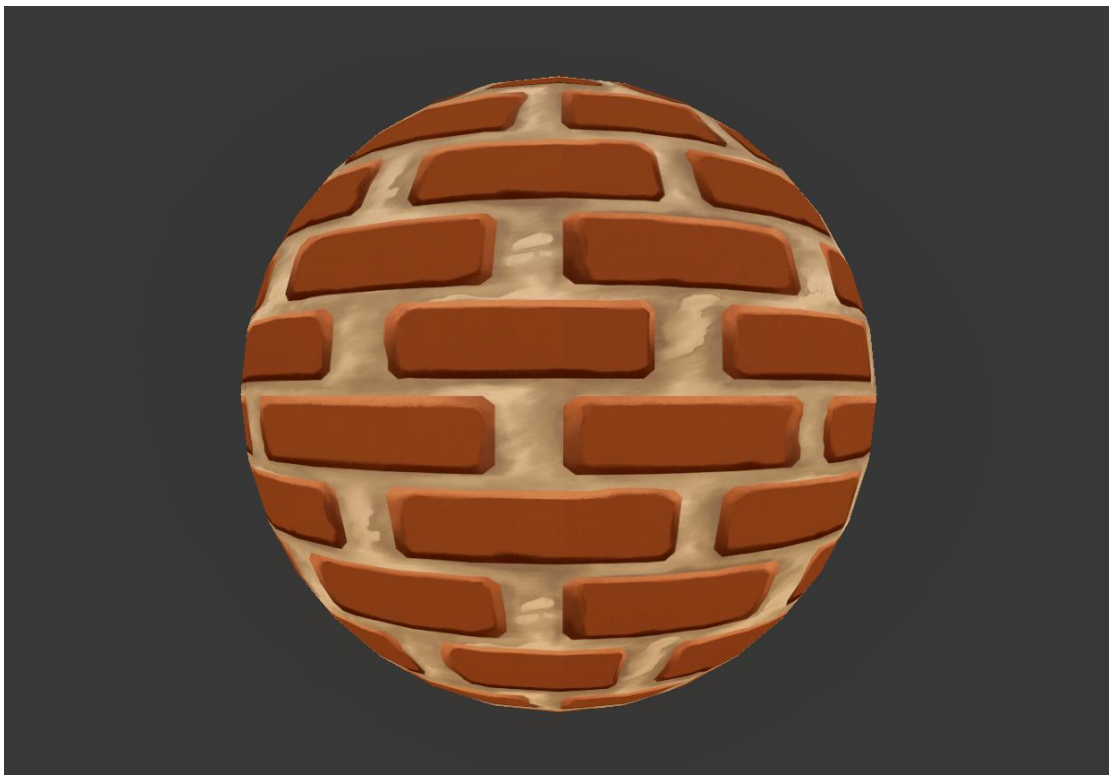
Pikselien yhdistämisessä pikselin lopullinen väri saadaan yhdistämällä edellisessä vaiheessa laskettu väri väripuskuriin (color buffer) tallennetun värin kanssa. Syvyyspuskurin (depth buffer) avulla selvitetään primitiivin näkyvyys. Syvyyspuskuri sisältää primitiivien etäisyydet kamerasta. Tyypillisesti sitä käytetään valitsemaan lähin primitiivi, kun niitä on tallessa useampi. Pikselien yhdistämisen vaihe voi sisältää muitakin puskureita, joilla lopullista kuvaa käsitellään, mutta nämä ovat tärkeimmät. Yhdistämisen jälkeen kuva on valmis näytöllä näytettäväksi. (Akenine-Möller ym. 2018, 24—25.)

Renderöintiputken malleja on useita erilaisia vaihtuviin tarkoituksiin, mutta yllä kuvattu runko vastaa niille yleisintä rakennetta. Unityllä on myös oma versionsa renderöintiputkesta. Vanha projekti on luotu Unityn versiossa 2017.3.0f3, joka käyttää Built-in Render Pipelinea.

## 2.3 Varjostimet

Kyle Halladay (2019) määrittelee varjostimen (shader) renderöintiä ohjaavaksi ohjelmaksi. Prosessorissa suoritetuista ohjelmista poiketen varjostimet ajetaan näytönohjaimessa. Ne antavat kehittäjien hallita renderöintiputkessa tapahtuvia muokattavia operaatioita. (Halladay 2019, 1.)

Varjostimien tärkeimmät tehtävät liittyvät verteksien ja pikselien käsittelyyn. Verteksien varjostamiseen käytetään verteksivarjostimia (vertex shader) ja pikselien varjostamiseen pikselivarjostimia (pixel shader), joista jälkimmäiset tunnetaan myös fragmenttivarjostimina (fragment shader). Esimerkkinä fragmenttivarjostimessa asetetaan tekstuuri (kuva 2).



KUVA 2. Palloon on lisätty sarjakuvamainen tiiliseinää esittävä tekstuuri

Varjostimia kirjoitetaan erityisillä grafiikan ohjelmointikielillä, kuten OpenGL Shading Language (GLSL) tai High-Level Shading Language (HLSL). Varjostimien luomiseen on olemassa myös artisteille suunnattuja graafisia työkaluja, kuten Unity-pelimoottorin Shader Graph. Unity on pitkään käyttänyt HLSL-kieltä varjostimissaan, ja myös Shader Graphilla luodut varjostimet pohjautuvat siihen. (Unity Documentation 2020a.)

## 2.4 Projektin tekniset tarpeet

Jotta voidaan lähteä suunnittelemaan parasta etenemistapaa projektin päivittämiselle, tulee ensin selvittää sen tarpeet. Dante’s Infernyan julkaistulla versiolla ei ole korkeita graafisia vaatimuksia. Pelin visuaalinen tyyli on erittäin sarjakuvamainen, ja se käyttää materiaaleissaan vain tekstuureja, normal-karttoja ja toisinaan myös metallic-karttoja. Valtaosa pelin taideresursseista siis lienee esitettävissä Unityn sisäisillä perusvarjostimilla. Tyylistä huolimatta peli ei pyörinyt kovin tehokkaasti vanhemmilla tietokoneilla, joten olisi hyvä optimoida sitä lisää. Erityisesti tiimin omat HLSL-varjostimet tehtiin ensisijaisesti näyttävyys mielessä huomioimatta niiden kuormitusta.

Pelihahmo käyttää useita materiaaleja tietyn tyylin saavuttamiseksi. Saman vaikutelman luominen eri työkaluilla saattaa käydä laskennallisesti raskaaksi tai kohdata muita teknisiä ongelmia. Toisaalta materiaalien määrän vähentäminen voisi olla laskennallisesti tehokkaampaa, sillä se vähentäisi myös mallin piirtokutsujen määrää. Tämä riippuu varjostinten toteutustavasta, mutta lähtökohtaisesti on yleensä tehokkaampaa hoitaa mallin piirtäminen yhdessä varjostimessa.

Valaistus ei ole keskeinen osa peliä, mutta sitä käytetään tukemaan pelimekaniikkoja. Kaikki valonlähteet ovat reaaliaikaisia, eli niiden luomat valot ja varjot muuttuvat pelin aikana. Pelaaja voi sammuttaa talon valoja katkaisijoista. Tämä tekee peliympäristöt väliaikaisesti pimeämmiksi, mikä vuorostaan korostaa itsevalaisevien esineiden näkyvyyttä (kuva 3). Valoja ei ole useita, mutta niiden luomien varjojen tulisi näyttää riittävän teräviltä. Peli on käyttänyt netistä löydettyä

kolmannen osapuolen ratkaisua varjojen terävöittämiseen, ja sama vaikutus tulisi saada aikaan uudessa versiossa.


KUVA 3. Esineiden reunat erottuvat taustasta, kun valoja on kytketty pois päältä

Dante’s Infernya käyttää erikseen asennettavaa Post-processing Stack v2 -pakettia jälkikäsittelyn efekteihin. Tarjotuista efekteistä käytössä on fog, anti-aliasing, ambient occlusion, depth of field, bloom ja color grading. Saman ulkonäön saavuttamiseksi uudessa ympäristössä pitäisi olla nämä samat ominaisuudet tarjolla.

# 3 RENDERÖINTIPUTKIEN KARTOITUS

## 3.1 Built-in Render Pipeline

Unityn alkuperäinen renderöintiputki tunnetaan nimellä Built-in Render Pipeline. Kaikki uudet perusprojektipohjaan luodut Unity-projektit käyttävät sitä, ja sitä käyttäen alkuperäinen Dante's Infernya on myös luotu. Alkuperäinen peli ei ollut kovin optimoitu eikä Built-in Render Pipeline tarjoa tähän yhtä paljon helposti käytettäviä ratkaisuja kuin muut vaihtoehdot. Tästä syystä voidaan olettaa, että tätä putkea ei käytettäisi pelin jatkokehityksessä.

Koska on tarkoitus vaihtaa Built-in Render Pipeline johonkin toiseen, ei ole tarpeen käydä sen ominaisuuksia läpi. Tiivistettynä sen sekä hyvä että huono puoli on sen kaikenkattavuus. Putkella voi tehdä mitä vain kehittäjä haluaa, mutta se myös sisältää kaikki mahdolliset ominaisuudet. Tämä lisää turhaan projektin renderöinnin yleiskustannuksia, kun tyypillisesti yhdellä pelillä on rajalliset tarpeet.

Unityyn on Built-in Render Pipelinen jälkeen lisätty vaihtoehtoisia putkia. Ne ovat tiettyihin käyttötarkoituksiin optimoituja. Pitää kartoittaa niiden keskeiset ominaisuudet ja eroavaisuudet, jotta voitaisiin valita niiden välillä Dante's Infernyalle sopiva.

## 3.2 Scriptable Render Pipeline -putket

Scriptable Render Pipeline (SRP) on Unityn kehittämä ohjelmointirajapinta (API), joka antaa kehittäjän ajoittaa ja säätää renderöinnin komentoja C#-skripteillä (Unity Documentation 2020b). SRP:n avulla kehittäjät voivat luoda alusta alkaen oman renderöintiputken, tai muokata olemassa olevaa putkea. Se ei siis itsessään ole renderöintiputken ratkaisu kuten Built-in Render Pipeline, mutta Unityn uudet putket on rakennettu sen päälle.

SRP:n etu on sillä luodun putken muokattavuus. Kehittäjät voivat kirjoittaa SRP:llä täysin tarpeisiinsa optimoidun renderöintiputken. Tämä on kuitenkin vaativa prosessi, joka kasvattaa projektin laajuutta huomattavasti. Uuden renderöintiputken luominen tältä pohjalta myös olettaa, että kehitystiimistä löytyy grafiikkaohjelmoinnin taitoa.

Unity on julkaissut SRP:n yhteydessä kaksi sitä käyttävää putken mallia. Ne ovat käyttövalmiita ja Unityn aktiivisesti tukemia. Useimpien kehittäjien tarpeisiin kyseiset valmispaketit ovat riittäviä. SRP:n avoimen rakenteen kautta niitä on myös mahdollista muokata ja antaa niille lisäominaisuuksia. Tässä projektissa viitataan molempien polkujen tapauksessa versioon 7.5.1, koska ne olivat uusimmat Unityn Long Term Support -version kanssa yhteensopivat versiot. Projektissa käytettiin siis Unityn versiota 2019.4.13f1.

Versiosta riippumatta yksi SRP:n suurimmista uudistuksista on uusi Volume-järjestelmä, johon jälkikäsittelyn efektit on siirretty. Volume on koko peliin vaikuttava tai tietyissä pelimaailman koordinaateissa toimiva kokoelma asetuksia. Tiettyyn paikkaan sidottu Volume tulee käyttöön, kun kamera menee sille määritetyn 3D-tilan sisälle. Tätä kautta jälkikäsittelyn efektit, kuten depth of field, voivat muuttua pelaajan liikkuessa ympäristöstä toiseen.

### 3.2.1 High Definition Render Pipeline

High Definition Render Pipeline (HDRP) on Unityn tarjoama, SRP:llä rakennettu putki. Se on tarkoitettu korkeatasoista grafiikkaa vaativille projekteille. Sen merkittävimpiä ominaisuuksia ovat monipuoliset menetelmät valaistuksen ja varjojen toteuttamiseen, uniikit materiaalit erityistarpeisiin ja tuki uudemmille korkean tason grafiikkaan tähtääville teknologioille, kuten realistisimman mahdollisen valaistuksen pohjana toimivalle säteenseurannalle (raytracing). (Unity Documentation 2020c.)

HDRP:n alustatuki on suhteellisen suppea sen projekteilta oletettujen korkeiden järjestelmävaatimusten takia. Tuetut konsolit ovat Xbox One ja Playstation 4. Muut alustat ovat kaikki laskennallisia varjostimia tukevat laitteet, jotka käyttävät

jotain seuraavista rajapinnoista: DirectX 11, DirectX 12, Metal ja Vulkan. Tämän perusteella HDRP:n pitäisi toimia esimerkiksi iPhonella, jos sen kyseinen malli tukee kyseisiä varjostimia. Laskennalliset varjostimet (compute shader) tekevät tyypillisesti prosessorille tarkoitettuja laskentatoimia näytönohjaimella. VR-kehitykseen tuettuja alustoja on suppeasti.

Putkessa on valojen käyttäytymiseen monta vaihtoehtoa, jotka keskittyvät Physically-Based Lighting -tekniikoihin. Valojen laskemisessa käytetään Physical Light Unit -yksiköitä (PLU). Ne on tarkoitettu toimimaan parhaiten pelimaailmoissa, joiden mittasuhteet vastaavat tosielämää. HDRP tarjoaa kaksi vaihtoehtoista renderöintitapaa. **Suora renderöinti** (forward rendering) laskee valot kerran jokaista materiaalia kohden. Se on usein parempi, kun pelissä ei ole paljon valoja. **Viivästetty renderöinti** (deferred rendering) piirtää kaikki näytöllä näkyvät materiaalit ensin geometriapuskuriin (G-buffer), ja sitten laskee valot jokaiselle näytön pikselille. Lisäksi vaihtoehtona on käyttää molempia, jolloin asetuksen voi tehdä erikseen jokaista kameraa kohden.

HDRP:n materiaaleissa on tuki monille korkean tason grafiikan ominaisuuksille, kuten detail-kartat ja subsurface scattering. Näiden hyödyntäminen vaatii projektilta niiden käyttämät taideresurssit. Erityiskäytön valmiita varjostimia HDRP:llä on esimerkiksi hiuksille, kankaille ja siirtokuville (decals), jotka ovat maailman pinnoille projisoituja yksittäisestä mallista irrallisia tekstuureita.

SRP:n Volume-ominaisuutta voi HDRP:ssä käyttää jälkikäsittelyn efektien lisäksi myös muuttamaan sumun, valon ja varjon asetuksia. Tätä kautta on käytettävissä myös esim. ambient occlusion -asetus, joka lisää realismia simuloimalla varjoja ahtaissa paikoissa, ja screen space reflection -asetus, jolla luodaan heijastuksia vaikkapa veden pintaan. Reunanpehmennys on muokattavissa kamerakohtaisesti, ja sen toteutukseen on seuraavat vaihtoehdot: FXAA, TAA, SMAA ja MSAA. Näistä MSAA (Multisample anti-aliasing) ratkaisee reunanpehmennyksen ongelmat paremmin kuin muut, mutta on myös laskennallisesti raskain.

Monet HDRP:n ominaisuudet vaikuttavat pieniin yksityiskohtiin ja niiden oikeaoppisella yhteiskäytöllä voi luoda erittäin realistisen kuvan. Sitä voi käyttää

myös tyylitellympiin graafisiin ilmeisiin, mutta yksityiskohtainen valaistus ja varjot ovat kaikille HDRP-projekteille yleisiä.

### 3.2.2 Universal Render Pipeline

Universal Render Pipeline (URP) on toinen SRP-pohjainen, Unityn mukana tuleva putki. Se on tarkoitettu mobiili- ja XR-alustoille. URP:n uniikit ominaisuudet ovat sen helppokäyttöiset työkalut, 2D-valaistuksen tuki ja tuettujen kohdealustojen määrä. (Unity Documentation 2020d.)

URP:n tukemat konsolit ovat Xbox One, PlayStation 4 ja Nintendo Switch. Lisäksi URP tukee seuraavia alustoja: Windows, Universal WIndows Platform, Mac, iOS, Android, WebGL ja useimmat VR-alustat. URP:ssä on laaja tuki eri VR-alustoille.

Kolmiulotteisiin maailmoihin soveltuvista renderöintitavoista URP voi käyttää vain suoraa renderöintiä. Tämän perusteella URP ei sovellu projekteihin, joissa käytetään intensiivisesti valoja. Valoissa on muitakin rajoituksia, kuten ei reaaliajan heijastusten varjoja Spot-tyypin valoille eikä varjoja ollenkaan reaaliajan Point-tyypin valoille. URP:n varjostuksessa on käytössä vain yksi varjostuskutsu (shader pass), mikä rajoittaa varjostinten monimutkaisuutta. Yksi kutsu renderöi mallin geometrian kerran. Näiden puutteiden vastapainoksi putki erikoistuu 2D-renderöintiin. URP tukee kaksiulotteisia valoja ja varjoja, mikä tekee siitä parhaan vaihtoehdon sprite-grafiikkaa käyttäville projekteille.

Materiaalit URP:ssä muistuttavat Built-in Render Pipelinen vastaavia, eikä niissä ole tukea erilaisille kartoille kuten HDRP:ssä. Realismiin tarkoitettu Lit-materiaali voi ottaa vastaan tekstuurin, specular-kartan, metallic-kartan, normal-kartan, occlusion-kartan ja emission-kartan. Perusmateriaaleista on myös tarjolla pelkistetyt versiot mobiilialustoja varten.

URP:n yksinkertaiset materiaalit ja 2D-tuki tekevät siitä erittäin hyvän putken mobiilipelien tai muiden laskennallisesti kevyeksi tarkoitettujen projektien toteuttamiseen. Realismiin tähtäävien ominaisuuksien puuttuminen myös varmistaa sen, että alustatuki on voitu pitää laajana.

### 3.3 Valinta ja perusteet

Kehitystiimillä ei ole sen pienen koon takia resursseja oman SRP-putken luomiseen. Tästä syystä valinta tehtiin URP:n ja HDRP:n välillä. Niiden tapauksessa olisi hyväksyttävää päätyä jatkokehittämään putken ominaisuuksia, jos projekti sellaista vaatisi.

Dante’s Infernya oli alun perin julkaistu Windowsille ja Macille. Sen potentiaalisen uuden version kohdealustoista ei vielä ollut tietoa, mutta nähtiin järkevänä pitää mahdollisuudet avoimina. Graafisen tyylin perusteella olisi pitänyt olla mahdollista siirtää peli mobiilialustoille menettämättä tyylin laatua merkittävästi. HDRP:n rajallisen alustatuen kannalta se vaikutti huonolta vaihtoehdolta.

Verratessa tarkemmin HDRP:n ja URP:n ominaisuuksia (Unity Documentation 2020e, 2020f) näytti siltä, että HDRP:ssä oli sillä hetkellä enemmän hyödyllisiä ominaisuuksia. Monet kiinnostavammat ominaisuudet, kuten siirtokuvat, olivat vasta kehityksessä URP:lle (taulukko 1). Toisaalta pelin vetovoima oli sen taiteellisessa tyylisuuntauksessa graafisen tarkkuuden sijaan. Tiimillä ei myöskään ollut käyttöä HDRP:n edistyneille materiaaleille, sillä niiden käyttämien karttojen tekeminen olisi ollut pienelle tiimille raskasta. Projektin alhaiset vaatimukset ja laaja kohdeyleisö mielessä pitäen lähdettiin siirtämään peliä URP:een.

TAULUKKO 1. HDRP:n ja URP:n merkittäviä eroja (Unity Documentation 2020e, 2020f)

| Ominaisuus | HDRP (7.5.1) | URP (7.5.1) |
|---|---|---|
| Viivästetty renderöinti | OK | Tulossa |
| Kamera | | |
| Reunanpehmennys | MSAA (suora renderöinti)<br>TAA<br>FXAA<br>SMAA | MSAA |
| Kameroiden pinoaminen (Camera stacking) | - | OK |
| Reaaliajan valot | | |
| Varjostus | Tiled/Clustered | Yksi kutsu |
| Säteenseuranta | OK | - |
| Globaali valaistus | | |
| Screen Space Reflections | OK | - |
| Sumu | | |
| Linear / Exponential Squared | - | OK |
| Local Volumetric | OK | - |
| Varjostimet | | |
| Siirtokuvat | OK | Tulossa |
| Materiaalien lisätuki (Anisotropic, Subsurface Scattering, Iridescence, Translucence) | OK | - |
| Detail-maskit | OK | - |
| Jälkikäsittelyn efektit | | |
| Ambient Occlusion | OK | Tulossa |
| Exposure | OK | - |
| Kameran liikkeen sumennus (Motion Blur) | OK | Tulossa |

# 4 PROJEKTIN PÄIVITTÄMINEN

## 4.1 Renderöintiputken käyttöönotto

URP-paketin lataamisen ja päälle kytkemisen jälkeen kaikki pelin mallit näkyivät kirkkaan pinkkeinä, mikä on Unityn tapa esittää rikkinäinen tai puutteellinen materiaali malleissa. Tämä johtui siitä, että URP käyttää täysin omia materiaalejaan. Vanhat Unityn sisäänrakennetut varjostimet ja tiimin kirjoittamat varjostimet eivät ole yhteensopivia, ja ne täytyi päivittää.

Sisäänrakennettuja varjostimia varten on oma työkalu, joka käy läpi projektin materiaalit ja korvaa niissä käytetyt yhteensopimattomat varjostimet URP:n vastaavilla. Built-in-putken Standard-varjostin korvautui vastaavalla Lit-varjostimella. Yleisimmät mallien tarpeet koskivat tekstuureja ja normal-karttoja. Koska nämä syötteet löytyivät Lit-varjostimesta, se todettiin riittäväksi. Tämän lisäksi pelihahmon kissanviiksien aiemmin käyttämälle Tree Soft Occlusion Leaves -varjostimelle ei löytynyt suoraa vastinetta. Se on tarkoitettu litteiden puunlehtien renderöintiin, mutta se sopi myös kuvana esitettyihin kissanviiksiin. Tähän tarkoitukseen valittiin Sprite Lit Default -varjostin, joka on 2D-kuvien perusvarjostin. Tämän ainoaksi ongelmaksi ilmeni se, että varjostin ei reagoi 3D-valoihin.

Siirron jälkeen valaistuksessa oli uusi ongelma. Kaikki pelin valot olivat Point Light -tyyppisiä, joita URP tukee vain etukäteen leivottuina Baked-tyyppisinä valoina. Leivotut valot eivät voi antaa valaistusta tai varjoja liikkuville objekteille. Pelin visuaaliselle ilmeelle oli kuitenkin eduksi, että liikkuvilla hahmoilla ja esineillä on nämä. Kompromissiksi valot vaihdettiin Spot Light -tyyppisiksi, joka ei yhtä hyvin vastaa valaisimien mallin luomaa odotusta valaistuksesta, mutta sillä tuotettu valo päivittyy reaaliajassa. Vanhan ja uuden valoratkaisun erot näkyvät kuvassa 4.

KUVA 4. Vertailussa pelin julkaisuversion valot (vasen) ja URP-putkeen siirretty versio, joka käyttää Spot-tyypin valoja

Jälkikäsittelyn efektit saatiin URP:n omasta Post-processing-resurssipaketista. Vanhaan versioon verrattuna joitakin ominaisuuksia oli siirretty ulos paketista, mutta niiden löydyttyä asetukset saatiin ennalleen. Näiden muutosten jälkeen peli muistutti jo hyvin paljon alkuperäistä versiota. Projektia varten luodut varjostimet olivat vielä rikki, ja ne pyrittiin korvaamaan Shader Graph -työkalun avulla.

### 4.2 Varjostimien suunnittelu

Projektin vanhoja, HLSL-ohjelmointikielellä kirjoitettuja varjostimia ei voitu automaattisesti muuttaa Universal Render Pipelinen kanssa yhteensopiviksi. URP:n mukana tullut Shader Graph on visuaalinen työkalu, jolla varjostimia luodaan kaaviokuvina. HLSL-varjostimien uudelleen kirjoittamisen sijaan päätettiin rakentaa uudet tällä työkalulla. Päätökseen vaikutti työkalun saama aktiivinen tuki Unityltä sekä kirjoitettujen varjostimien manuaaliseen päivittämiseen liittyvän dokumentaation alhainen taso.

Shader Graphin käytössä tulisi haasteeksi sen rajallinen määrä efektejä jokaista yksittäistä varjostinta kohden. Esimerkiksi kissahahmon monimutkaista

HLSL-varjostinta ei välttämättä voisi olla mahdollista luoda sellaisenaan. Tarvitut lisätoiminnot voisi kirjoittaa ja lisätä SRP:n työkaluilla, jos sellaisiin löytyisi mallitoteutukset internetistä.

### 4.2.1 Esineiden rajausvarjostimen suunnittelu

Pelissä pelaajan tehtävä on kaataa ja rikkoa talon tasoilla olevat esineet. Nämä esineet ovat satunnaisesti ohjelman asettamia ja niitä on monia erilaisia. Ne ovat myös usein pieniä pelimaailman kokoon nähden, joten pelaajan voi olla vaikea löytää niitä kaikkia. Esineiden näkyvyyden parantamiseksi niillä on reunoja korostava varjostin. Taustalla toimiva skripti säätelee reunojen paksuutta, muuttaen niitä leveämmiksi sitä mukaa, kun pelaaja etenee ja jäljellä olevat esineet vähenevät. Tämä edelleen lisää näkyvyyttä, kun pelaajan silmä etsii yhä harvinaisempia kohteita. Toinen pelin aikana muutettu arvo on reunuksen väri, jolla esineet erotetaan toisistaan tyypin mukaan.

Pelitestauksessa oli huomattu, että esineitä oli edelleen vaikea huomata pelimaailmasta. Yksi tunnistettava syy oli siinä, että korostus oli yksivärinen ja kirkas. Pelin värimaailma oli muutenkin värikäs ja kirkas, joten korostuksella olisi pitänyt olla näiden lisäksi jotain muuta hypätäkseen pelaajan huomioon. Yksinkertaisin ratkaisu olisi lisätä efektiin liikettä.

Toinen ongelma löytyi rajauksen toteutustavasta. Vanha varjostin toimi siten, että alkuperäisestä mallista skaalattiin isompi kopio samaan paikkaan ja asentoon. Tuloksena saatu malli väritettiin korostusvärillä ja työnnettiin poispäin kamerasta piirtymään alkuperäisen mallin taakse luoden reunustuksen vaikutelman. Verteksien työntäminen ei aina tuottanut mallin muotoa vastaavaa tai edes kovin nättiä tulosta (kuva 5). Efekti paheni entisestään, kun reunustusta suurennettiin esineiden vähetessä. Tarkemmin mallin muodossa pysyvä ratkaisu olisi piirtää efekti mallin päälle osaksi sen väritystä. Korostuksen sijainnin valitseminen mallin ja sitä ympäröivän maailman rajan perusteella voisi luoda luotettavamman efektin. Tämän rajan voisi saada kamerasta käyttämällä sen syvyyspuskuria.

KUVA 5. Reunoja ei voi kasvattaa liikaa, tai ne menevät toisistaan ja ympäristöstä läpi

### 4.2.2 Hahmon toon-varjostimen suunnittelu

Dante’s Infernyassa pelaaja ohjaa kissaa, joka on kamerakulman takia aina keskeisessä osassa näytöllä. Sille haluttiin uniikki, silmiinpistävä ja värikäs ulkoasu. Kissan roolin kannalta nähtiin sopivaksi, että se olisi myös tyylisuuntaukseltaan erottuva. Muissa pelin malleissa käytetystä varjostustyylistä poiketen tämän hahmon varjostus perustui sarjakuvamaiseen toon shading -tekniikkaan.

Akenine-Möllerin ym. (2018, 652) määritelmän mukaan toon shading -varjostuksessa mallit piirretään kiinteillä väreillä, joilla on selkeät rajat. Toon-varjostukselle on useita eri tyylejä. Yksi tyypillisin on kahden sävyn varjostus, jossa pikselille annetaan yksi kahdesta väristä. Värin valinta tapahtuu vertaamalla kehittäjän antamaa rajaa kuvaavaa arvoa mallin normaalin ja valon tulosuunnan pistetuloon (dot product). Määrittämällä lisää rajoja voidaan lisätä värikerrosten määrää. On myös mahdollista määrittää valon intensiteetti

uudelleen tietyiksi väreiksi käyttämällä yksiulotteista tekstuuria. Tästä tekstuurista käytetään usein termiä **toon ramp**. Kuvassa 6 on käytetty toon ramp -tekstuuria, jossa värit ovat keltainen, valkoinen ja musta. Toon-varjostukselle läheisiä ovat ääriviivat, joista yksinkertaisin on siluetti. Sillä erotetaan malli taustasta.

KUVA 6. Diffuse-varjostus (vasen) ja kolmen värin Toon-varjostus (oikea)

Kissahahmo käytti toon-varjostuksesta ja siluetista johdettua yhdistelmää, joka tuotti valon suunnan perusteella hahmolle kirkkaan rajauksen. Tämän efektin toteutustapa kuitenkin peitti toisten vaikutusta, ja kissa ei saanut varjoja. Varjostimen uuden version tulisi vastaanottaa myös varjoja kunnolla, jotta hahmo näyttäisi olevan osa muuta pelimaailmaa. Hahmolle oli olemassa käyttämätön normal-kartta ja koettiin, että sekin olisi hyvä saada käyttöön.

### 4.2.3 Uuden läpinäkyvyysvarjostimen suunnittelu esineille

Peli käytti hahmoon sidottua kolmannen persoonan kameraa. Kamera lepäsi pienellä etäisyydellä hahmon vasemmasta tai oikeasta kyljestä, jota pelaaja pystyi halutessaan vaihtamaan. Pelin ohjausmalli odotti pelaajalta, että hän

liikuttaa hahmoa yhdellä kädellä ja kameraa toisella. Kameran sijainti vaihtui myös automaattisesti skriptin toimesta, jonka tehtävä oli estää esineitä tulemasta kameran tielle. Pelin aikana pelaaja kuitenkin joutui usein ahtaisiin tiloihin tai seinien viereen navigoidessaan pelimaailmassa. Tämä ajoittain sai skriptin heittelemään kameraa edestakaisin häiritsevästi. Skriptiä ei sopisi poiskaan ottaa, sillä siitä on joillekin pelaajille apua. Ilman sitä pelaaja joutuu täyteen vastuuseen kameran kohdistamisesta.

Toinen lähestymistapa olisi muuttaa ympäristön renderöintiä. Esineitä voisi muuttaa läpinäkyviksi, kun ne tulevat pelaajan ja kameran väliin. Näiden näköesteiden täytyisi muuttua vain osittain läpinäkyviksi, sillä muuten pelaaja voisi hämmentyä täysin katoavista huonekaluista. Koko malliin tasaisesti lisätyn läpinäkyvyyden sijaan esineisiin tehtäisiin varjostimessa reikä, joka paljastaa pelaajan ja muun pelimaailman esineen takana. Tämä päätettiin toteuttaa uutena varjostimena, jota käytettäisiin ympäristön kiinteisiin esineisiin, kuten huonekaluihin.

### 4.3 Varjostimien toteuttaminen

Shader Graph virtaviivaistaa tyypillisiä varjostimien tehtäviä Master-nodeilla. Luotuun kaavioon kuuluu aina yksi node, jonka rooli on vastaanottaa kaikki käsitelty data lopullista käyttöönottoa varten. Uudet varjostimet päädyttiin tekemään Physically Based Rendering -tekniikkaan (PBR) pohjautuvaan kaaviopohjaan. PBR on realismiin tähtäävä renderöintitekniikka (Pharr & Humphreys 2010, 1).

PBR Graph -pohja oli sopiva projektin varjostimiin, sillä sen suurin ero muihin pohjiin oli tuki valon vastaanottamiselle. Sillä kaikkiin malleihin saatiin helposti visuaalinen yhtenäisyys. Tämä oli tärkeä varmistaa heti alkuun, sillä myöhemmin varjostimia toteutettaessa niiden tyylit voisivat helposti alkaa erota toisistaan.

### 4.3.1 Esineiden rajausvarjostimen päivittäminen

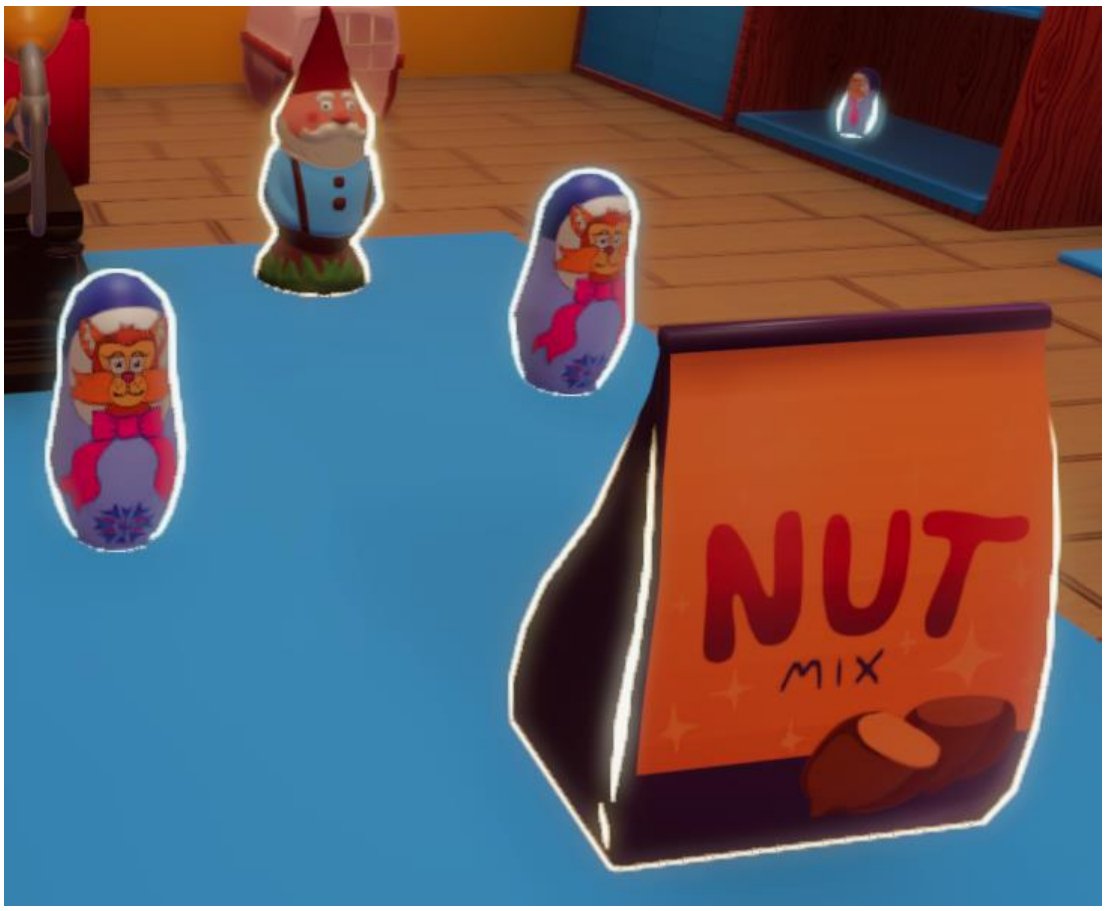
Malliksi löydettiin ratkaisu, jossa oli käytetty pipeline-resurssin Depth Texture -ominaisuutta, joka antaa kameran luoda syvyyttä kuvaavan tekstuurin varjostimien käytettäväksi. Mallin etäisyyttä kamerasta ja sen normaalien rajoja vertaamalla saatiin löydettyä reunojen sijainti. Tekniikan avulla reunojen piirtämiseen valittiin vain mallista löytyvät pikselit, ja reunus pysyi mallin päällä.

Korostukseen saatiin paksuuden tuntua käyttämällä reunusta emissiosyötteenä. Emissiovärit käyttävät pipelinen HDR-värejä luodakseen kirkkaan värin. Kameraan liitetty jälkikäsittelyn bloom-efekti saa ne hohtamaan. Nämä hohtavat reunukset näyttävät vuotavan mallista ulos ympäröivään maailmaan. Efekti on paljon sulavampi kuin vanha verteksien liikuttaminen, ja se sopii yhtä hyvin kaikkiin pelin esineiden malleihin.

Efekti animoitiin ilmestymään vaakasuorina viivoina mallin yläreunasta ja liikkumaan sitä pitkin alas — lopulta kadoten mallin pohjalla. Peräkkäiset viivat näyttivät aaltoilevan liikkeessä. Liikkuminen tapahtui ajan perusteella, eli pelin jatkuessa efekti pyörii taustalla automaattisesti. Liikkuva, hohtava värirajaus todettiin heti huomiota herättäväksi.

Vanhan rajausvarjostimen mukaisesti skripteille paljastettiin muuttujat rajauksen paksuudelle ja värille. Mahdollisesti myös animaatioon liittyvät muuttujat, kuten nopeuden, voisi ottaa huomioon pelitilanteen muuttuessa.

Varjostin tukee vanhan toteutuksen tapaan tekstuureita. Siihen lisättiin myös tuki metallic-kartoille ja normal-kartoille. Vain osalla esineiden malleista oli tällaiset kartat olemassa, mutta niitä päästiin kuitenkin kokeilemaan. Lopullinen vaikutelma näytti tavalliselta Lit-varjostimen materiaalilta, johon oli lisätty rajausefekti (kuva 7).




KUVA 7. Valmis varjostin

Ongelmaksi havaittiin läpinäkyvät esineet. Lasipullossa oli ollut käytössä kaksi materiaalia: etiketin ja korkin yhteinen materiaali ja lasimateriaali. Lasi ei näyttänyt reunuksia, sillä reunojen paikallistamisen skripti ei tunnistanut pulloa. Tässä vaiheessa ei voitu enää koettaa korjata ongelmaa. Vaihtoehtoina oli tehdä pullosta värjätty tai antaa reunuksen näkyä vain korkissa ja etiketissä.

### 4.3.2 Hahmon toon-varjostimen päivittäminen

Isoin ongelma kissan varjostimen kanssa olivat ennen ristiriitaiset efektit. PBR Graphin käyttäminen uuden version pohjana ratkaisi jo osan näistä. Peruskäytäntöjen mukaan varjostimen Master-nodelle syötettiin tekstuuri ja normal-kartta.

Unity-kehittäjä Alex Lindmanin blogipostia (2019) seuraten varjostimeen lisättiin keinotekoinen toon shading -efekti. Artikkelin mukaan olisi voitu hakea skriptillä

tiedot malliasetelman keskeisestä Main Light -valosta ja muista valoista erikseen ja tehdä niille omat operaatiot. Pelissä ei kuitenkaan käytetty Directional Light -tyyppistä ensisijaista valoa, joten varjostimessa käytettiin vain lisävaloja käsittelevät toiminnot. Valoista saatiin yhdistetty suunta ja intensiteetti, jota käytettiin toon rampin kanssa luomaan lisäkerros varjostusta hahmon päälle.

Valon heijastuksia simuloiva specular-osuus ajettiin vielä tekstuurin läpi. Suora specular-osuus olisi ollut liian muovisen näköinen karvaisessa eläimessä. Tekstuuri hajotti valon ja teki siitä tyylitellyn näköisen (kuva 8). Valon diffuse-osa käytettiin sellaisenaan, mutta alhaisella teholla.

KUVA 8. Valon specular-osuus siniseksi värjättynä

Skriptin hakeman valon diffuse-osaa käytettiin myös sivuvalon luomiseen. Efekti piirsi kissan päälle kirkkaan reunan, kun valo tulee hahmon takaa. Tämä efekti toimi myös silloin, kun valot on sammutettu. Se luo kontrastia silloin, kun peli on muuttunut harmahtavaksi ilman valoja. Kaikkien valoefektien jälkeen lopputulos

oli hahmo, joka ensinnäkin näytti kuuluvan ympäröivään maailman, mutta joka oli myös persoonallisen näköinen (kuva 9).

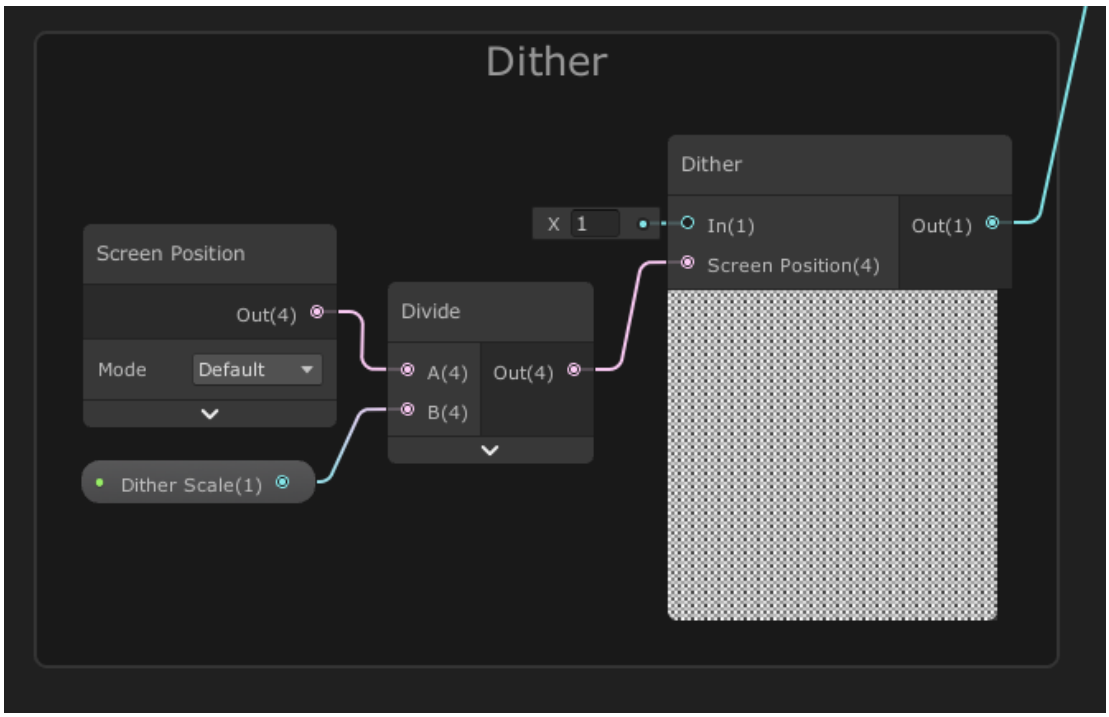
KUVA 9. Kissassa toon-varjostus harmaasävyisellä ramp-tekstuurilla, tyylitelty specular ja valkoinen sivuvalo

### 4.3.3 Esineiden läpinäkyvyysvarjostimen toteutus

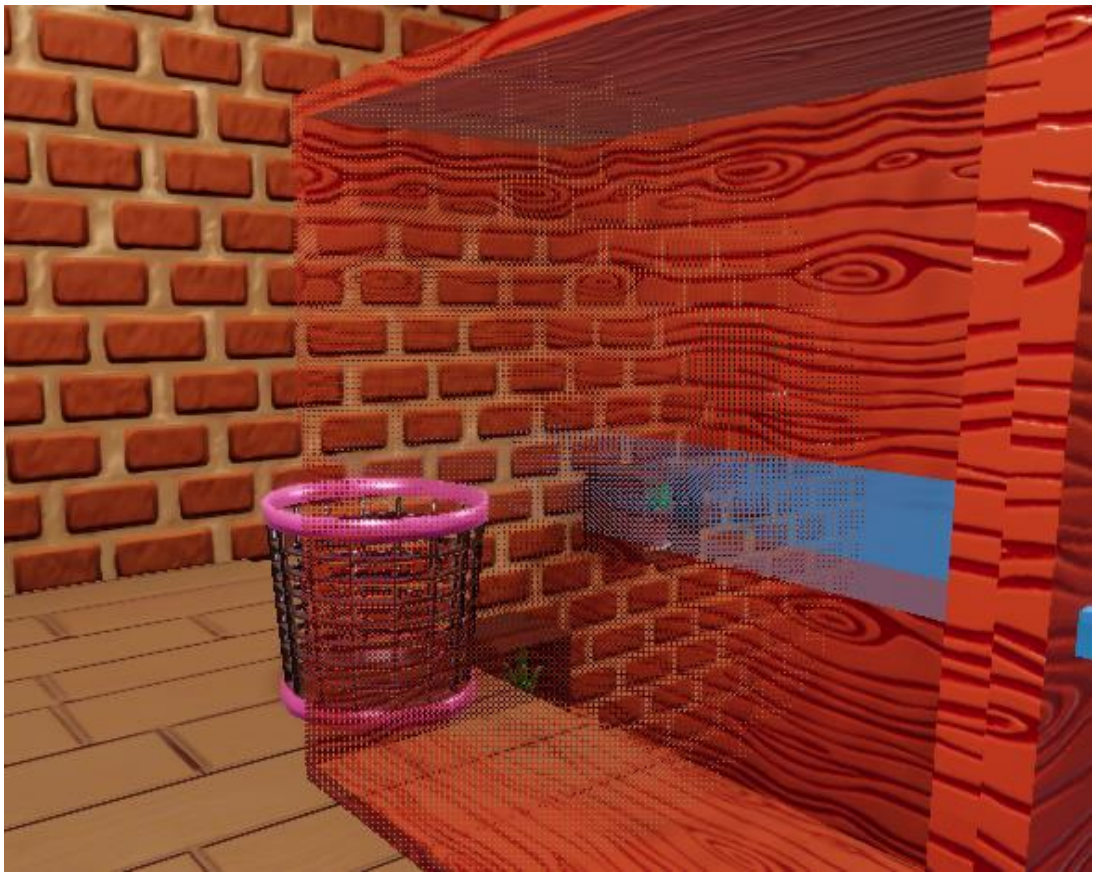
Alkuperäinen idea läpinäkyvyysvarjostimelle olisi käyttänyt vähitellen kasvavaa läpinäkyvyyttä pelaajaa peittävissä pikseleissä. URP:n heikkous tässä oli sen rajoitus yhteen varjostuskutsuun. Jotta Shader Graphista olisi saatu osittain läpinäkyviä pikseleitä, PBR Graphin Master-noden pinta täytyi vaihtaa läpikuultamattomasta (opaque) läpinäkyvään (transparent). Tämä yhdistelmä aiheutti häiriöitä koko mallin näkyvyydessä, sillä node ei antanut muuttaa renderöintijärjestykseen vaikuttavaa ZWrite-asetusta. Läpikuultamattomalla pinnalla pikselien näkyvyys oli joko 0 tai 1, mutta ei mitään niiden väliltä. Tämä loi tarkasti malliin projisoidun pyöreän reiän (kuva 10), mikä ei ollut kovin nättiä.

KUVA 10. Mallin läpi projisoitu reikä

Päädyttiin käyttämään sekoitussävytykseen (dithering) pohjautuvaa menetelmää. Liukuvasti näyttöä pitkin muuttuvan läpinäkyvyyden vaikutelma saavutettiin AlphaClipThreshold-syötettä säätämällä (kuvio 4). Sen arvo määrittää läpinäkyvyyden rajan, jonka alle jäävät pikselit hylätään. Läpinäkyvyysarvoa muutettiin pelaajan sijainnin mukaan. Pelaajan sijainti kääntyi näyttökoordinaatiksi, jota käytettiin Dither-noden keskipisteenä. Tulos oli yhdestä koordinaatista pyöreästi ulospäin harveneva ruudukko, jonka peittämällä alueella pikselit hylättiin paljastaen mallin taakse jäänyt maailma (kuva 11).




KUVIO 4. AlphaClipThreshold-syötteeseen kytketty Dither-node Shader Graphissa



KUVA 11. Sekoitussävytyksellä pehmennetty reiän projisointi

Varjostin otti skripteiltä syötteenä pelaajan sijainnin, läpinäkyvyysarvon ja efektin koon. Sijainti sääti efektin paikkaa, muut arvot sen intensiteettiä. Läpinäkyvyyden kattavuus siis muuttui sen mukaan, paljonko sitä tarvittiin pelaajan näyttämiseen ruudulla. Näillä efekti saatiin myös pois päältä silloin kun esine ei tullut kameran ja pelaajan väliin. Muilla syötteillä saatiin käyttöön vanhat assetit tekstuurit ja normal-kartat mukaan lukien, ja ne toimivat tavallisesti efektin kanssa. Huomiona voisi mainita, että valot ja varjot käyttäytyvät mallissa hieman erikoisesti heti, kun siihen ilmestyy reikiä. Tämä täytyy laskea vain efektin ominaisuudeksi.

# 5 POHDINTA

Opinnäytetyön tavoitteena oli vertailla renderöintiputkia ja ottaa selvää Dante's Infernyan varjostimien toteutuksesta valitussa sellaisessa. Putkien vertailussa onnistuttiin riittävästi, sillä valinta vaikutti jo pintapuolisen tutkinnan jälkeen selkeältä. Toteutusvaiheen jälkeen näytti edelleen siltä, että oli tehty oikea valinta polkujen välillä. Oli kuitenkin hyödyllistä perehtyä HDRP:n ja URP:n eroihin syvemmin, koska molempien oma dokumentaatio paljastui suhteellisen suppeaksi. Vertailemalla putkia ristiin tulivat niiden rajat selkeämmiksi. Perusteellisempi tutkimus olisi ollut tietysti molempien putkien käyttöönotto ja käytännön kokeilu, mutta sellaisella projektilla olisi huomattavasti isompi laajuus.

Varjostimien toteutus onnistui erittäin hyvin. Kaikista efekteistä tuli alkuperäisiä paremman näköiset ja teknisellä tasolla peliin sopivat. Valitussa URP-putkessa pelin efektit toimivat myös laskennallisesti tehokkaammin. Esille nousseita ratkaisemattomia ongelmia oli kuitenkin muutama, kuten valot ja lasipullon läpinäkyvyysongelma. Lasipullo päätyi näyttämään lähes samalta kuin ennen päivittämistä, joten siinä tapauksessa ei epäonnistuttu merkittävästi. Ongelman vaikutus peliin on myös pieni. Muuttuneiden valojen vaikutus on paljon suurempi. Siinä tapauksessa on jo kyse pelin visuaalisen tyylin muutoksesta. Se on kysymys, joka jää kehitystiimin pohdittavaksi.

Opinnäytetyöstä pitäisi olla hyötyä aloitteleville Unity-kehittäjille tai niille, joilla on kokemusta Unitystä mutta ei sen SRP-systeemistä. Työ nostattaa esille käytännön ongelmia, kuten valojen tyypin muutoksen. Kehittäjille on eduksi herätä ajattelemaan tällaisia asioita, jos jo työstettyä projektia aiotaan siirtää putkesta toiseen. Koska Unityn kaltaisessa pelimoottorissa on aina jonkinlaiset perusasetukset valmiiksi päällä, voivat jotkin itsestäänselvyyksinä pidetyt ominaisuudet kadota pelistä siirrossa. Vastaavaa oikean pelin siirtoprojektin dokumentointia ei löytynyt internetistä helposti.

Jatkotutkimusaiheena voisi lähestyä renderöintipolun valitsemisen kysymystä toisenlaisen peli-idean näkökulmasta, joka käyttäisi HDRP-putkea. Käytännön puolelta tämä työ ei koskenut HDRP-putkeen, ja sitä voisi olla jollakin

kiinnostuneella taholla syy tutkia. Yleisellä tasolla URP ja HDRP kehittyvät koko ajan ja tämän työn peruskysymyksiä voi olla syytä kysyä uudestaan.

**LÄHTEET**

Akenine-Möller, T., Haines, E., Hoffman, N., Pesce, A., Iwanicki, M., Hillaire, S. 2018. Real-Time Rendering. 4. painos. Boca Raton: CRC Press.

Butterfield, A., Ekembe Ngondi, G., Kerr. A. 2016. A Dictionary of Computer Science. 7. Painos. E-kirja. Oxford: Oxford University Press. https://www.oxfordreference.com/view/10.1093/acref/9780199688975.001.0001/acref-9780199688975

Halladay, K. 2019. Practical Shader Development: Vertex and Fragment Shaders for Game Developers. E-kirja. USA: Apress. https://learning.oreilly.com/library/view/practical-shader-development/9781484244579/

Lindman, A. 2019. Custom Lighting in Shader Graph: Expanding your graphs in 2019. Blogi. Julkaistu 31.7.2019. Luettu 5.11.2020. https://blogs.unity3d.com/2019/07/31/custom-lighting-in-shader-graph-expanding-your-graphs-in-2019/

Pharr, M., Humphreys, G. 2010. Physically Based Rendering: From Theory to Implementation. 2. painos. San Francisco: Morgan Kaufmann Publishers

Unity Documentation. 2020a. Shading language used in Unity. Luettu 12.11.2020. https://docs.unity3d.com/Manual/SL-ShadingLanguage.html

Unity Documentation. 2020b. Scriptable Render Pipeline introduction. Luettu 12.11.2020. https://docs.unity3d.com/Manual/scriptable-render-pipeline-introduction.html

Unity Documentation. 2020c. High Definition Render Pipeline overview. Luettu 12.11.2020. https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@7.5/manual/index.html

Unity Documentation. 2020d. Universal Render Pipeline overview. Luettu 12.11.2020. https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@7.5/manual/index.html

Unity Documentation. 2020e. High Definition Render Pipeline/Built-in Render Pipeline comparison. Luettu 2.11.2020. https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@7.5/manual/Feature-Comparison.html

Unity Documentation. 2020f. Feature comparison table. Luettu 2.11.2020. https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@7.5/manual/universalrp-builtin-feature-comparison.html