Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Miikka Vainikainen

Jälkikäsittelytehosteet pelikehityksessä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

7.5.2020

Tekijä Otsikko Sivumäärä Aika	Miikka Vainikainen Jälkikäsittelytehosteet pelikehityksessä 48 sivua 7.5.2020
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Pelisovellukset
Ohjaaja	Lehtori Antti Laiho

Insinöörityön tarkoituksena oli perehtyä Unity-pelimoottorin jälkikäsittelyn työkalujen asennukseen, erilaisiin renderöintipolkuihin ja asennettujen jälkikäsittelyn työkalujen ominaisuuksiin, rajoitteisiin ja asetuksiin. Unity-pelimoottorin tarjoaman jälkikäsittelyn työkalun eri tehosteisiin tutustumisen jälkeen oli tarkoitus luoda omia jälkikäsittelyn tehosteita käyttäen HLSL-ohjelmointikieltä ja luoda kolme erilaista varjostinta ja työkalua käyttäen C#-ohjelmointikieltä, jonka avulla luodut tehosteet olisivat käyttäjän saatavilla ja niitä pääsisi säätämään pelikameraan luodun skriptin avulla.

Insinöörityöraportissa tutustuttiin Unity-pelimoottorin pakettimanagerin tarjoaman jälkikäsittelyn työkalun asennukseen askel askeleelta käyttäen havainnollisia kuvia ja vinkkejä. Jälkikäsittelyn työkalujen asennuksen jälkeen perehdyttiin eri renderöintipolkujen valintaan ja sen tärkeyteen henkilökohtaista peliprojektia aloittaessa. Unity-pelimoottorin pakettimanagerista asennetun jälkikäsittelyn työkalun eri tehosteisiin tutustuttiin pintapuolisesti niiden käyttötarkoituksia, ominaisuuksia, säätimiä ja rajoitteita tarkastellessa.

Omat jälkikäsittelyn tehosteet toteutettiin käyttäen HLSL-ohjelmointikieltä Unity-pelimoottorin varjostimia hyödyntäen ja noudattaen referenssien kautta saatuja ohjelmointivinkkejä. Tehosteiden luonnin jälkeen ne yhdistettiin työkaluun, joka toimi pelikameraan ohjelmoidun skriptin avulla ja tarjosi käyttäjälle helppokäyttöisen käyttöliittymän tehosteita käyttäessä. Työkalu toteutettiin käyttäen Unity-pelimoottorin tarjoamaa C#-ohjelmointikieltä ja sen eri kehityskirjastoja.

Omat jälkikäsittelyn tehosteet ja niille luotu työkalu täyttivät työlle asetetut tavoitteet. Työkalun avulla käyttäjä pystyy yksinkertaisesti siirtämään työkalu-skriptin oman peliprojektinsa kameraan ja tehosteet ovat käytettävissä välittömästi.

Avainsanat	pelit, jälkikäsittely, Unity, pelimoottori, pelikehitys

Author Title Number of Pages Date	Miikka Vainikainen Post-processing in game development 48 pages 7th May 2020
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Technology
Professional Major	Game Applications
Instructor	Antti Laiho, Senior Lecturer

The goal of the Bachelor's thesis was to get acquainted with the installation of Unity game engine's post-processing tools, various rendering paths and the features, limitations and settings of the installed post-processing tools. After reviewing the various effects of the post-processing tool provided by the Unity game engine, it was intended to create your own post-processing effects using the HLSL programming language, creating three different shaders and a tool using the C # programming language to make the effects available to the user.

In this project, the step-by-step installation of the post-processing tool provided by the Unity game engine package manager was introduced using observational images and tips. After installing the post-processing tools, the choice of different rendering paths and its importance when starting a personal game project were introduced. The various effects of the post-processing tool installed from the Unity game engine package manager were superficially examined when considering their uses, features, controls, and limitations.

Self-made post-processing effects were implemented using the HLSL programming language, utilizing the shaders of the Unity game engine and following the programming tips obtained through the references. After creating the effects, they were combined with a tool that worked with a script programmed into the game camera, providing the user with an easy-to-use interface when applying effects. The tool was implemented using the C# programming language provided by the Unity game engine and its various development libraries.

Self-made post-processing effects and the tool created for them meet the goals set for the thesis. The tool allows the user to simply transfer the tool script to the camera of their own game project and the effects are available immediately.

Keywords	games, post-processing, Unity, game development

Sisällys

Lyhenteet

HLSL	High-Level Shading Language. Korkean tason varjostinkieli.
EXR	Open source high dynamic range imaging image file. Avoimen lähdekoodin laajan dynaamisen alueen kuvaformaatti.
FFXA	Fast Approximate Anti Aliasing. Nopean arvion reunanpehmennys.
SMAA	Subpixel Morphological Anti Aliasing. Alipikselin morfologinen reunanpehmennys.
TAA	Temporal Anti-aliasing. Väliaikainen reunanpehmennys.
SAO	Scalable Ambient Obscurance. Skaalattava ympäristön okluusio.
MSVO	Multi Scale Volumetric Obscurance. Moniluokkainen volumetrinen okluusio.
HDR	High Dynamic Range. Korkea dynaaminen alue.
LDR	Low Dynamic Range. Matala dynaaminen alue.
3D LUT	3D Lookup table. Kolmiulotteinen värien laskutaulukko.

1 Johdanto

Insinöörityössä perehdytään Unity-pelimoottorissa jälkikäsittelyn työkalujen asennukseen henkilökohtaiseen peliprojektiin. Työssä käydään läpi myös oman peliprojektin alussa välttämättömän renderöintipolun valinta ja tutustutaan kaikkiin Unity-pelimoottorin tarjoamiin renderöintipolkuihin ja väriavaruuksiin. Työssä tutustutaan pintapuolisesti Unity-pelimoottorin pakettimanagerista saatavaan jälkikäsittelyn työkaluun ja sen tarjoamiin eri tehosteisiin ja ominaisuuksiin. Työkaluja tarkastellessa käydään läpi päällisin puolin niiden käyttötarkoitus, eri tehosteiden säätimet, mahdolliset rajoitteet eri pelialustoille peliprojektia rakentaessa ja tarvittavat asetukset eri tehosteiden käyttöönoton mahdollistamiseksi.

Työkalujen asennuksen, renderöintipolkujen tarkastelun ja pakettimanagerista saadun jälkikäsittelyn työkalujen ominaisuuksiin tutustumisen jälkeen työssä luodaan oma peliprojekti käyttäen Unity-pelimoottoria. Tähän peliprojektiin tuodaan valmiiksi rakennettu 3D-kaupunki. Kaupunkiin ohjelmoidaan liikutettava hahmo, jossa peliprojektin kamera simuloi pään liikettä. Tämän lisäksi peliprojektin kameraan ohjelmoidaan skripti käyttäen Unity-pelimoottorin käyttämää C#-ohjelmointikieltä, joka mahdollistaa omien jälkikäsittelyn tehosteiden käyttämisen pelikuvassa. Jälkikäsittelyn tehosteet luodaan käyttäen Unity-pelimoottorin varjostimia ja varjostimissa käytettävää HLSL-ohjelmointikieltä.

Omia jälkikäsittelyn tehosteita tehdään kolme erilaista. Ensimmäinen on roisketehoste, joka lisää pelikuvan päälle veri- tai myrkkytehosteen. Toinen on vääristymätehoste, joka liikuttaa pelikuvan pikseleitä luoden kuvan vääristymän. Kolmas ja viimeinen tehoste on pikselitehoste, joka tekee pelikuvasta pikselimäisen.

Luodut tehosteet pyritään yhdistämään yhdeksi kokonaiseksi työkaluksi. Tämä työkalu luodaan peliprojektin kameraan luodun skriptin sisään. Työkalu tuo kaikki tehosteissa olevat säätimet ja ominaisuudet käyttäjän saataville pelikameran valvontaikkunasta käsin, jotta työkalun käyttöliittymä olisi helppokäyttöinen.

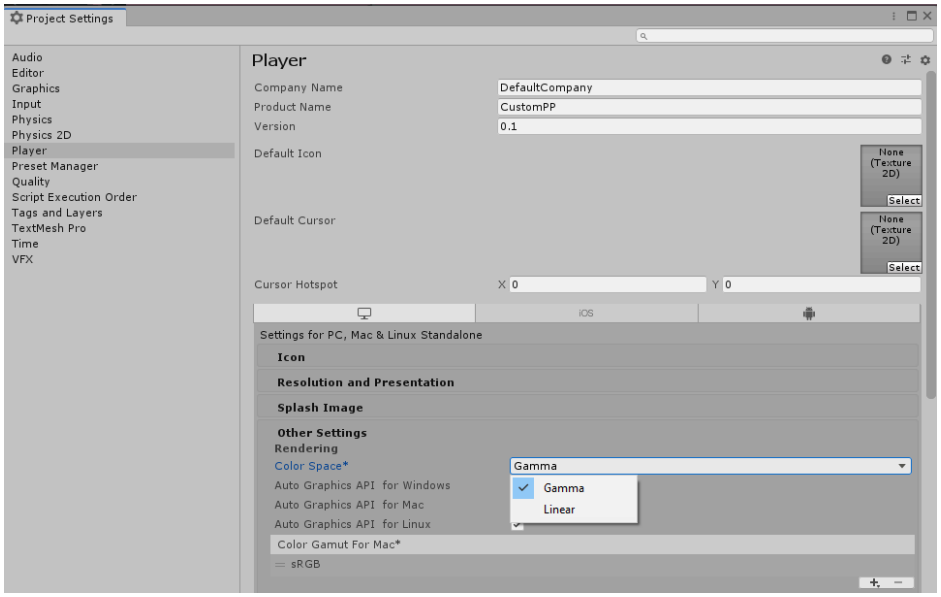
Työ on teknisesti haastava, joten insinöörityön lukijalta edellytetään Unity-pelimoottorin perusominaisuuksien osaamista ja C#- sekä HLSL-varjostinkielten perusteiden tietämystä.

2 Väriavaruus, renderöintipolut ja jälkikäsittelyn työkalut

Unity-pelimoottori tarjoaa laajan valikoiman erilaisia jälkikäsittelyn työkaluja, mutta ennen niiden käyttöä on peliprojektin alussa pohdittava väriavaruuden valintaa. Pelin väriavaruuden valinta on isossa osassa sitä, miten valo käyttäytyy peliprojektin objektien kanssa. Toinen pohdinnan aihe on, minkälaista renderöintipolkua haluaa peliprojektissa käyttää. Renderöintipolun valinta vaikuttaa siihen, miten pelimoottori renderöi pelikuvan valaistusta ja varjoja sekä mitä jälkikäsittelyn työkaluja on saatavilla omassa peliprojektissa. Renderöintipolun valinnassa on hyvä ottaa huomioon myös, mille alustalle on peliä työstämässä, koska oikean renderöintipolun valinta vaikuttaa vahvasti pelin visuaaliseen ilmeeseen ja eri alustojen vaatimuksiin sekä rajoituksiin. (3.)

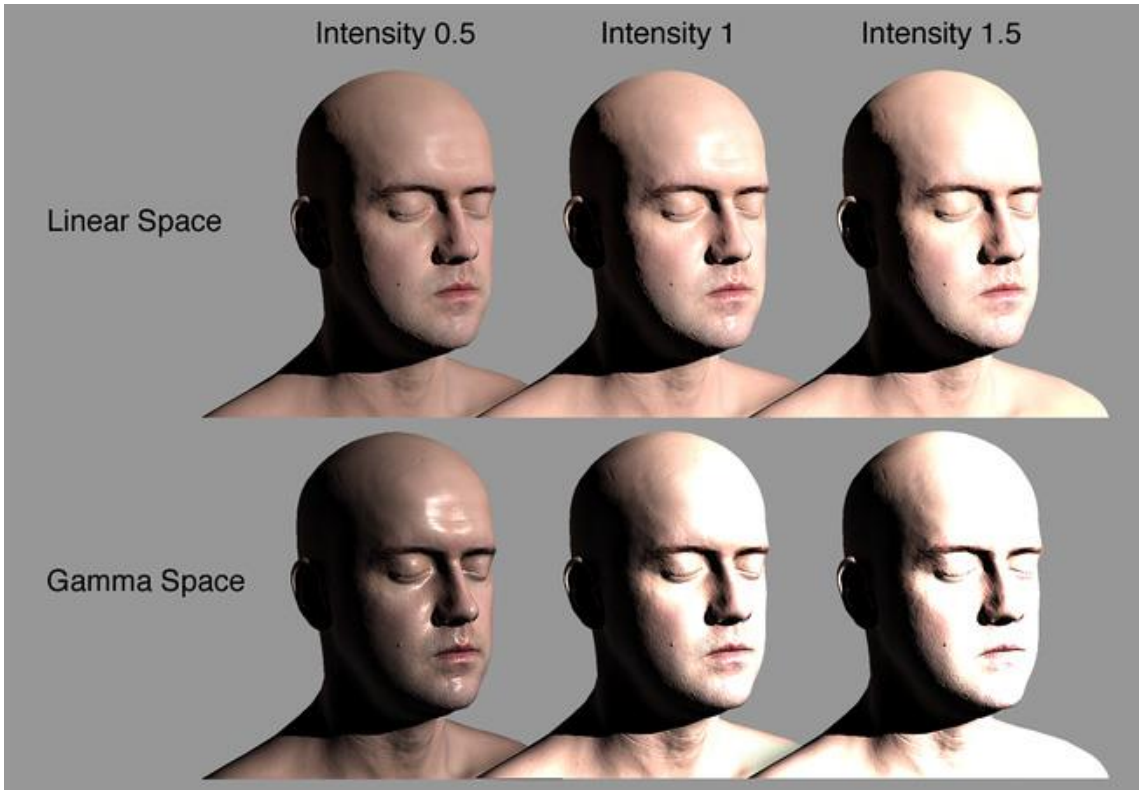
2.1 Väriavaruus

Unity-pelimoottori antaa mahdollisuuden työskennellä joko lineaarisessa tai gammaväriavaruudessa. Gammaväriavaruus on Unityssa peliprojektia luodessa oletuksena valittuna, mutta ennen peliprojektin aloittamista on hyvä pohtia, kumpi väriavaruus sopii omaan peliin parhaiten, koska esimerkiksi lineaarisella väriavaruudella renderöity pelikuva antaa tarkempia tuloksia. Kuvassa 1 nähdään lineaarisen ja gammaväriavaruuden työkaluikkuna, joka löytyy pelimoottorin yläpalkin valikosta *Edit* > *Project Settings* > *Player. (23.)*



Kuva 1. Lineaarisen- ja gammaväriavaruuden vaihdon työkaluikkuna (24).

Yleisesti pelin tekstuurit tallennetaan gammaväriavaruuteen, mutta varjostimet olettavat lineaarista väriavaruutta. Tämä johtaa epätarkkoihin tuloksiin, kun tekstuurit käyttävät varjostimia väriavaruuden laskennassa. Epätarkkojen tulosten korjaamiseksi suositellaan vaihtamaan lineaarisen väriavaruuteen, jotta varjostimet tuottaisivat virheettömiä tuloksia väriavaruuden laskennassa. Lineaarisen ja gammaväriavaruuden erovaisuudet ovat havaittavissa kuvassa 2. (23.)

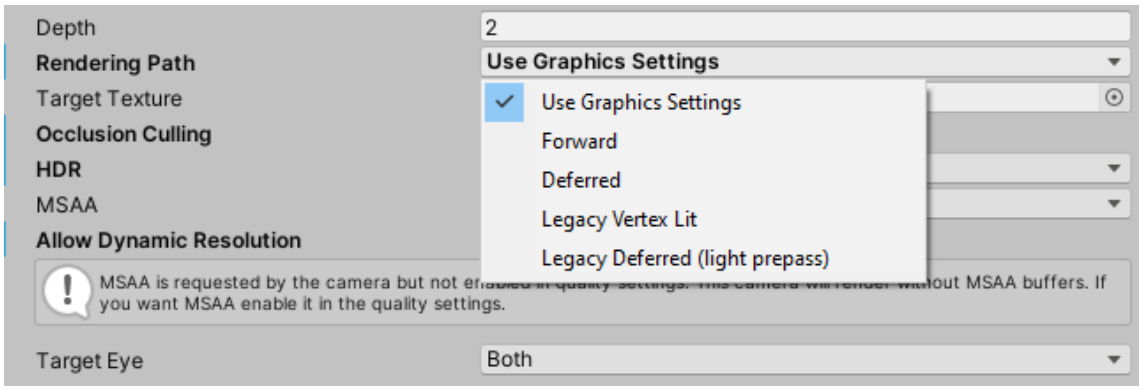


Kuva 2. Väriavaruuksien valoisuuden erot peliobjektissa (23).

Gammaväriavaruutta käytettäessä varjostimissa olevat värit ja tekstuurit on jo valmiiksi korjattu gammaväriavaruuden laskelmien mukaisesti. Kun näitä värejä ja tekstuureita käytetään varjostimissa, värin kirkkaus on suurempi kuin pitäisi, jos sitä verrataan lineaariseen väriavaruuteen. Gammaväriavaruutta käytettäessä valon kirkkaus kasvaa objektin pinnalla epälineaarisesti, mikä voi johtaa liian kirkkaaseen valotukseen objektin päällä. Lineaarisessa väriavaruudessa valon vaste objektin pinnalla pysyy lineaarisena, vaikka valon voimakkuus kasvaisikin. Tämä mahdollistaa realistisemman pinnan varjostumisen ja paremman värivasteen objektin pinnassa. (23.)

2.2 Sisäänrakennettu renderöintipolku

Sisäänrakennettu renderöintipolku (Built-in Render Pipeline) on Unity-pelimoottorin yleiskäyttöön tarkoitettu polku. Se on oletuksena käytössä oleva polku, jos peliprojektin luo käyttäen 3D- tai 2D-mallipohjaa. Se tarjoaa valaistuksen ja varjostuksen renderöintitapoja eri ominaisuuksineen, ja niistä käydään läpi kaksi yleisintä, jotka ovat suora ja viivästetty renderöinti. Renderöintitavan muuttaminen on mahdollista kuvassa 3 esitetyn esimerkin mukaisesti pelikameran näkymässä olevan työkaluikkunan kautta.



Kuva 3. Renderöintitavan valinta pelikameran näkymässä (24).

Molemmat renderöintitavat tarjoavat eri valaistuksen ja varjostuksen ominaisuudet riippuen peliprojektin tarpeista. (4.)

Suora renderöinti

Suora renderöinti (Forward Rendering) on sisäänrakennetun renderöintipolun oletusrenderöintitapa. Reaaliaikaiset valot ovat laskennallisesti aikaa vieviä renderöidä suorassa renderöinnissä. Tämän prosessin hillitsemiseksi suorassa renderöinnissä on mahdollista muokata, kuinka monta valoa renderöidään pikselitarkkuudella missäkin ympäristössä. Jos peliprojekti ei käytä paljon reaaliaikaisia valoja tai valo itsessään ei ole suuressa osassa peliprojektia, tämä renderöintitapa on hyvä valinta. (4.)

Suorassa renderöinnissä toimii suurin osa Unity-pelimoottorin pakettimanagerista saatavilla olevista jälkikäsittelyn työkaluista, mutta jotkin tehosteet tarvitsevat toimiakseen viivästettyä renderöintiä. Näistä rajoitteista puhutaan tarkemmin luvussa 3.

Viivästetty renderöinti

Viivästetty renderöinti (Deferred Rendering) sisältää eniten valaistus- ja varjostusmahdollisuuksia kaikista sisäänrakennetuista renderöintipoluista. Viivästetty renderöinti tarvitsee toimiakseen näytönohjaimen tuen, ja se sisältää tiettyjä rajoituksia. Suurin osa näytönohjaimista, jotka on valmistettu vuoden 2006 jälkeen, tukee viivästettyä renderöintiä. Mobiiliohjelmoinnissa viivästetty renderöinti on tuettu alustoilla, joissa on vähintään OpenGL ES 3.0 -ohjelmointirajapinnan tuki. (4.)

Viivästetty renderöinti ei tue läpinäkyviä objekteja tai kameran ortografisia projektioita, toisin kuin suora renderöinti. Se ei myöskään tue reunanpehmennystä (Anti-aliasing), mutta tämä tehoste on mahdollista implementoida peliprojektiin käyttäen jälkikäsittelyn työkaluja. Viivästetyllä renderöinnillä on myös rajoitettu tuki kameran maskeille (culling mask) ja kaikki pelinäkymän objektit ottavat oletuksena aina varjon vastaan. (4.)

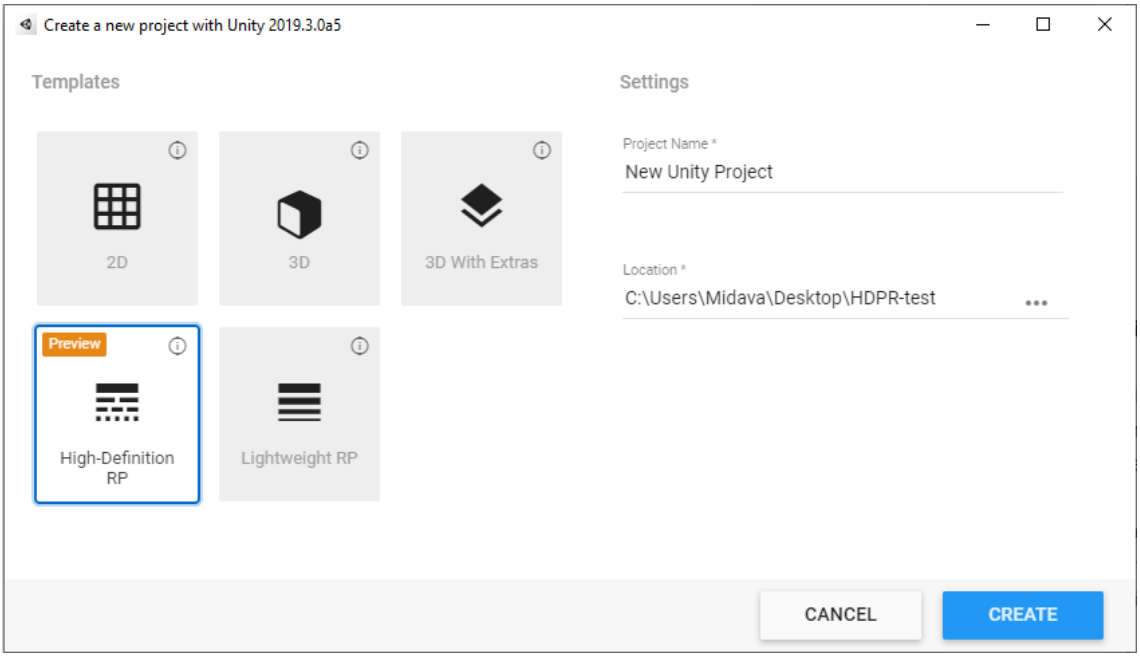
Jos peliprojekti käyttää paljon reaaliaikaisia valoja ja vaatii yleisesti korkeatasoista valaistusta ja peliprojektin kohdelaitteisto tukee viivästettyä renderöintiä, tämä renderöintitapa on oiva valinta. (4.) Viivästetyn renderöinnin käyttö sisältää enemmän jälkikäsittelyn tehostemahdollisuuksia, josta kerrotaan tarkemmin luvussa 3.

2.3 Ohjelmoitava renderöintipolku

Ohjelmoitava renderöintipolku (Scriptable Render Pipeline) antaa käyttäjälle täyden hallinnan Unity-pelimoottorin renderöintipolkuun ja tarjoaa työkalut, joilla on mahdollista luoda erittäin tarkkaa ja modernia grafiikkaa. Se antaa käyttäjälle mahdollisuuden käyttää C#-ohjelmointikieltä hallitakseen kaikkea, mitä kamera renderöi pelikuvassa. Tämä tekee Unity-pelimoottorin renderöinnistä muokattavan ja antaa näin käyttäjälle hallinnan tehdä omaan peliprojektiin tarvittavat toimenpiteet juuri oikean renderöinnin saavuttamiseksi. (7.)

Unity tarjoaa oman renderöintipolun ohjelmoinnin tueksi ohjelmointirajapinnan (SRP Core), jota hyödyntämällä käyttäjä pääsee alkuun oman renderöintipolun ohjelmoimisessa. Unity tarjoaa myös kaksi valmiiksi ohjelmoitua polkua, jotka antavat pohjan oman renderöintipolun ohjelmoimisen tueksi sen sijaan, että käyttäjä lähtisi kehittämään omaa

renderöintipolkua tyhjästä. Nämä polut ovat korkean tason renderöintipolku ja universaali renderöintipolku. Näitä polkuja voi käyttää myös sellaisinaan oman peliprojektin renderöintitarpeiden muokkaamiseksi, ja ne ovat asennettavissa ennen peliprojektin luomista Unity-pelimoottorin valintaikkunasta kuvassa 4 esitetyllä tavalla. (7.)



Kuva 4. Renderöintipolun valinta ennen peliprojektin luontia (24).

Korkean tarkkuuden renderöintipolku (High Definition Render Pipeline) on kohdistettu alustoihin, jotka käyttävät hyväkseen nykyaikaisia laskennallisia varjostimia (compute shaders), ja vaatii täten kohdealustalta yhteensopivan graafisen tuen toimiakseen. Se hyödyntää useita valaistustekniikoita ja valaistusarkkitehtuureita ja antaa työkalut, joita tarvitaan korkealla graafisella tasolla työskennellessä esimerkiksi modernien pelien, teknisten esittelyjen ja animaatioiden luomisessa. Korkean tarkkuuden renderöintipolku tukee myös suoraa ja viivästettyä renderöintiä. (6.)

Korkean tarkkuuden renderöintipolku sisältää oman implementaation jälkikäsittelyn työkaluista. Se on sisäänrakennettu renderöintipolkua käytettäessä, joten sitä ei tarvitse erikseen asentaa. Tämä implementaatio käyttää jälkikäsittelyvolyymijärjestelmää. Nämä järjestelmät ovat asennettuina, kun valitsee korkean tarkkuuden renderöintipolun ennen peliprojektin luomista. (6.)

Universaali renderöintipolku (Universal Render Pipeline) on Unityn valmistama, valmiiksi rakennettu renderöintipolku. Se tarjoaa taitelilijaystävälliseen työnkulkuun tarkoitettuja työkaluja, joiden avulla pystyy luomaan helposti ja nopeasti optimoitua grafiikkaa useille alustoille aina mobiilista konsoleihin ja tietokoneisiin. Universaalia renderöintipolkua käyttävät peliprojektit eivät ole yhteensopivia korkean tarkkuuden renderöintipolun tai sisäänrakennetun renderöintipolun kanssa. (5.)

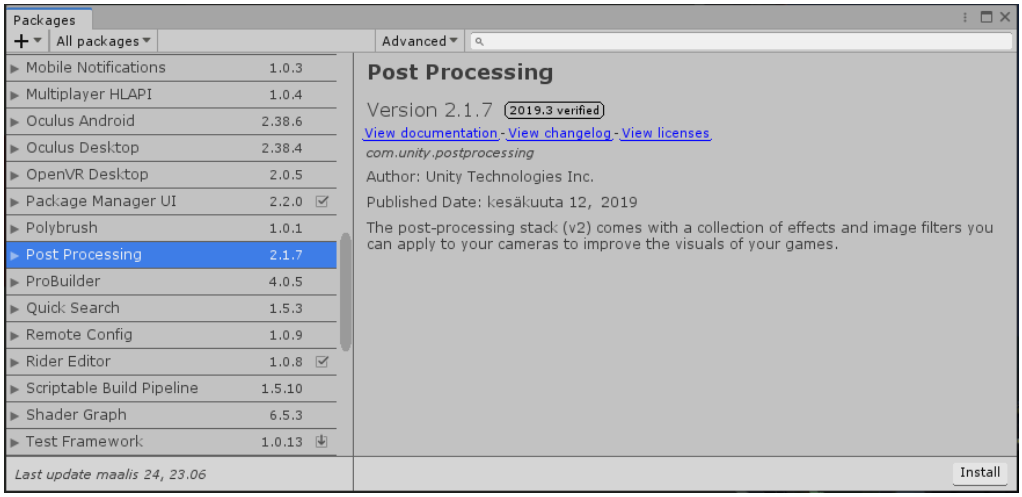
Universaali renderöintipolku sisältää omat implementaatiot jälkikäsittelyn työkaluista samalla tavalla kuin korkean tarkkuuden renderöintipolku ja käyttää myös jälkikäsittelyvolyymijärjestelmää. Universaalin renderöintipolun käyttämät jälkikäsittelyn työkalut ovat valmiiksi asennettuina, kun valitsee universaalin renderöintipolun ennen peliprojektin luomista. (5.)

2.4 Jälkikäsittelyn työkalujen asennus ja käyttöönotto

Unity-pelimoottorin jälkikäsittelyn tehosteet ovat koko ruudun suodattimia ja tehosteita, jotka suoritetaan pelimoottorissa olevan kameran kuvapuskurissa, ennen kuin pelikuva esitetään näytöllä. Jälkikäsittelyn työkaluja käytetään säännöllisesti elokuvamaisten ominaisuuksien saavuttamiseksi, koska nämä tehosteet usein simuloivat elokuvakameran ominaisuuksia, kuten alueen syvyyttä ja liikkeen sumentamista. (2.)

Jälkikäsittelytehosteiden asennus

Unity-pelimoottorin tarjoamat tehosteet ovat ladattavissa ilmaiseksi pakettimanagerista. Tehosteita ei tarvitse erikseen ladata, jos peliprojektiin on asennettu jokin Unity-pelimoottorin mukana tulevista ohjelmoitavista renderöintipoluista. Kuvassa 5 nähdään Unityn pakettimanageri ja sen sisältämä jälkikäsittelyn työkalut.

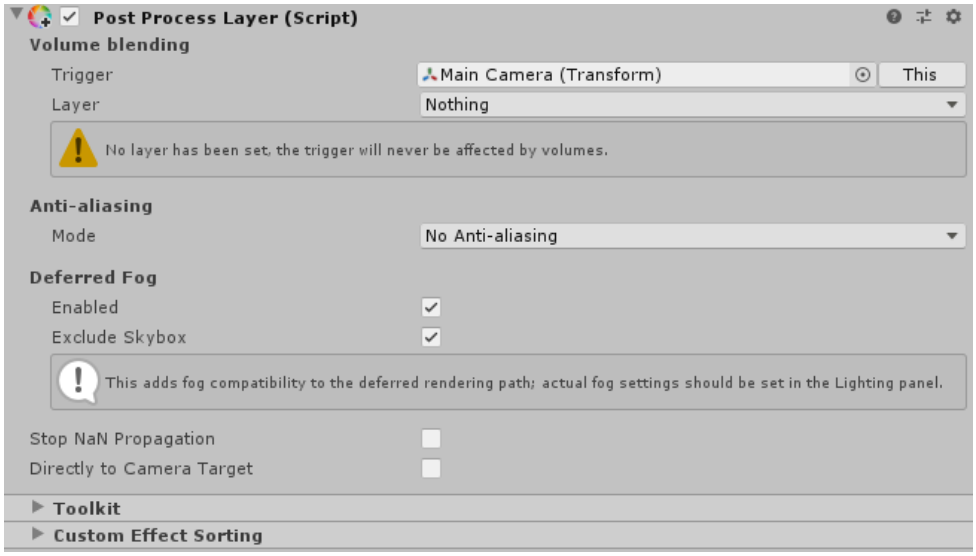


Kuva 5. Unityn pakettihallinnassa sijaitseva jälkikäsittelyn työkalut (24).

Jälkikäsittelyn työkaluihin pääsee käsiksi pelimoottorin yläpalkin valikosta *Window* > *Package Manager* (22) ja josta ne voidaan asentaa peliprojektiin.

Jälkikäsittelykerros

Jälkikäsittelyn työkalujen lataamisen jälkeen on peliprojektin sisälle luotava tiettyjä elementtejä työkalujen toiminnan takaamiseksi. Ensimmäinen askel on aktivoida pelikameran näkymässä jälkikäsittelykerros (Post Process Layer). Tämä onnistuu klikkaamalla pelikameran näkymän alapuolen komponenttivalikosta *Component* > *Rendering* > *Post-process layer* (22). Kuvassa 6 nähdään pelikamerassa aktivoitu jälkikäsittelykerros.



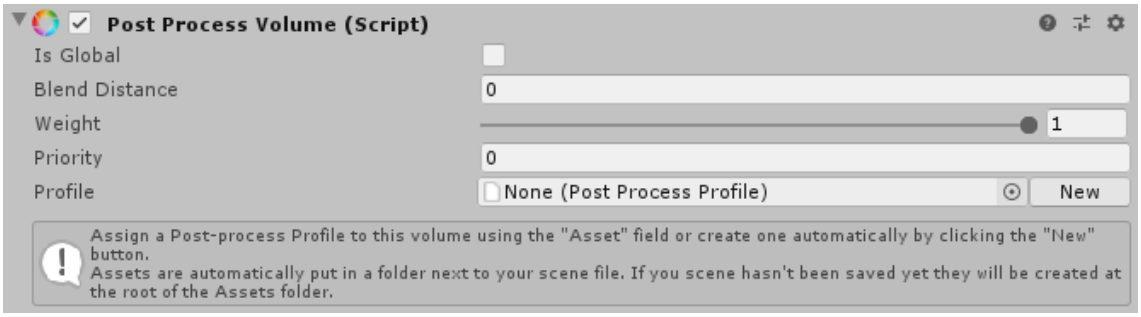
Kuva 6. Jälkikäsittelykerroksen työkaluikkuna (24).

Jälkikäsittelykerroksen työkaluikkunasta sijaitsee volyymin sekoituksen (Volume blending) kohdan alavalikosta käynnistin (Trigger), jossa on oletuksena pelikamera. Käynnistimen voi vaihtaa, jos halutaan jonkin muun objektin käynnistävän volyymin sekoituksen. Jos käynnistimien jättää tyhjäksi, globaalit volyymit toimivat edelleen. Kerros (Layer) on suositeltavaa luoda jokaiselle volyymille, koska muuten volyymeilla ei ole mitään vaikutusta. (22.)

Reunanpehmennys (Anti-aliasing) ja viivästetty sumu (Deferred Fog) käydään läpi myöhemmin tässä työssä, luvussa 3. Turhien pikseleiden pysäytys (Stop NaN Propagation) poistaa kaikki turhat pikselit ja korvaa ne mustalla värillä, ennen kuin jälkikäsittelyn tehosteet renderöidään. Työkalupakki (Toolkit) sisältää muutamia työkaluja pelinäkymän tallentamiseen EXR-tiedostomuotoon. Viimeinen osio sisältää omien tehosteiden sekoittamisen (Custom Effect Sorting), joka mahdollistaa omien tehosteiden renderöintijärjestyksen vaihtamisen. Omien jälkikäsittelyn tehosteiden luontiin tutustutaan luvussa 4. (22.)

Jälkikäsittelyvolyymi

Jälkikäsittelyvolyymin (Post Process Volume) pystyy asettamaan haluamaansa peliobjektiin, mutta yleisesti on suositeltavaa tehdä jokaiselle volyymille oma tyhjä peliobjekti, joka luodaan peliprojektin hierarkiaan. Kun tyhjä peliobjekti on luotu, jälkikäsittelyvolyymi voidaan luoda peliobjektin näkymän alapuolella lisäämällä se komponenttivalikosta *Component > Rendering > Post-processing volume.* Jälkikäsittelyvolyymi tulee asettaa samalle kerrokselle (Layer) kuin jälkikäsittelykerros, koska muuten annetut tehosteet eivät näy pelinäkymässä (22). Kuvassa 7 nähdään tyhjään peliobjektiin luotu jälkikäsittelyvolyymi.

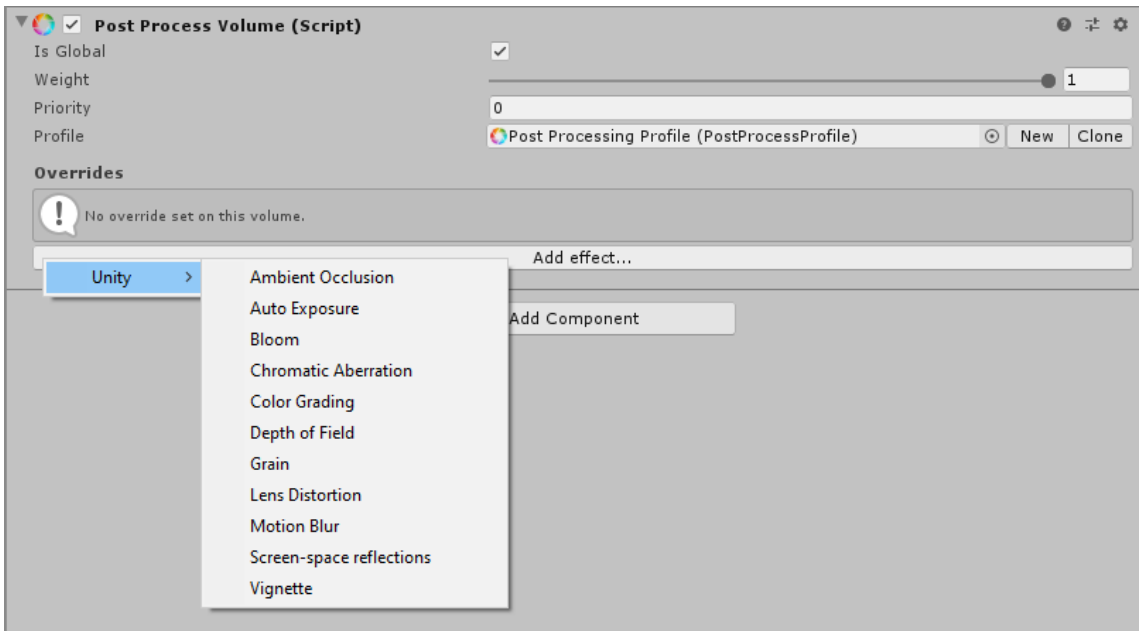


Kuva 7. Jälkikäsittelyvolyymin työkaluikkuna (24).

Jälkikäsittelyvolyymi laitetaan käyttämään joko globaalia tai paikallista asetusta. Globaali asetus vaikuttaa jatkuvasti pelinäkymään. Paikallisella asetuksella pystytään käynnistimien (Trigger) ja törmäyttimien (Collider) avulla käynnistämään volyymiin luodut jälkikäsittelyn tehosteet pelinäkymän eri kohdissa. Käyttämällä esimerkiksi törmäytintä jossain kohdassa pelinkulkua voidaan käynnistää kyseisen jälkikäsittelyn volyymin tehosteet. Sekoitusetäisyydellä (Blend Distance) voidaan vaikuttaa siihen, kuinka kaukaa kyseinen volyymi alkaa renderöidä annettuja tehosteita. Painolla (Weight) pystytään vaikuttamaan siihen, millä intensiteetillä annetut tehosteet vaikuttavat pelinäkymään. Tärkeys (Priority) antaa tietylle volyymille oman tärkeyden, jos volyymeitä on luotu monta. Mitä suurempi numero, sitä suurempi tärkeys kyseisen volyymin tehosteilla on pelinäkymään. (22.)

Jälkikäsittelyvolyymi tarvitsee tehosteiden luonnin avuksi myös jälkikäsittelyprofiilin, joka pystytään luomaan painamalla uusi (New) -nappulaa jälkikäsittelyvolyymin työkaluikkunassa. Tämä luo profiilin kyseiselle volyymille ja antaa mahdollisuuden lisätä jälkikäsittelyn tehosteita.

Jälkikäsittelyvolyymiin asetetun profiilin luonnin jälkeen jälkikäsittelyn tehosteet ovat valmiita käytettäviksi. Niitä voi lisätä jälkikäsittelyvolyymissä painamalla *Add effect…* -painiketta ja aukeavasta valikosta valitsemalla haluttu tehoste kuvassa 8 esitetyllä tavalla. Seuraavaksi työssä käydään pintapuolisesti läpi työkalussa olevia jälkikäsittelyn tehosteita.



Kuva 8. Jälkikäsittelyprofiilin asettamisen jälkeen saatavilla olevat jälkikäsittelyn tehosteet (24).

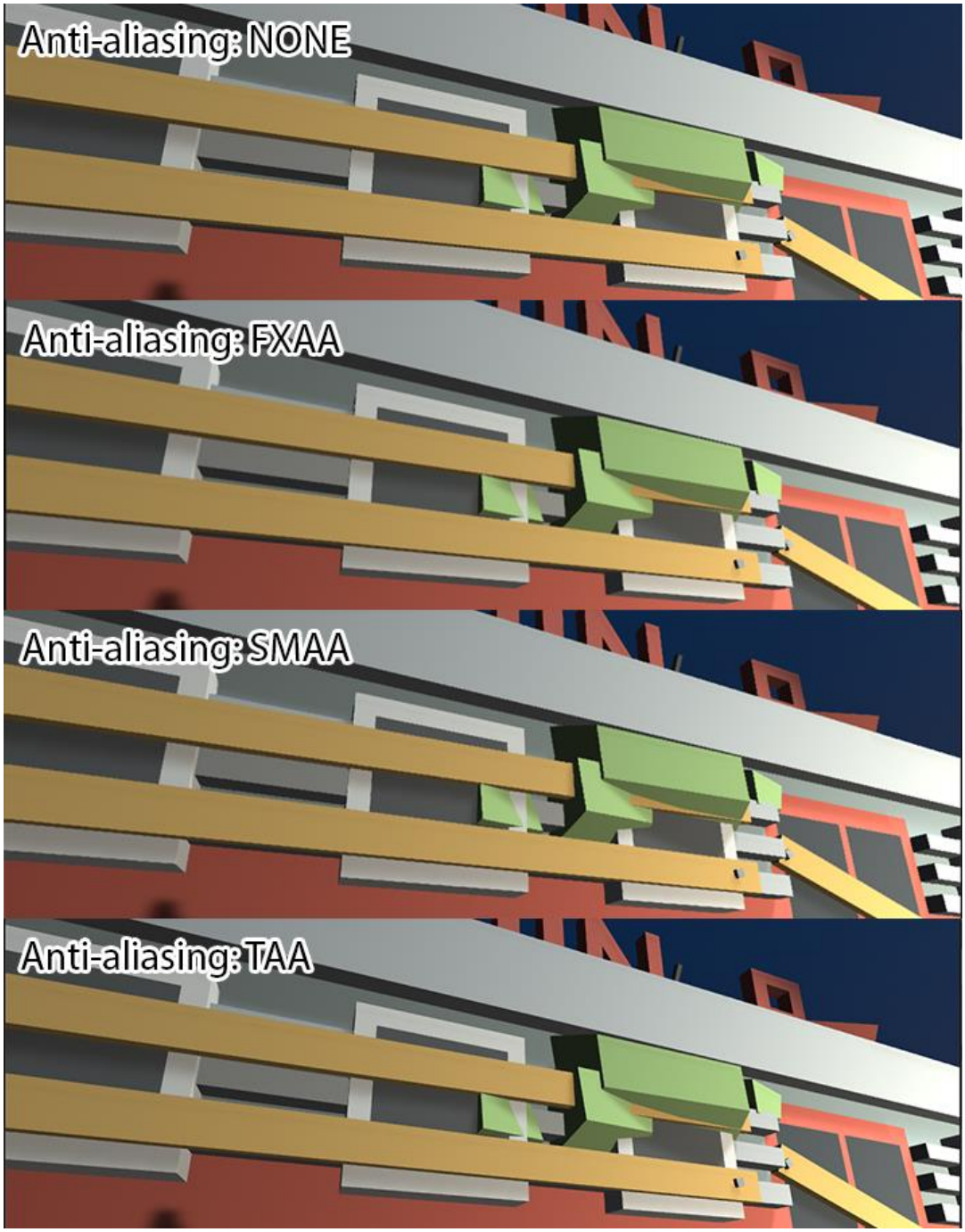
Volyymeita ja profiileita pystyy luomaan useita, jos haluaa esimerkiksi vaihdella jälkikäsittelyn tehosteita eri kohdissa pelinkulkua. Tämä mahdollistaa visuaalisesti muuttuvan graafisen kokemuksen pelaajalle riippuen pelitilanteesta. (22.)

3 Jälkikäsittelyn työkalut Unity-pelimoottorissa

Jälkikäsittelyn työkaluja ja niiden ominaisuuksia tutkiessa käytetään Unity-pelimoottorin sisäänrakennettua renderöintipolkua ja pakettimanagerin tarjoamia jälkikäsittelyn työkaluja. Työkalut käydään läpi pääpiirteittäin tarkastellen niiden ominaisuuksia ja vaatimuksia.

3.1 Reunanpehmennys

Unityn reunanpehmennystehoste (Anti-aliasing) tarjoaa kolme erilaista algoritmia, jotka on tehty estämään grafiikan reunojen porrastumista. Porrastumisella tarkoitetaan sitä, kun näytönohjaimella ei ole riittävän suurta resoluutiota suoraa linjaa varten, vaan näytölle tuleva grafiikka näyttää rosoiselta. Reunanpehmennys täyttää näiden porrastumien aukot väripaikkauksella, joka vähentää porrastumisen näkyvyyttä, mutta tekee myös reunoista epäselvempiä. (8.)

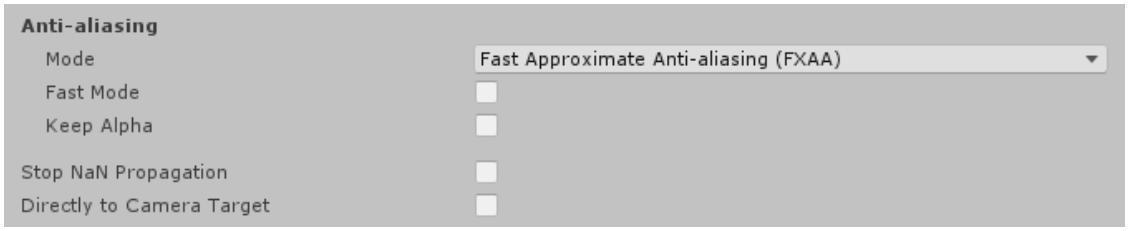


Kuva 9. Unity-pelimoottorin reunanpehmennystehosteet (24).

Kuvassa 9 on esitetty kaikki tarjolla olevat tehosteet niiden oletusasetuksia käyttämällä. Käytetyn pelinäkymän 3D-mallien reunanpehmennyksen muutos ei ole suuri, mutta vaikutus on kuitenkin minimaalisesti havaittavissa.

Nopean arvion reunanpehmennys

FFXA eli nopean arvion reunanpehmennys on laskennallisesti kevyt algoritmi, joka on suositeltu, kun alustana käytetään esimerkiksi mobiilia. Kuvassa 10 nähtävä nopea tila (Fast Mode) tulisi ottaa käyttöön mobiili- ja Nintendo Switch -alustoilla, koska se lisää huomattavasti suorituskykyä verrattuna näiden alustojen normaalitilaan. (8.)

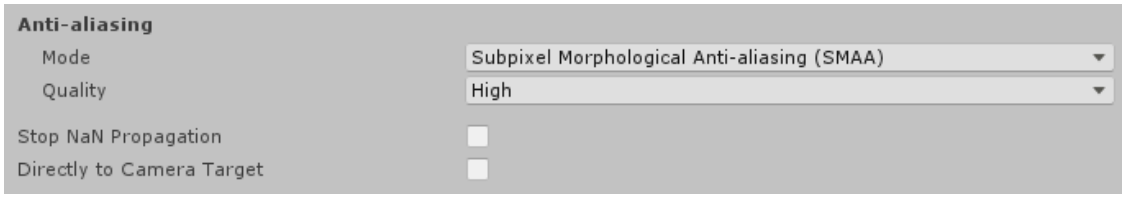


Kuva 10. Nopean arvion reunanpehmennyksen työkaluikkuna (24).

Modernit konsolit hyötyvät siitä myös vähän, mutta työpöydän näytönohjaimissa sillä ei ole eroa. Tämän reunanpehmennyksenalgoritmin käyttö vaatii tuen Shader Model 3 -versioon Unityn varjostin-ohjelmointikielestä. (8.)

Alipikselin morfologinen reunanpehmennys

SMAA eli alipikselin morfologinen reunanpehmennys on korkealaatuisempi, mutta hitaampi reunanpehmennystehoste kuin FFXA. Riippuen pelin taiteellisesta tyylistä, se voi toimia yhtä hyvin kuin TTA. Kuvassa 11 nähtävän laatu (Quality)-asetuksen vähentäminen parantaa tehosteen nopeutta.

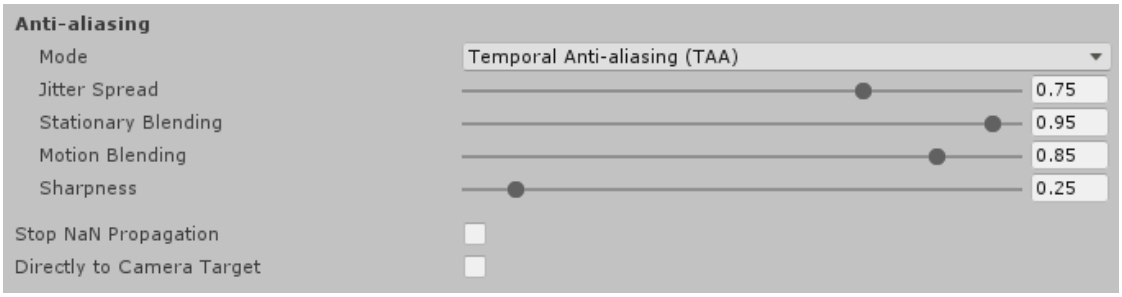


Kuva 11. Alipikselin morfologisen reunanpehmennyksen työkaluikkuna (24).

SMAA ei tue virtuaalista todellisuutta (VR) tai lisättyä todellisuutta (AR), eikä sitä suositella käytettäväksi mobiililalustoille. Tämän reunanpehmennysalgoritmin käyttö vaatii tuen Shader Model 3 -versioon Unityn varjostin-ohjelmointikielestä. (8.)

Väliaikainen reunanpehmennys

TAA eli väliaikainen reunanpehmennys on edistyneempi reunanpehmennystekniikka, jossa kehykset kerätään ajan kuluessa historiapuskuriin, jotta reunojen tasoittamisesta tulee tehokkaampaa. Tehoste tasoittaa huomattavasti paremmin liikkeessä olevia reunoja, mutta vaatii liikevektoreita, mikä tekee siitä laskennallisesti enemmän aikaa vievän kuin FXAA, minkä vuoksi sitä suositellaan käytettäväksi vain työpöytä- ja konsolialustoilla. Kuvassa 12 esitetyssä tehosteen työkaluikkunassa on useita tehosteelle ominaisia muuttujia, joita voidaan säätää halutun pelinäkymän muutoksen aikaansaamiseksi.



Kuva 12. Väliaikaisen reunanpehmennyksen työkaluikkuna (24).

TAA ei ole tuettu GLES2 renderöintialustoilla, ja se vaatii toimiakseen liikevektoreita, syvyystekstuureita ja tuen Shader Model 3 -versioon Unityn varjostin-ohjelmointikielestä. (8.)

3.2 Ympäristön okluusio

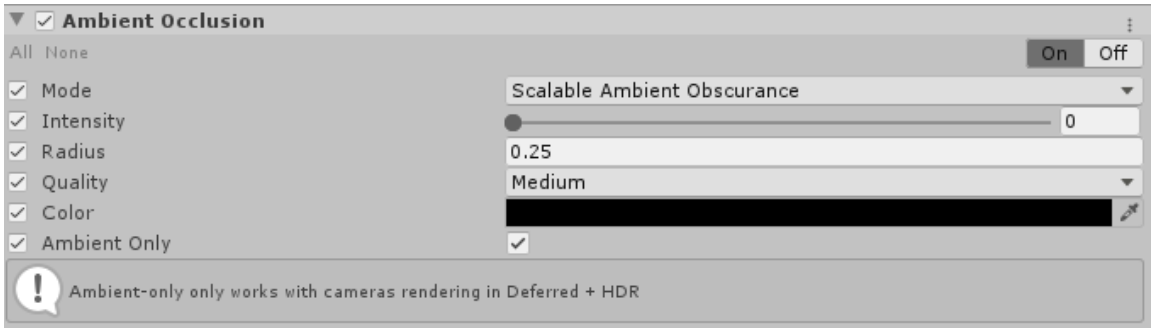
Unityn ympäristön okluusiotehoste (Ambient Occlusion) tarjoaa kaksi algoritmia, jotka arvioivat ympäristön okluusiota reaaliajassa koko näytöllä. Se tummentaa taitoksia, reikiä, risteyksiä ja pintoja, jotka ovat lähellä toisiaan. Tosielämässä tällaisilla alueilla on taipumus estää tai peittää ympäröivä valo, joten ne näyttävät tummemmilta (9). Kuvassa 13 ovat nähtävissä tehosteen muutokset eri algoritmeja käytettäessä.

Kuva 13. Tummentuneet reunat eri tehostetapoja käytettäessä (24).

Ympäristön okluusiotehosteen vaikutus on laskennallisesti aikaa vievä sen käsittelyajan takia, minkä vuoksi sitä suositellaan käytettäväksi vain työpöytä- ja konsolialustoilla. Siihen käytetty prosessointiteho riippuu puhtaasti näytön tarkkuudesta ja tehosteparametreista, eikä se riipu pelissä olevan kohdan monimutkaisuudesta, kuten todellinen ympäristön okluusio tekisi. (9.)

Skaalattava ympäristön okluusio

SAO eli skaalattavan ympäristön okluusio on standarditoteutus, joka toimii parhaiten moderneilla alustoilla. SAO on laskennallisesti aikaa vievä tehoste, varsinkin kun sitä katsotaan lähellä pelikameraa. Tästä syystä kannattaa käyttää kuvassa 14 esitettyä matala säde (Radius) -asetusta. Matalalla säteellä ympäristön okluusiovaikutus näyttää vain pikseleiltä, jotka ovat lähellä pelikameran leiketilaa, mikä on hyvä suorituskykyä varten, koska ne voidaan tallentaa välimuistiin tehokkaasti. (9.)

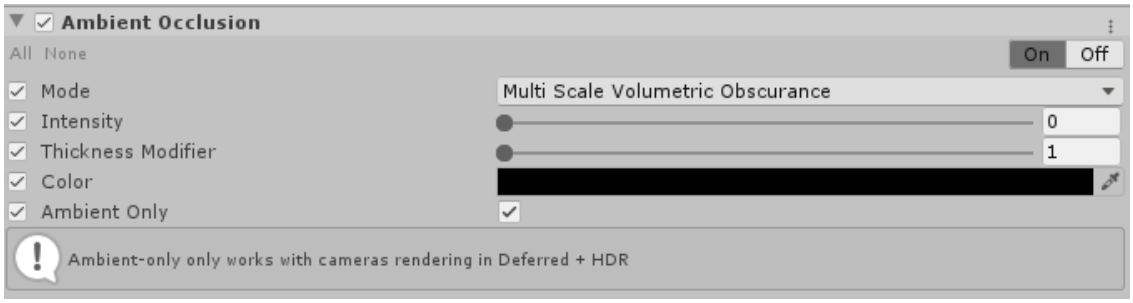


Kuva 14. Skaalattavan ympäristön okluusion työkaluikkuna (24).

Suuremmalla säteellä generoidut näytteet ovat kauempana lähdepikselistä, eikä välimuistista ole hyötyä, mikä hidastaa vaikutusta. Kameran näkökulman vuoksi lähellä olevat kohteet käyttävät suurempia säteitä kuin kaukana olevat, joten kameran lähellä olevan objektin ympäristön okluusion laskeminen on hitaampaa kuin kauempana olevan kohteen. Laadun (Quality) heikentäminen parantaa myös suorituskykyä. Yleisesti ottaen tätä vaikutusta ei kannata käyttää mobiilialustoilla ja konsolialustoilla suositellaan käyttämään MSVO:ta, koska se on paljon nopeampi ja näyttää paremmalta useimmissa tapauksissa. Tämän ympäristön okluusion algoritmin käyttö vaatii syvyys- ja normaalitekstuurit ja tuen Shader Model 3 -versioon Unityn varjostinohjelmointikielestä, ja jotkin ominaisuuksista toimivat vain viivästetyn renderöinnin tilassa. (9.)

Moniluokkainen volumetrinen okluusio

MSVO eli moniluokkainen volumetrinen okluusio on moderni versio ympäristön okluusiotehosteesta, joka on optimoitu käytettäväksi työpöytä- ja konsolialustoilla. Yleisesti se näyttää paremmalta ja toimii nopeammin kuin muut ympäristön okluusion tehosteet, mutta vaatii laskennallisen varjostimen ja Shader Model 4.5 -tuen (9). Kuvassa 15 nähtävässä työkaluikkunassa on neljä muokattavaa asetusta.



Kuva 15. Moniluokkaisen volumetrisen okluusion työkaluikkuna (24).

Intensiteetti (Intensity) muokkaa varjostuksen voimakkuutta. Tiheyden modifikaattori (Thickness Modifier) muuttaa säädettäessä varjostuksen tiheyttä ja potentiaalisesti myös aiheuttaa kohteessa mustia kehiä. Väriä (Color) muuttamalla saadaan okluusiotehosteen väriä muutettua oletuksena olevasta mustasta. Jos vain ympäristö (Ambient Only) valinta on päällä, okluusiotehoste vaikuttaa vain ympäristön luomiin valoihin. Tämä valinta toimii vain viivästetyssä (deferred) ja korkea dynaamisessa renderöinnissä (HDR).

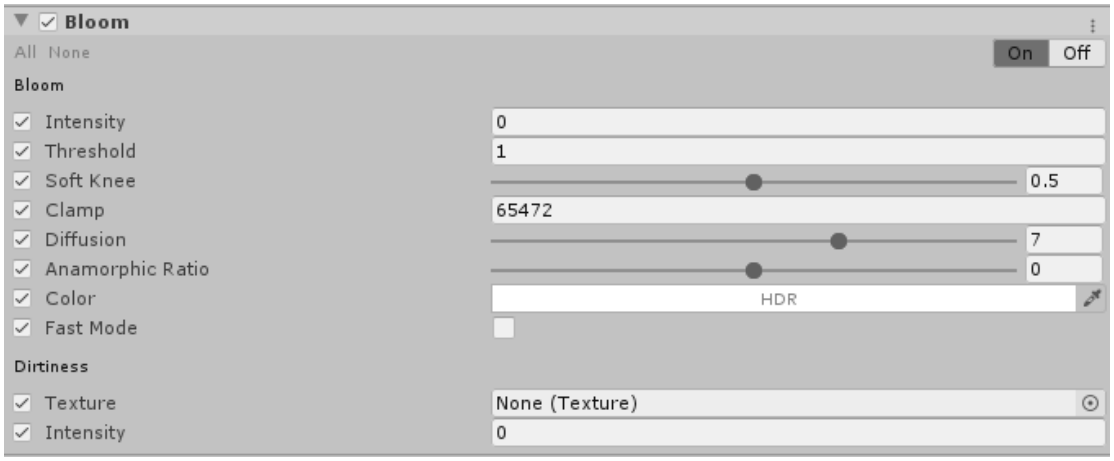
3.3 Kukoistus

Kukoistustehostetta (Bloom) käytetään tuottamaan valoa, joka ulottuu objektin kirkkaiden alueiden reunoista. Kuvassa 16 on nähtävissä, että kun intensiteettiä (Intensity) nostaa, objektin kirkkaat reunat muuttuvat vieläkin kirkkaimmiksi. Kukoistuksen lisäksi tehoste sisältää likaisuus (Dirtiness) -tehosteen, jolla saadaan aikaseksi esimerkiksi tahroja tai pölyä kukoistustehosteen diffraktoimiseksi.

Kuva 16. Kukoistuksen luoma kirkkauden muutos (24).

Likaisuustehostetta käytetään usein ensimmäisen persoonan ammuskelupeleissä. Oikein valotettujen HDR-pelikohtausten tapauksessa kynnysarvo tulisi asettaa arvoon ~ 1, jotta vain pikselit, joiden arvo on yli 1, vuotavat ympäröiviin kohteisiin. LDR:ssä työskenteltäessä arvoa pitää laskea, että vaikutus on havaittavissa.

Kuvassa 17 nähtävä diffuusio (Diffusion) -parametrin laskeminen nopeuttaa kukoistuksen vaikutusta. Mitä kauempana anamorfinen suhde (Anamorphic Ratio) on nollasta, sitä hitaampi se on.

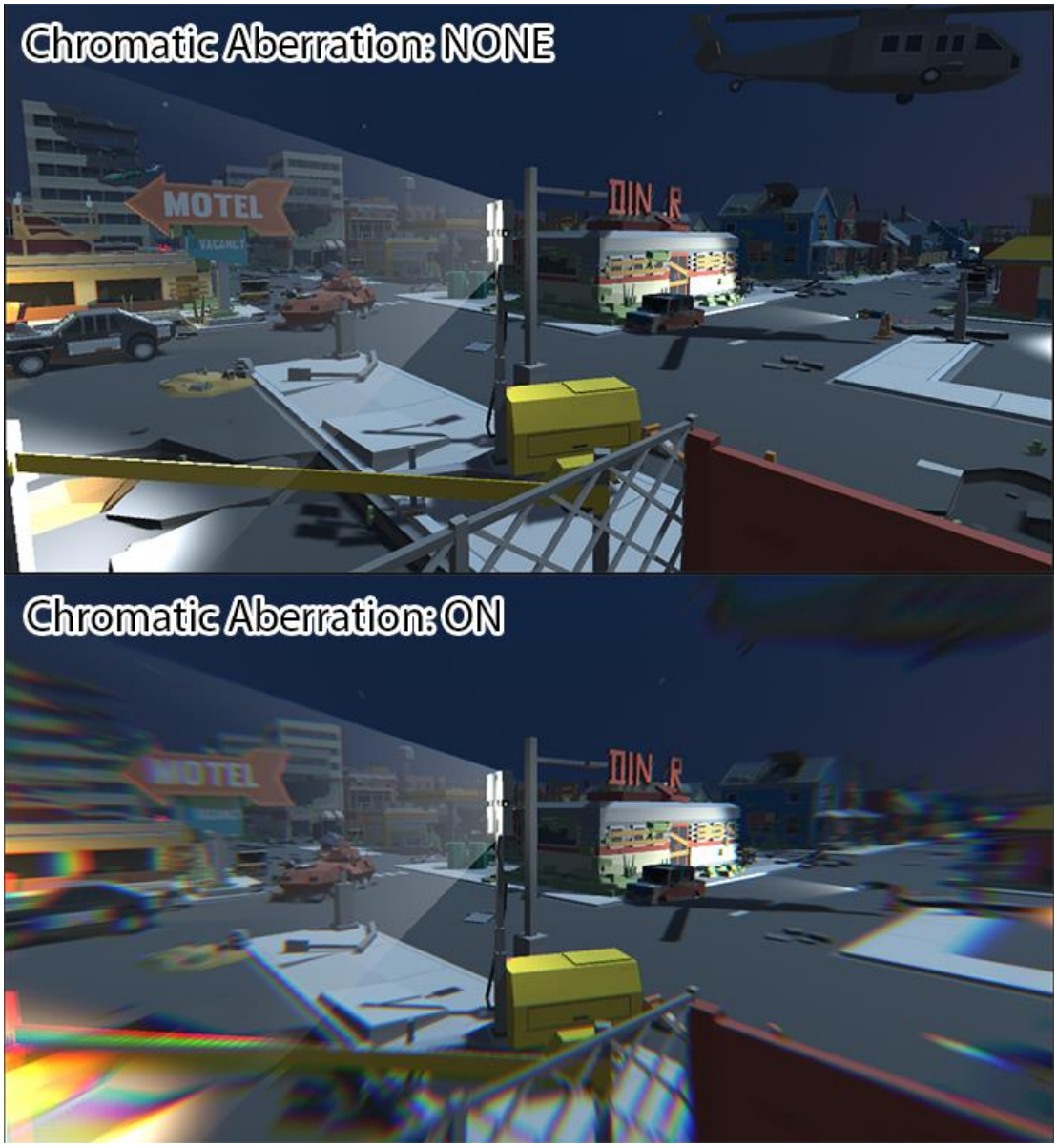


Kuva 17. Kukoistuksen työkaluikkuna (24).

Mobiili- ja muilla samankaltaisilla alustoilla on suositeltavaa käyttää nopea tila (Fast Mode) -asetusta, koska se lisää merkittävästi suorituskykyä. Kukoistustehosteen käyttö vaatii tuen Shader Model 3 -versioon Unityn varjostin-ohjelmointikielestä. (10.)

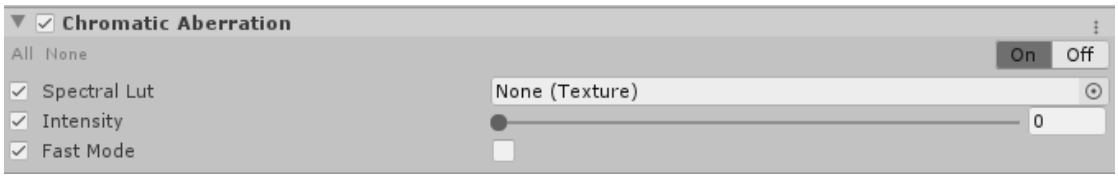
3.4 Kromaattinen poikkeama

Valokuvauksessa kromaattiset poikkeamat (Chromatic Aberration) ovat kameran linssissä tapahtuva vaikutus, jolloin linssi ei pysty yhdistämään kaikkia värejä samaan pisteeseen. Kuvassa 18 vaikutus näkyy värin reunuksina pitkin rajoja, jotka erottavat kuvan tummat ja kirkkaat osat. Unityssa kromaattista poikkeamatehostetta käytetään simuloimaan tätä valokuvauksessa kameran linssissä tapahtuvaa vaikutusta. Sitä käytetään taiteellisissa tehosteissa, kuten kameran iskuissa tai päihteiden vaikutusta jäljitellessä.



Kuva 18. Kromaattisen poikkeaman tehoste pelinäkymän reunoilla (24).

Tämä tehoste tukee punaisen, sinisen, vihreän ja violetin reunustamista sekä käyttäjän määrittelemää reunustintekstuuria käyttäen mukautettua reunustamista (Spectral Lut). Suorituskyky riippuu intensiteettiarvosta (Intensity), joka on esitetty kuvassa 19.



Kuva 19. Kromaattisen poikkeaman työkaluikkuna (24).

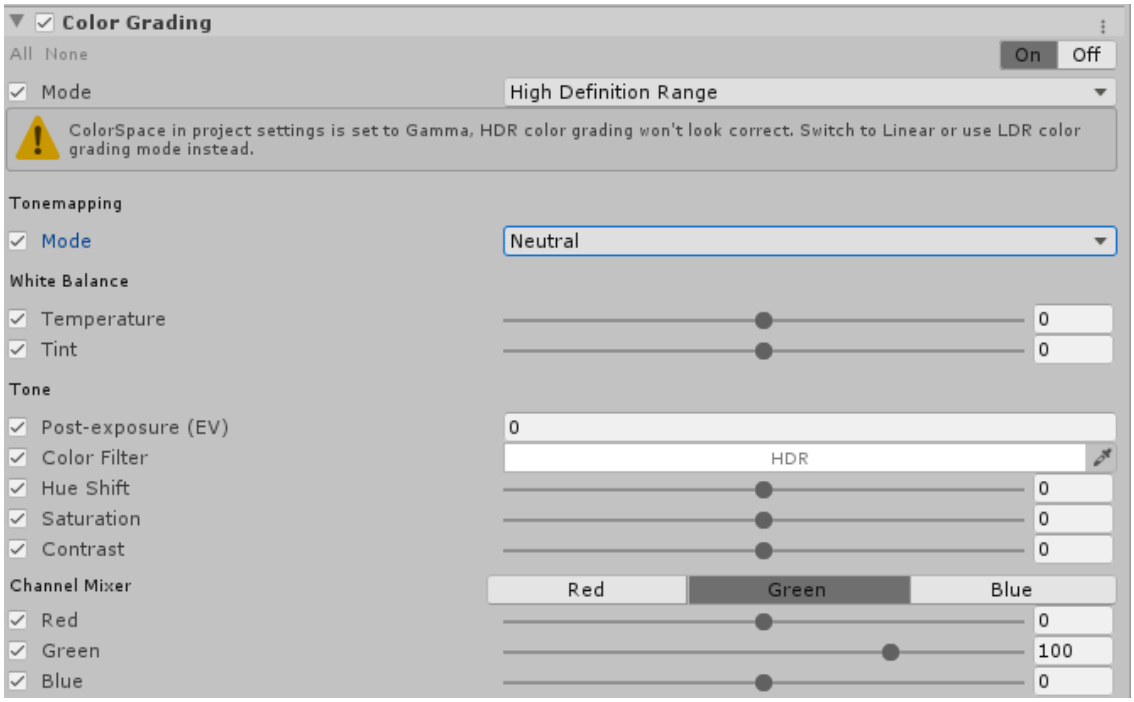
Mitä korkeampi intensiteettiarvo on, sitä hitaampi renderöinti. Pikatilan (Fast Mode) käyttö on suositeltavaa, koska se on paljon nopeampi, mutta graaffisesti tehosteen laatu on heikompi. Kromaattisen poikkeaman käyttö vaatii tuen Shader Model 3 -versioon Unityn varjostin-ohjelmointikielestä. (11.)

3.5 Väriluokittelu

Unityn väriluokittelu (Color Grading) on työkalu, jolla muutetaan tai korjataan pelikameran lopullisen kuvan värejä ja kirkkauksia. Kuvassa 20 on nähtävissä tehosteella luotu pelinäkymän värien muutos, joka antaa pelinäkymälle visuaalisesti psykedeelisen tunnelman.

Kuva 20. Väriluokittelulla tehty tehoste (24).

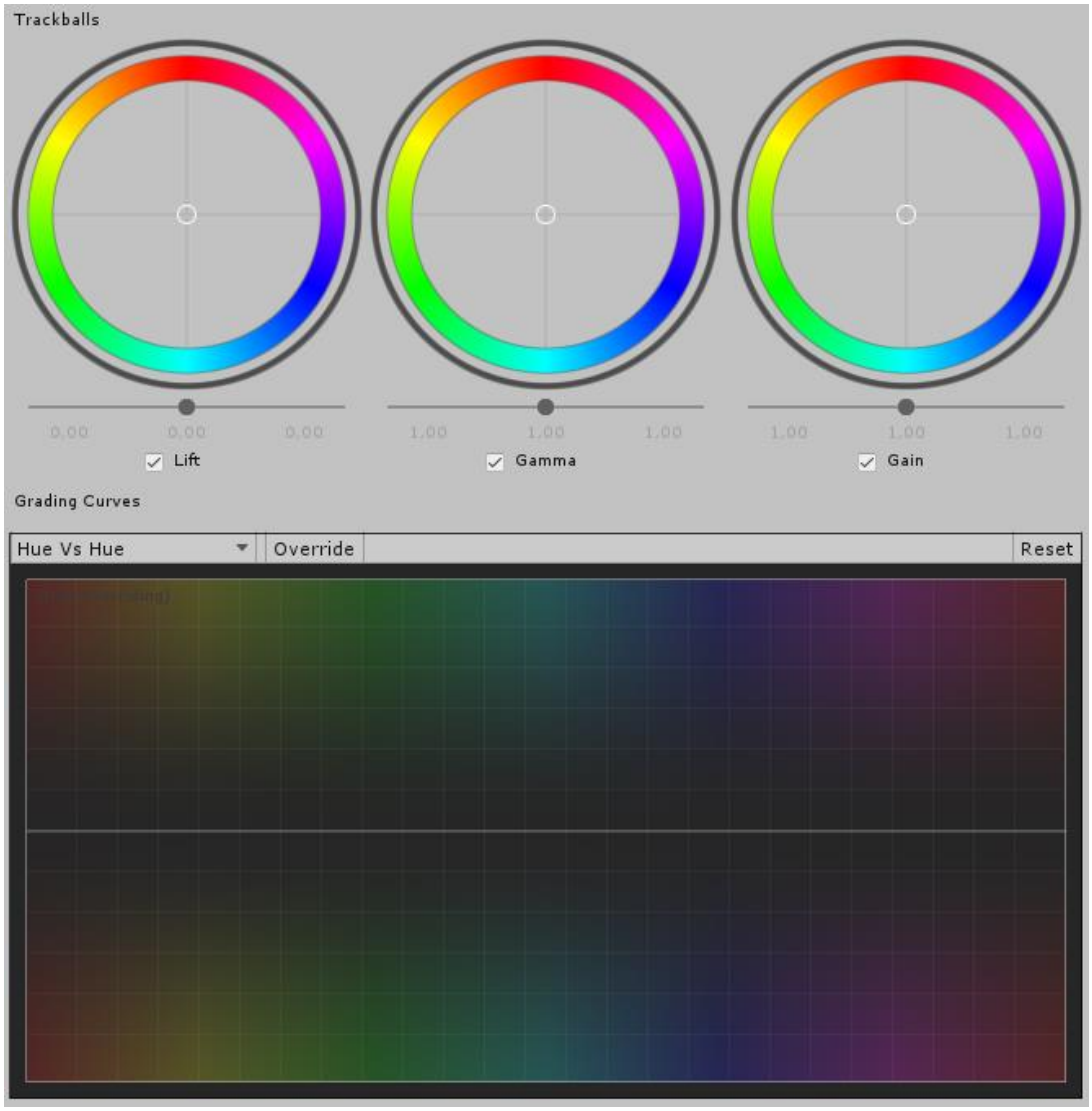
Tehosteessa on kolme erilaista tilaa, jotka ovat kuvassa 21 esitetyt matala dynaaminen alue (Low Dynamic Range), joka on tarkoitettu mobiilialustoille, mutta voidaan myös käyttää muilla alustoilla; korkea dynaaminen alue (High Dynamic Range), joka on tarkoitettu alustoille, jotka tukevat HDR-renderöintiä, ja ulkoinen (External), joka mahdollistaa räätälöityjen 3D LUT -sovellusten tuonnin ulkoisesta ohjelmistosta. (12.)



Kuva 21. Väriluokittelun tehostetyökalun osa 1 (24).

Sävykartoitus (Tone Mapping) on prosessi, jolla kuvan HDR-arvot kartoitetaan uudelleen alueelle, joka soveltuu näytettäväksi pelikuvassa. Sävykartoitusta tulisi aina käyttää HDR-kameraa käytettäessä, muuten arvot, joiden intensiteetti on yli 1, kiinnitetään pisteeseen 1, mikä muuttaa pelikuvan valon tasapainoa. Valkotasapainolla (White Balance) voi säätää pelikuvan värilämpötilaa ja valkoisen vivahdetta. Sävytehosteella (Tone) voi säätää pelikuvan värikerrointa, sävyä, kylläisyyttä, kirkkautta ja kontrastia. Kanavasekoitinta (Channel Mixer) voidaan käyttää muokkaamaan valitun värikanavan kokonaisseosta, joka on kaikkien tehosteen kolmen värin arvojen summa. (12.)

Kuvassa 22 nähtäviä väripalloja (Trackballs) käytetään kolmisuuntaiseen väriluokitteluun. Pisteen sijainnin siirtäminen yhdessä väripallossa säätää pelikuvan sävyä kohti pisteen sijainnin värin tonaalialuetta. Väripalloja ovat nosto (Lift), joka säätää tummia sävyjä tai varjoja, gamma (Gamma), joka säätää keskisävyjä, ja saanti (Gain), joka säätää korostuksia. (12.)



Kuva 22. Värilajittelun tehostetyökalun osa 2 (24).

Porrastuskäyrät (Grading Curves) on edistyksellisempi tapa säätää kuvan eri värisävyjä, kylläisyyttä tai valoisuutta. Säätämällä käyrät kahdeksassa käytettävissä olevassa kuvaajassa voi saavuttaa halutun värisävyn vaihdon vaikutukset, joidenkin valojen kylläisyyden arvon pienentämisen ja paljon muuta. (12.)

3.6 Viivästetty sumu

Unityn viivästetty sumu -työkalu (Deferred Fog) luo illuusion sumusta, joka on intensiteetiltään sitä suurempi, mitä kauempana objektit ovat kamerasta. Kuvassa 23 esitetty sumu vähentää kaukana olevien objektien värikylläisyyttä, jolla saadaan aikaseksi värimuutos ulkoympäristössä. Sumun (Fog) asetukset löytyvät valaistus (Lighting) välilehdeltä. (13.)

Kuva 23. Sumutehoste pelikuvassa luo sumun illuusion (24).

Kuvassa 24 nähtävä viivästetty sumu näkyy kameran valvontanäkymässä vain, jos pelikamera on asetettu viivästettyyn renderöintitilaan. Se on oletuksena käytössä ja antaa lisätehosteen valaistus -välilehden sumuasetuksiin. (13.)

Kuva 24. Sumun työkaluikkuna kameran valvontanäkymässä (24).

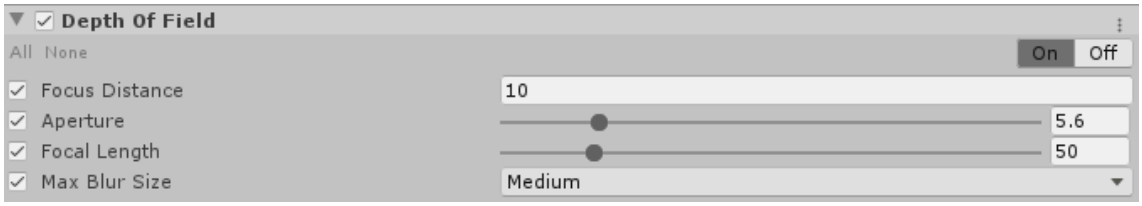
Työkaluikkuna sisältää valinnan tehosteen aktivoimiselle ja antaa myös mahdollisuuden poistaa taivaan sumentaminen, jotta sumu ei vaikuta taivaan värikylläisyyteen.

3.7 Syväterävyys

Syväterävyystehoste (Depth of Field) simuloi kameran linssin tarkennusta. Kuvassa 25 nähtävässä esimerkissä se tarkentaa pelinäkymän keskellä olevaan autoon ja sumentaa edessä ja takana olevat asiat. Tämä antaa myös visuaalisen kuvan objektin etäisyydestä.

Kuva 25. Syväterävyystehoste sumentaa kaukana tarkennuksen ulkopuolella olevat objektit (24).

Tarkennuksen etäisyyden (Focus Distance) arvo vaikuttaa siihen, miten lähelle pelinäkymää tehoste tarkentaa sumentaen muut alueet. Mitä pienempi aukkosuhteen (Aperture) tai suurempi polttovälin (Focal Length) arvo on, sitä pienempi vaikutus alueen syvyydellä on pelinäkymään. Tehosteen nopeus on sidottu kuvassa 26 esitetyn työkaluikkunan maksimiin sumennuskokoon (Max Blur Size). Jos sen arvo on yli keskikokoisen (Medium), on sitä suositeltu käytettäväksi vain pöytätietokonealustoilla ja riippuen jälkikäsittelyn budjetista myös konsolialustoilla. Mobiilialustoilla on parempi pysyä matalassa (Low) arvossa. (14.)

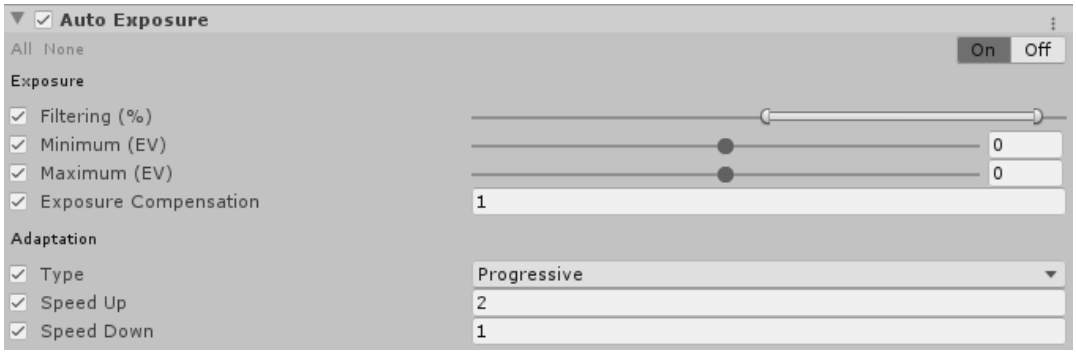


Kuva 26. Alueen syvyyden työkaluikkuna (24).

Alueen syvyystehosteen käyttö vaatii tuen Shader Model 3.5 -versioon Unityn varjostinohjelmointikielestä ja syvyystekstuurin. (14.)

3.8 Automaattinen valotus

Automaattinen valotus (Auto Exposure) -tehosteella simuloidaan silmien käyttäytymistä kirkkaasta hämärään siirtyessä ja toisinpäin. Oikeassa elämässä silmillä menee hetki tottua uuteen valotasoon. Tehoste luo muistiosion jokaiselle kuvaruudulle ja suodattaa sen keskimääräisen valotiheyden löytämiseksi. Kuvassa 27 nähtävää suodatusaluetta (Filtering) käytetään tumman ja kirkkaan alueen poissulkemiseksi.

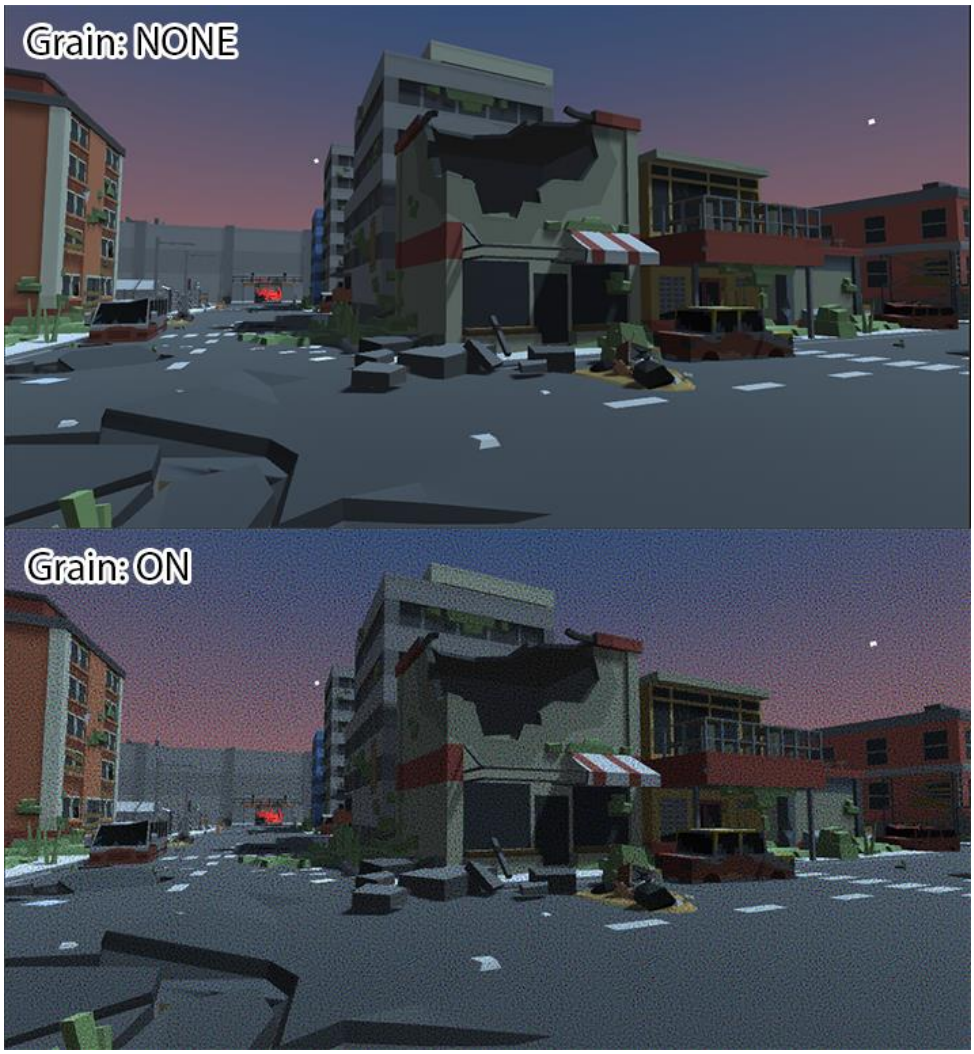


Kuva 27. Automaattisen valotuksen työkaluikkuna (24).

Keskimääräisen valotiheyden laskemiseksi kannattaa ottaa huomioon se, että yleensä ei haluta erittäin tumman ja kirkkaimman osan vaikuttavan liikaa tulokseen. Pienin / Suurin (Minimum / Maximum) -arvot kiinnittävät lasketun valotiheyden tietylle alueelle. Tehoste vaatii laskennallisen-varjostimen (compute shader) ja tuen Shader Model 5 -versioon Unityn varjostin-ohjelmointikielestä. (15.)

3.9 Rakeisuus

Rakeisuustehoste (Grain) perustuu johdonmukaiseen gradientin kohinaan. Sitä käytetään usein simuloimaan vanhojen filmikameroiden luomaa epätarkkuutta ja kohinaa, joka on nähtävissä kuvan 28 esimerkissä. Tehosteella saadaan aikaseksi epätarkkaa ja rakeista kuvaa. (16.)



Kuva 28. Rakeisuustehoste näkyy kohinan tavoin pelinäkymässä (24).

Kuvassa 29 nähtävä väritys (Colored) -valinta antaa mahdollisuuden värilliseen kohinaan, mutta tehoste toimii nopeammin, jos sen ottaa pois päältä. Intensiteettiä (intensity) säätämällä vaikutetaan tehosteen rakeisuuteen ja kokoa (Size) säätämällä rakeiden kokoon.

Kuva 29. Rakeisuuden työkaluikkuna (24).

Valotiheyden vaikutusta (Luminance Contribution) säätämällä saadaan aikaseksi pienemmillä arvoilla vähemmän kohinaa tummemmilla alueilla. Rakeisuustehosteen käyttö vaatii tuen Shader Model 3 -versioon Unityn varjostin -ohjelmointikielestä. (16.)

3.10 Liikkeen sumennus

Liikkeen sumennus (Motion Blur) on yleisesti käytetty jälkikäsittelytehoste, joka simuloi kuvan sumentumista nopeassa liikkeessä. Tehoste aktivoituu, kun pelikameralla kuvatut objektit liikkuvat nopeammin kuin pelikameran valotusaika. Kuvassa 30 esitettyä tehostetta käytetään nopeudentunnun aikaansaamiseksi ja on yleisesti käytetty esimerkiksi auto- ja kilpapeleissä.

Kuva 30. Liikkeen sumennus pelikuvassa, jossa kameraa liikutetaan akselin ympäri kovalla vauhdilla (17).

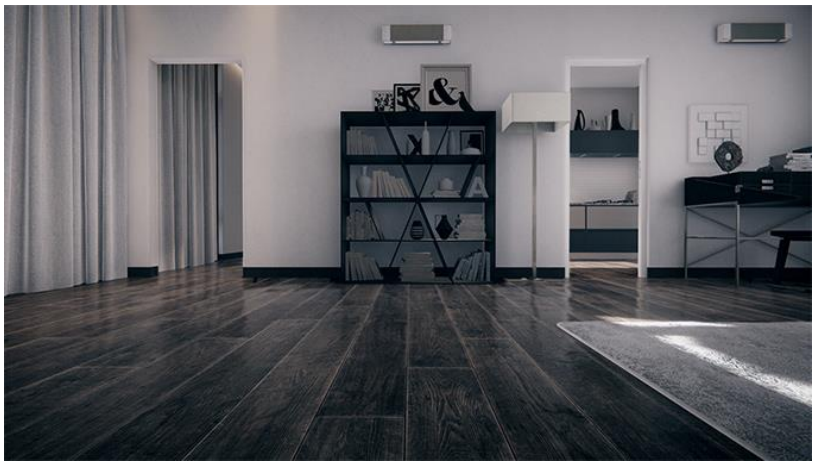
Kuvassa 31 esitettyä sulkijan kulmaa (Shutter Angle) säätämällä suuremmaksi saadaan aikaiseksi pidempikestoinen liikkeen sumennus. Näytteiden määrä (Sample Count) määrittää, kuinka monta näytettä sumennoksesta otetaan, mikä vaikuttaa laatuun ja tehokkuuteen.

Kuva 31. Liikkeen sumentamisen työkaluikkuna (24).

Pienemmän näytemäärän (Sample Count) käyttö johtaa parempaan suorituskykyyn. Tehoste ei tue Unity-pelimoottorissa ainakaan toistaiseksi virtuaalista todellisuutta (VR) tai lisättyä todellisuutta (AR) ja vaatii toimiakseen liikevektorin, syvyystekstuurin sekä tuen Shader Model 3 -versioon Unityn varjostin -ohjelmointikielestä. (17.)

3.11 Näyttötilan heijastus

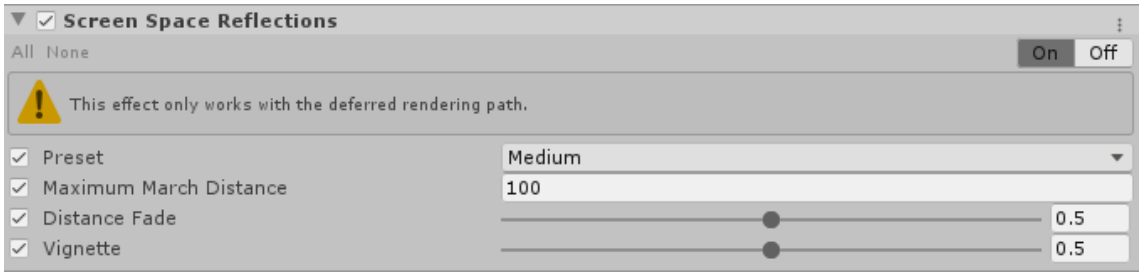
Näyttötilan heijastustehostetta (Screen Space Reflections) käytetään heijastusten aikaansaamiseksi esimerkiksi märillä pinnoilla tai lätäköissä. Tehoste toimii ainoastaan pelikameran näyttötilassa, joten se ei heijasta pintoja, jotka eivät ole sillä hetkellä näkyvillä. Oikein käytettäessä tehoste antaa visuaalisesti vaikuttavia tuloksia, mutta on laskennallisesti aikaa vievää. Kuvassa 32 on nähtävissä lattian heijastuksia kuvassa olevista objekteista.



Kuva 32. Pelikuvan lattia kiiltää näyttötilan heijastustehosteen ansiosta (18).

Tehoste toimii ainoastaan viivästetyssä renderöinnissä (Deferred Rendering), koska se ottaa tekstuurin valotusdatan käyttäen G-puskuria (G-buffer), joka kerää dataa pelinäkymän geometriasta ja on saatavilla vain kyseisessä renderöintitavassa. Nykyinen näyttötilan heijastuksen toteutus Unityssa on suunniteltu suorituskyky ennen laatua -tekniikalla, jotta se olisi käyttökelpoinen moderneissa konsoleissa ja pöytätietokoneissa.

Mukautettua (Custom) esiasetusta ei kannata käyttää kuin ottaakseen kauniisti renderöityjä kuvia, koska sen käyttö liikkuvassa pelinäkymässä vie todella paljon tehoa. Konsolialustoilla ei kannata mennä korkeammalle kuin kuvassa 33 esitettyyn keskitasoon (Medium), koska se vaatii näytönohjaimelta paljon tehoa varsinkin täysresoluutioisessa pelinäkymässä. Pienemmillä resoluutioilla esiasetettua laatua voi muuttaa saavuttaakseen samanlaisia tuloksia kuin korkeammalla resoluutiolla.



Kuva 33. Näyttötilan heijastuksen työkaluikkuna (24).

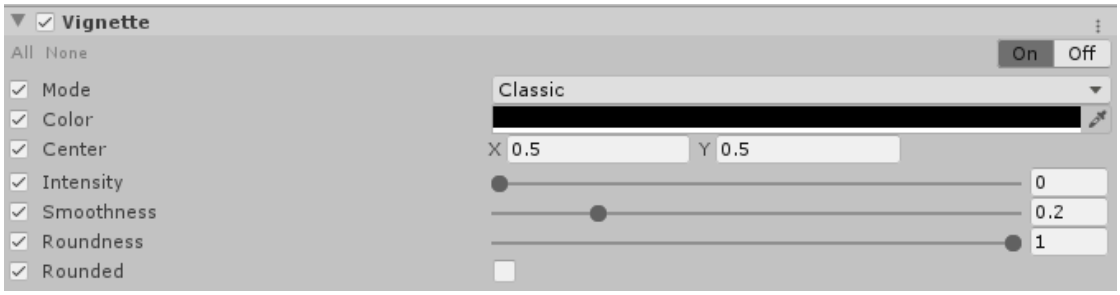
Näyttötilan heijastustehoste ei tue Unity-pelimoottorissa toistaiseksi virtuaalista todellisuutta (VR) tai lisättyä todellisuutta (AR) ja vaatii toimiakseen laskennallisen varjostimen (Compute Shader) liikevektoreita, viivästetyn renderöintipolun (Deferred Rendering Path) sekä tuen Shader Model 5 -versioon Unityn varjostin -ohjelmointikielestä. (18.)

3.12 Kehys

Kehystehosteella (Vignette) pystyy luomaan värillisiä kehyksiä pelikuvan ympärille saavuttaakseen esimerkiksi keskitetyn kuvan halutuissa peliskenaarioissa. Kuvassa 34 tehostetta on käytetty saamaan aikaan hienovarainen musta kehys pelikuvan objektien keskittämiseksi.

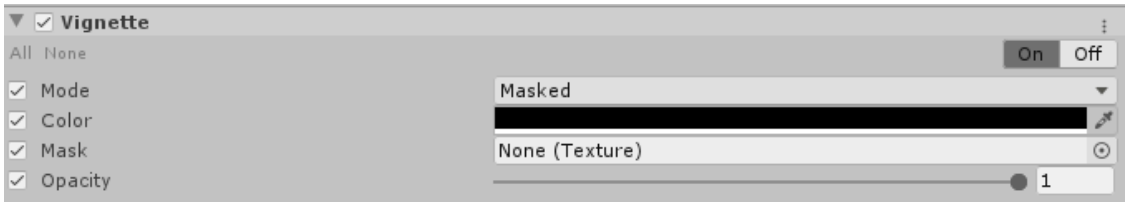
Kuva 34. Pelinäkymän reunoilla näkyvä vinjettitehoste (24).

Unity-pelimoottorissa vinjettitehosteella on kaksi tilaa: klassinen (Classic) ja naamioitu (Masked). Kuvassa 35 on nähtävissä klassisen kehyksen työkaluikkuna.



Kuva 35. Klassisen kehyksen työkaluikkuna (24).

Klassisessa kehyksessä on mahdollista muuttaa kehyksen väriä (Color), sijaintia (Center), voimakkuutta (Intensity), pehmeyttä (Smoothness) ja pyöreyttä (Roundness) (19). Kuvassa 36 on nähtävissä on toisen vinjettettitehosteen eli naamioidun kehyksen työkaluikkuna.



Kuva 36. Naamioidun kehyksen työkaluikkuna (24).

Naamioitu kehys mahdollistaa värien (Color) muokkaamisen ja mukautetun tekstuurin (Mask) käytön kehyksen päällä, millä voidaan saavuttaa uniikkeja kehystehosteita. (19.)

4 Omien jälkikäsittelytehosteiden luonti

Omien jälkikäsittelytehosteiden luontiprosessi alkoi Unity-pelimoottorin käynnistyksellä. Käynnistyksen yhteydessä luotiin uusi projekti 3D-työtilaan. Projekti luotiin käyttäen Unityn versiota 2019.3.4f1, ja siinä käytettiin sisäänrakennetun renderöintipolun oletusasetusta, eli siihen ei erikseen valittu mitään Unityn tarjoamista renderöintitavoista. Kuvassa 37 on nähtävissä peliprojektiin tuotu demokaupunki, joka sisälsi paljon erilaisia 3D-malleja ja valonlähteitä.

Kuva 37. Peliprojektiin tuotu demokaupunki (25).

Demokaupungin lisäksi pelimaailmaan ohjelmoitiin yksinkertainen hahmo, jonka avulla pystyttiin ohjaamaan kameraa ensimmäisessä persoonassa.

4.1 Tehosteiden renderöiminen pelikamerassa

Demokaupungin ja hahmon luonnin jälkeen tarkasteltiin, miten pelikamerassa saataisiin aikaan tehosteita, jotka käyttäisivät koko pelikuvaruudun pikseleitä ja jotka antaisivat käyttää normaalin pelikuvan kanssa itseohjelmoituja varjostimia jälkikäsittelytehosteen aikaansaamiseksi. Tämän mahdollisti Unityn C#-ohjelmointikielen metodi OnRenderImage (20), jota käyttämällä pystyttiin vaikuttamaan renderöidyn kuvan lähteeseen ja lopullisen pelikuvan väliin esimerkkikoodissa 1 esitetyllä tavalla, johon pystyttiin luomaan omia varjostimia pelikuvan tehostamiseksi

```
public class CustomPostProcessingEffect : MonoBehaviour
{
      public Material effectMaterial;


      void OnRenderImage (RenderTexture source, RenderTexture destination)
      {
            Graphics.Blit(source, destination, EffectMaterial);
      {
```

Esimerkkikoodi 1. Peliprojektin kameraan asetettu koodi, jossa pelikamera renderöi normaalin pelikuvan lisäksi annetun materiaalin varjostintehosteen (20).

OnRenderImage-metodi otti sisäänsä lähteen (source), joka sisälsi renderöidyn kuvan eli kaiken, mitä senhetkinen pelitilanne pitää sisällään. Metodi tarvitsi myös määränpään (destination), joka oli tässä tapauksessa kameran kohde eli pelikuva. Skriptin kirjoittamisen jälkeen se kiinnitettiin kameraan, minkä jälkeen alkoi varjostintehosteiden luontiprosessi.

4.2 Varjostintehosteiden luonti

Omien varjostintehosteiden luonti alkoi erilaisten varjostintehosteiden referenssien etsimisellä. Referenssien etsimisen jälkeen päädyttiin tekemään kolme erilaista tehostetta, jotka olivat roiske-, pikseli- ja vääristymätehoste. Tehosteiden ohjelmoinnin pohjana käytettiin Unityn valmiiksi ohjelmoitua kuvatehostevarjostinta, joka on nähtävissä esimerkkikoodissa 2.

```
Shader "Folder/Subfolder"
{
    Properties
    {
        _MainTex ("Texture", 2D) = "white" {}
    }
    SubShader
    {
        Cull Off ZWrite Off ZTest Always

        Pass
        {
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag
            #include "UnityCG.cginc"

            struct appdata
            {
                float4 vertex : POSITION;
                float2 uv : TEXCOORD0;
            };

            struct v2f
            {
                float2 uv : TEXCOORD0;
                float4 vertex : SV_POSITION;
            };

            v2f vert (appdata v)
            {
                v2f o;
                o.vertex = UnityObjectToClipPos(v.vertex);
                o.uv = v.uv;
                return o;
            }

            sampler2D _MainTex;

            fixed4 frag (v2f i) : SV_Target
            {
                fixed4 col = tex2D(_MainTex, i.uv);
                col.rgb = 1 - col.rgb;
                return col;
            }
            ENDCG
        }
    }
}
```

Esimerkkikoodi 2. Varjostintehosteiden pohjana käytetty ratkaisu.

Valmiiksi ohjelmoitu kuvatehostevarjostin oli luotavissa peliprojektin hierarkiaan projektinäkymässä painamalla hiiren oikealla painikkeella *Create* > *Shader* > *Image Effect Shader.* Tämä varjostin antoi hyvän pohjan omien jälkikäsittelytehosteiden varjostinten luontiin, koska se sisälsi verteksien ja UV-koordinaattien muokkaamiseen tarvittavat metodit valmiiksi kirjotettuina. Ilman muokkausta se käänsi pelikuvassa näkyvät värit vastakkaisiksi.

Roisketehoste

Roisketehosteen apuna käytettiin itsetehtyjä tekstuureita, jotka sisälsivät veren ja myrkyn roiskeet. Esimerkkikoodi 3:ssa esitettyyn varjostimeen luotiin päätekstuurin lisäksi sekoitustekstuuri. Päätekstuuri kerrottiin sekoitustekstuurilla, minkä avulla saatiin roisketekstuuri ilmestymään pelikuvaan. Sekoitustekstuurin lisäksi luotiin läpinäkyvyysmuuttuja, jonka avulla käyttäjän oli mahdollista säätää tekstuurin näkyvyyttä pelikuvassa.

```
Shader "CustomPP/Poisony"
{
    Properties
    {
        _MainTex("Base RPG", 2D) = "white"{}
        _BlendTex ("Blend Texture", 2D) = "white"{}
        _PoisonyOpacity ("Poisony Opacity", Range(0,1)) = 1
    }

    SubShader
    {
        Pass
        {
            CGPROGRAM
            #pragma vertex vert_img
            #pragma fragment frag
            #include "UnityCG.cginc"

            uniform sampler2D _MainTex;
            uniform sampler2D _BlendTex;
            fixed _PoisonyOpacity;

            fixed4 frag(v2f_img i) : COLOR
            {
                fixed4 mainTex = tex2D(_MainTex, i.uv);
                fixed4 blendTex = tex2D(_BlendTex, i.uv);

                fixed4 blendedMultiply = mainTex * blendTex;
                mainTex = lerp(mainTex, blendedMultiply, _PoisonyOpacity);
                return mainTex;
            }
            ENDCG
        }
    }
}
```

Esimerkkikoodi 3. Toinen roisketehosteen varjostinkoodeista, jossa päätekstuuri kerrottiin sekoitustekstuurilla myrkkyroisketehosteen aikaansaamiseksi.

Varjostin mahdollisti myös sekoitustekstuurin vaihtamisen mihin vain tekstuuriin, joka haluttiin asettaa normaalin pelinäkymän päälle.

Vääristymätehoste

Vääristymätehosteen tarkoitus oli antaa varjostimelle siirtymätekstuuri, jossa oli sekoitus mustan ja vaalean eri sävyjä. Näihin sävyihin tartuttiin koodissa siten, että sävyn eri pikseleitä siirrettiin pelikuvassa joko oikealle tai ylöspäin käyttäen kuvan UV-koordinaatteja. Esimerkkikoodissa 4 nähtävä siirtymätekstuuri käyttäytyi normaalikartan tavoin, mutta se kerrottiin varjostimeen luodulla arvolla. Tämä mahdollisti kuvan vääristymän, jota pystyttäisiin käyttämään esimerkiksi simuloimaan päihteiden vaikutuksen alaisena olevaa pelaajaa, huimausta tai vaikka kuumaa aavikkoa.

```
Shader "CustomPP/Dizzy"
{
      Properties
      {
            _MainTex("Texture", 2D) = "white" {}
            _DisplaceTex("Displacement Texture", 2D) = "white" {}
            _Effectiveness("Effectiveness", Range(0,0.1)) = 1
            _AnimationSpeed("AnimationSpeed", Range(0,4)) = 1
      }
            SubShader
            {

            Cull Off ZWrite Off ZTest Always

            Pass
            {
                  CGPROGRAM
                  #pragma vertex vert
                  #pragma fragment frag
                  #include "UnityCG.cginc"

                  struct appdata
                  {
                  float4 vertex : POSITION;
                  float2 uv : TEXCOORD0;
                  };

                  struct v2f
                  {
                  float2 uv : TEXCOORD0;
                  float4 vertex : SV_POSITION;
                  };

                  v2f vert(appdata v)
                  {
                  v2f o;
                  o.vertex = UnityObjectToClipPos(v.vertex);
                  o.uv = v.uv;
                  return o;
                  }

                  sampler2D _MainTex;
                  sampler2D _DisplaceTex;
                  float _Effectiveness;
                  float _AnimationSpeed;

                  float4 frag(v2f i) : SV_Target
                  {
```

```
                float2 distuv = float2(i.uv.x + _Time.x * _AnimationSpeed,
                i.uv.y + _Time.x * _AnimationSpeed);

                float2 disp = tex2D(_DisplaceTex, distuv).xy;
                disp = ((disp * 2) - 1) * _Effectiveness;

                float4 col = tex2D(_MainTex, i.uv + disp);
                return col;
                }
                ENDCG
                }
            }
        }
}
```

Esimerkkikoodi 4. Vääristymätehosteen varjostinkoodi, jolla pystyttiin vääristämään pelikuvaan asetetun tekstuurin pikseleitä niin, että se loi pelikuvan siirtymän.

Varjostimeen ohjelmoitiin myös tehokkuus- ja nopeusmuuttuja, joilla pystyttiin säätämään, millä voimakkuudella varjostin vaikuttaa pelikuvaan ja kuinka nopeasti se liikkuu pelikuvassa. Käyttäjän oli myös mahdollista vaihtaa olemassa oleva vääristymätekstuuri erilaisen pelikuvan siirtymän aikaansaamiseksi.

Pikselitehoste

Pikselitehosteessa varjostimeen luotiin päätekstuurin lisäksi kaksi muuttujaa, joiden tarkoitus oli antaa käyttäjälle mahdollisuus säätää, kuinka monta pikseliä renderöidään pelikuvassa horisontaalisesti ja vertikaalisesti. Muuttujien luonnin jälkeen kuvan UV-koordinaattien horisontaaliset ja vertikaaliset pikselit kerrottiin, pyöristettiin ja jaettiin näillä muuttujilla.

Pikseleiden vertikaalisia ja horisontaalisia arvoja oli mahdollista muokata arvojen 1 ja 1000 välillä, mikä mahdollisti pienellä arvolla erittäin epätarkan pelinäkymän. Arvojen suuruutta on mahdollista muokata varjostimesta käsin esimerkkikoodi 5:ssa esitetyn koodin kohdassa _Vertical("Vertical Pixels", Range(1,1000)) = 1 ja _Horizontal ("Horizontal Pixels", Range(1,1000)) = 1 muuttaen pituuden (Range) arvoja pienimmästä pikselitiheydestä suurempaan.

```
Shader "CustomPP/Pixely"
{
      Properties
      {
            _MainTex("Texture", 2D) = "white" {}
            _Vertical("Vertical Pixels", Range(1,1000)) = 1
            _Horizontal("Horizontal Pixels", Range(1,1000)) = 1
      }
            SubShader
      {
```

```
            Cull Off ZWrite Off ZTest Always

            Pass
            {
                  CGPROGRAM
                  #pragma vertex vert
                  #pragma fragment frag

                  #include "UnityCG.cginc"

                  struct appdata
                  {
                        float4 vertex : POSITION;
                        float2 uv : TEXCOORD0;
                  };

                  struct v2f
                  {
                        float2 uv : TEXCOORD0;
                        float4 vertex : SV_POSITION;
                  };

                  v2f vert(appdata v)
                  {
                        v2f o;
                        o.vertex = UnityObjectToClipPos(v.vertex);
                        o.uv = v.uv;
                        return o;
                  }

                  uniform float _Vertical;
                  uniform float _Horizontal;
                  sampler2D _MainTex;

                  float4 frag(v2f i) : SV_Target
                  {
                        float2 uv = i.uv;
                        uv.x *= _Vertical;
                        uv.y *= _Horizontal;
                        uv.x = round(uv.x);
                        uv.y = round(uv.y);
                        uv.x /= _Vertical;
                        uv.y /= _Horizontal;
                        fixed4 col = tex2D(_MainTex, uv);
                        return col;
                  }
                  ENDCG
            }
      }
}
```

Esimerkkikoodi 5. Pikselitehosteen varjostin, jonka avulla käyttäjä pääsee käsiksi pelikuvan horisontaaliseen ja vertikaaliseen pikselitiheyteen.

Näillä yksinkertaisilla laskutoimituksilla saatiin aikaseksi pikselitehoste, jonka avulla voidaan pelikuvaa muokata antamaan illuusio pikselöityneestä pelitilanteesta.

4.3 Tehostetyökalun ohjelmointi

Tehostetyökalun pohjana käytettiin kameraan luotua skriptiä. Tämä skripti sisälsi alkujaan materiaalimuuttujan ja OnRenderImage-metodin, joiden avulla pystyttiin pelikuvaa muokkaamaan materiaaliin asetetun varjostimen avulla. Tämä ratkaisu kuitenkin oli hieman kömpelö, koska jokaisen tehosteen testaukseksi ja pelikuvan muokkaamiseksi jokainen materiaali tuli vetää erikseen projektin hierarkiasta skriptiin ja varjostimen eri muuttujia säätää varjostimesta käsin.

Ensimmäiseksi skriptille annettiin ExecuteInEditMode-funktio, joka mahdollisti skriptin ominaisuuksien näkymisen pelikuvassa ilman, että peli tulisi käynnistää jokaisella testikerralla. Skriptiin luotiin yksityiset muuttujat jokaiselle materiaalille, jota pystyttiin käyttämään tehosteita kutsuttaessa skriptin ulkopuolelta. Tämän jälkeen skriptiin luotiin OnValidate-funktio, jonka sisällä tarkistettiin, onko käyttäjä asettanut tehosteen päälle kamerassa olevasta skriptistä. (21.) OnRenderImage-funktioon luotiin sama logiikka, jotta se ei turhaan renderöi pelikuvaan tehostetta, joka ei ole päällä.

Jokaiselle tehosteelle luotiin myös oma funktio, jonka sisällä etsittiin kyseinen materiaali tietystä paikasta projektin hierarkiaa. Tämän jälkeen materiaalin varjostimesta haettiin kaikki tarvittavat muuttujat, jotta käyttäjällä oli mahdollisuus säätää tehosteen ominaisuuksia skriptistä käsin.

5 Omien jälkikäsittelytehosteiden testaus

Insinöörityössä luotujen jälkikäsittelytehosteiden testaus suoritettiin käyttäen Unity-pelimoottoriin asennettua versiota 2019.3.4f1 ja sisäänrakennettua renderöintipolkua oletusasetuksineen. Demokaupunki, joka tuotiin aiemmin peliprojektiin, toimi tehosteiden graafisena alustana. Demokaupunkiin ohjelmoitu liikkuva hahmo, johon pelikamera oli asetettu, mahdollisti pelinäkymän muuttumisen ja visuaalisen näkymän vaihtelun, joka mahdollisti sen, että tehosteiden toimivuutta ja ominaisuuksia pystyttiin testaamaan myös liikkeessä.

Roisketehoste

Roisketehoste oli kaikista luoduista omista jälkikäsittelyn tehosteista kaikista yksinkertaisin ohjelmoinnin kannalta. Kuvassa 38 nähtävän veriroisketehosteen testauksen aikana ei havaittu mitään ongelmia. Roisketehoste toimi sellaisena, kuin sen oli alun perin tarkoituskin toimia.

Kuva 38. Veriroisketehoste. Yläkuvassa tehoste pois päältä ja alakuvassa päällä (25).

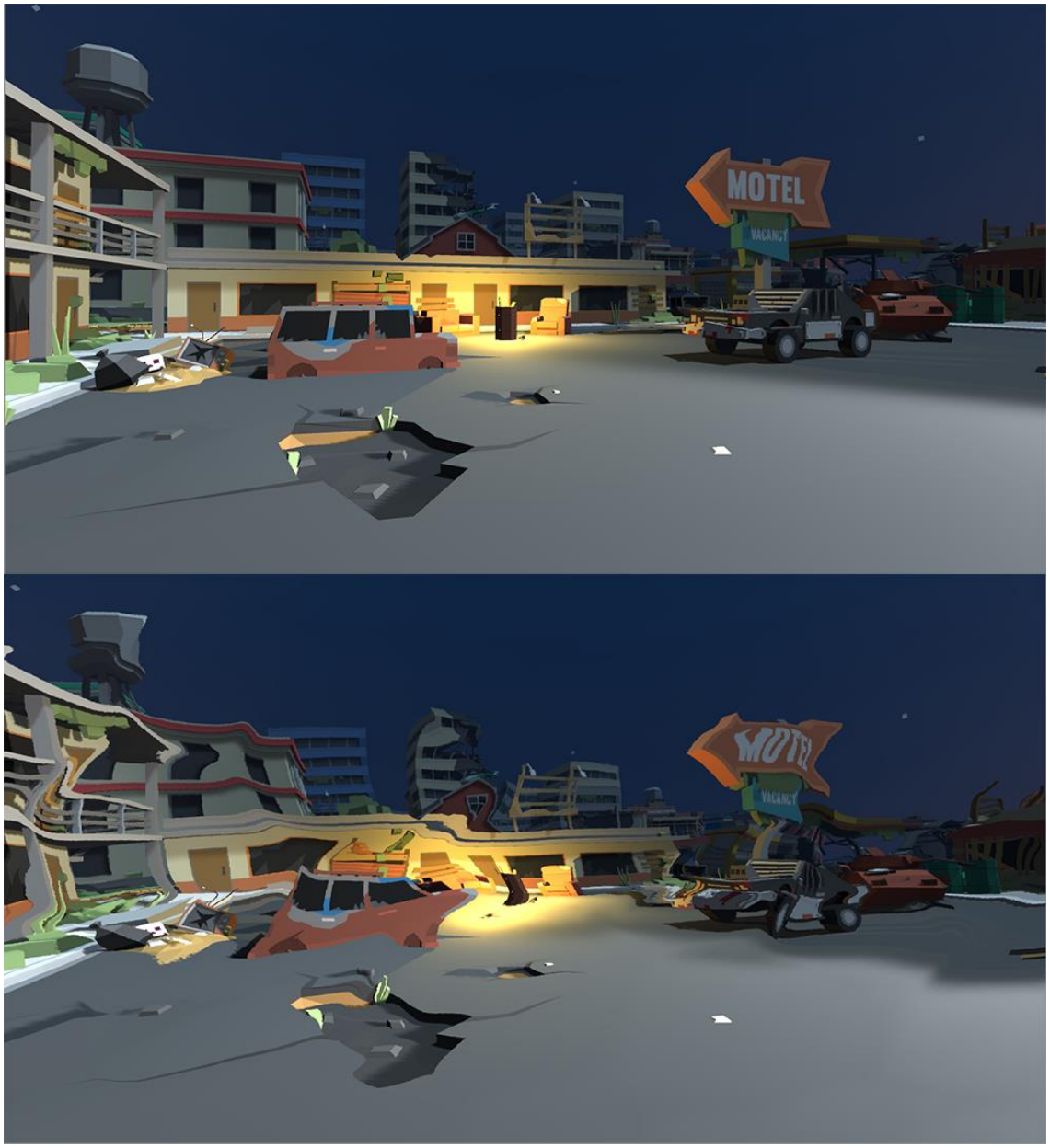
Tehosteeseen luodut tekstuurit toimivat pohjana päätekstuurin päälle implementoidussa pelinäkymässä kuvan molempien 38 ja 39 kuvien mukaisesti. Tekstuureita pystyi vaihtamaan, ja tehosteen läpikuultavaisuusmuuttuja toimi moitteitta.

Kuva 39. Myrkkyroisketehoste. Yläkuvassa tehoste pois päältä ja alakuvassa päällä (25).

Roisketehosteen yksinkertaisuus voidaan ilmentää myös puitteena, mutta tässä tapauksessa sen toiminta ja yksinkertaisuus oli tarkoituksellista. Koska roisketehosteesta luotiin veriroiske- ja myrkkyroisketehoste erikseen, se antoi käyttäjälle heti mahdollisuuden käyttää ja vaihtaa roisketehostetta omaan pelinäkymään sopivaksi myös pelin ollessa päällä.

Vääristymätehoste

Vääristymätehoste oli tehosteista kaikista mielenkiintoisin. Se sisälsi vääristymään asetetun siirtymätekstuurin vääristymän pelinäkymässä ja täten teki kuvasta tekstuurin avulla mielenkiintoisen. Tekstuurin vaihto toisenlaiseksi normaalikartaksi oli myös mahdollista erilaisen vääristymän aikaansaamiseksi. Tehosteeseen luoduilla animaatio- ja nopeusmuuttujilla saatiin aikaseksi hienovaraisia ja myös pelinäkymän kokonaan rikkovia kuvioita, kuten kuvassa 40 on esitetty.



Kuva 40. Vääristymätehoste. Yläkuvassa tehoste pois päältä ja alakuvassa päällä (25).

Tehoste oli ohjelmoitu siten, että vääristymä aikaansaatiin siirtämällä x- ja y- koordinaatteja sivulle. Normaalikartan tai muun tekstuurin vaihtaminen vaatii täten, että siirtymätekstuuri on saumaton. Jos siirtymätekstuuri ei ole saumaton, tulee tehosteesta oudon näköinen eikä se palvele sen tarkoitusta sille tarkoitetulla tavalla.

Pikselitehoste

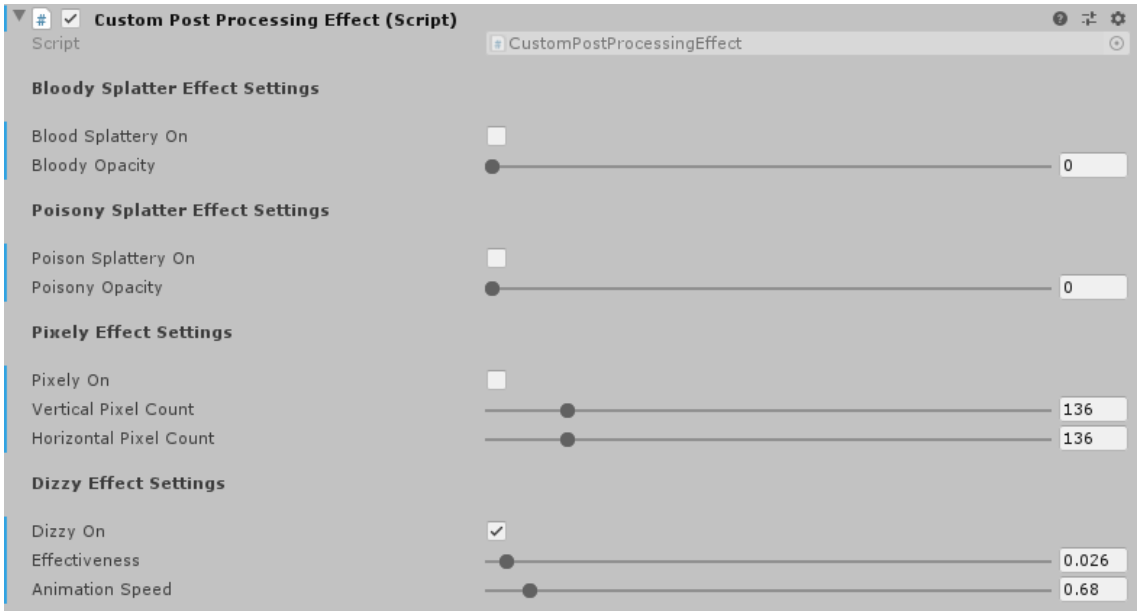
Pikselitehoste oli kaikin puolin onnistunut. Se mahdollisti pikseleiden määrän muutoksen vertikaalisesti ja horisontaalisesti tarttuen pelinäkymän UV-koordinaatteihin ja siihen ohjelmoitujen muuttujien avulla kuvan 41 esittämällä tavalla.

Kuva 41. Pikselitehoste. Yläkuvassa tehoste pois päältä ja alakuvassa päällä (25).

Pelinäkymän UV-koordinaatteja kertomalla, pyöristämällä ja jakamalla aikaansaatu tehoste oli hauska ja uniikki tehoste varsinkin, kun hahmoa liikutti ympäri pelikenttää. Se renderöi pikselöityä pelikuvaa jatkuvasti, joten se antoi mielenkiintoisen näkymän ympäri 3D-mallinnettua maailmaa.

Tehostetyökalu

Kuvassa 42 nähtävä tehostetyökalu toimi moitteitta sille annettujen parametrien ja muuttujien kanssa. Työkalun hyvä puoli oli se, että tehosteita pystyi testaamaan jo ennen pelin käynnistämistä sopivien tehosteiden aikaansaamiseksi ja muokkaamiseksi omaan peliin sopivaksi. Tehosteiden muuttujien muokkaus ja tehosteiden päälle ja pois päältä laittaminen onnistui myös pelin aikana, mikä mahdollisti sen, että tehostetyökalun koodiin pääsi käsiksi myös sen ulkopuolelta. Sen ansioista eri tehosteita pystyttiin vaihtamaan pelin ollessa päällä eri tehosteiden kutsumiseksi oman pelin eri kohdissa.



Kuva 42. Tehostetyökalu kameran valvontaikkunassa (25).

Tehostetyökalun aktivoiminen omassa peliprojektissa oli myös luotu helppokäyttöiseksi. Se oli vain yksi skripti, joka siirrettiin pelikameraan tehosteiden käyttöönoton mahdollistamiseksi. Tämän jälkeen käyttäjällä oli pääsy kaikkiin tehosteisiin suoraan pelikameran näkymän kautta. Kuvassa 43 ovat nähtävissä kaikki tehosteet samanaikaisesti päällä.

Kuva 43. Tehostetyökalun kaikki tehosteet. Yläkuvassa kaikki tehosteet pois päältä ja alakuvassa kaikki tehosteet päällä

Vaikka tehostetyökalu toimikin sille annettujen odotusten mukaisesti, sen parantaminenkin olisi mahdollista. Tehostetyökalu haki koodin avulla tehosteissa käytettävät varjostimet suoraan ennalta määrätystä paikasta. Tämä tarkoitti sitä, että kaikkien tehostetyökalussa käytettyjen varjostimien tuli olla tietyssä paikassa peliprojektin hierarkiaa.

Kaikin puolin tehostetyökalu toimi sille asetetuin ehdoin ja mahdollisti omien jälkikäsittelytehosteiden aktivoimisen ja deaktivoimisen jopa pelin ollessa päällä.

6 Yhteenveto

Insinöörityössä perehdyttiin Unity-pelimoottorin pakettimanagerista saatavaan jälkikäsittelyn työkaluun ja sen asennukseen eri vaiheineen. Työssä tutkittiin myös Unity-pelimoottorin väriavaruuksia ja eri renderöintipolkuja sekä niille asetettuja ominaisuuksia, rajoitteita ja käyttöönoton perusteita. Renderöintipolkuihin tutustumisen ja jälkikäsittelyn tehosteiden asennuksen jälkeen palattiin Unity-pelimoottorin pakettimanagerista saatavan jälkikäsittelyn työkalun eri tehosteisiin. Näihin tehosteisiin perehdyttiin päällisin puolin, jotta työn lukija saisi selkeän käsityksen niiden käyttötarkoituksesta, ominaisuuksista ja mahdollisista rajoitteista.

Unityn tarjoamien jälkikäsittelyn tehosteiden tarkastelun jälkeen alettiin ohjelmoida omia jälkikäsittelyn tehosteita. Nämä tehosteet luotiin varjostimilla, mikä vaati HLSL-ohjelmointikielen eri kehityskirjastojen ja ohjelmointitapojen opettelemista. Opettelun pohjana käytettiin Unityn verkkosivujen ohjelmointikirjastoa, Udemy-verkkosivun tarjoamia internetkursseja, Youtube-videoita, lukuisten eri varjostimien ohjelmointiin keskittyvien verkkosivujen tutkimista ja myös henkilökohtaista apua alan ammattilaisilta.

Omien jälkikäsittelyn tehosteiden luonnin jälkeen ohjelmoitiin työkalu, jonka avulla peliprojektissa oleva pelikamera pystyi renderöimään luotuja tehosteita. Työkalusta tehtiin myös helppokäyttöinen, jotta sen käyttöliittymä ja ominaisuudet olivat helposti käytettävissä. Työkalu ohjelmoitiin käyttäen C#-ohjelmointikieltä ja sen tarjoamia kehityskirjastoja. Työkalun ohjelmointi ei onnistunut ongelmitta, mutta henkilökohtainen apu alan ammattilaisilta oli perustana työkaluun implementoidun koodin toimivuuden takaamiseksi.

Omasta jälkikäsittelyn työkalusta tuli toimiva ja helppokäyttöinen. Se sisälsi sen ominaisuuksiin asetetut tavoitteet ja odotukset. Ohjelmoitujen varjostimien ja työkalun koodin refaktorointi mahdollisesti parantaisi suorituskykyä, mutta tämän tason projektissa se ei kuitenkaan vaikuttaisi prosessien nopeuteen huomattavasti. Työkalua pystyy käyttämään jälkikäsittelyn työkaluna Unity-pelimoottorilla tehdyissä peliprojekteissa, joissa käytetään sisäänrakennettua renderöintipolkua.

Lähteet

1 Post Processing Overview. 2020. Verkkoaineisto. Unity. <https://docs.unity3d.com/Manual/PostProcessingOverview.html>. Luettu 24.2.2020.

2 Post Processing Stack. 2020. Verkkoaineisto. Unity. <https://learn.unity.com/tutorial/introduction-to-the-post-processing-stack-2018#>. Luettu 24.2.2020.

3 Render Pipelines. 2020. Verkkoaineisto. Unity. < https://docs.unity3d.com/Manual/render-pipelines.html>. Luettu 25.2.2020.

4 Rendering Paths. 2020. Verkkoaineisto. Unity. <https://docs.unity3d.com/Manual/RenderingPaths.html>. Luettu 26.2.2020

5 Universal Render Pipeline. 2020. Verkkoaineisto. Unity. <https://docs.unity3d.com/Manual/universal-render-pipeline.html>. Luettu 26.2.2020

6 High Definition Render Pipeline. 2020. Verkkoaineisto. Unity. <https://docs.unity3d.com/Manual/high-definition-render-pipeline.html>. Luettu 26.2.2020

7 Scriptable Render Pipeline. 2020. Verkkoaineisto. Unity. <https://docs.unity3d.com/Manual/ScriptableRenderPipeline.html>. Luettu 26.2.2020

8 Anti-Aliasing. 2020. Verkkoaineisto. Unity. < https://github.com/Unity-Technologies/PostProcessing/wiki/Anti-aliasing>. Luettu 27.2.2020.

9 Ambient Occlusion. 2020. Verkkoaineisto. Unity. <https://github.com/Unity-Technologies/PostProcessing/wiki/Ambient-Occlusion>. Luettu 27.2.2020.

10 Bloom. 2020. Verkkoaineisto. Unity. <https://github.com/Unity-Technologies/PostProcessing/wiki/Bloom>. Luettu 27.2.2020.

11 Chromatic Aberration. 2020. Verkkoaineisto. Unity. <https://github.com/Unity-Technologies/PostProcessing/wiki/Chromatic-Aberration>. Luettu 28.2.2020.

12 Color Grading. 2020. Verkkoaineisto. Unity. <https://github.com/Unity-Technologies/PostProcessing/wiki/Color-Grading>. Luettu 28.2.2020.

13 Deferred Fog. 2020. Verkkoaineisto. Unity. <https://github.com/Unity-Technologies/PostProcessing/wiki/Deferred-Fog>. Luettu 29.2.2020.

14 Depth of Field. 2020. Verkkoaineisto. Unity. <https://github.com/Unity-Technologies/PostProcessing/wiki/Depth-of-Field>. Luettu 29.2.2020.

15 Auto Exposure. 2020. Verkkoaineisto. Unity. <https://github.com/Unity-Technologies/PostProcessing/wiki/Auto-Exposure>. Luettu 29.2.2020.

16 Grain. 2020. Verkkoaineisto. Unity. <https://github.com/Unity-Technologies/PostProcessing/wiki/Grain>. Luettu 1.3.2020.

17 Motion Blur. 2020. Verkkoaineisto. Unity. <https://github.com/Unity-Technologies/PostProcessing/wiki/Motion-Blur>. Luettu 2.3.2020.

18 Screen Space Reflections. 2020. Verkkoaineisto. Unity. <https://github.com/Unity-Technologies/PostProcessing/wiki/Screen-space-Reflections>. Luettu 3.3.2020.

19 Vignette. 2020. Verkkoaineisto. Unity. <https://github.com/Unity-Technologies/PostProcessing/wiki/Vignette>. Luettu 4.3.2020.

20 OnRenderImage. 2020. Verkkoaineisto. Unity. <https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnRenderImage.html>. Luettu 6.3.2020.

21 OnValidate. 2020. Verkkoaineisto. Unity. <https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnValidate.html>. Luettu 8.3.2020.

22 Postprocessing stack installation. 2020. Verkkoaineisto. Unity. < https://docs.unity3d.com/Packages/com.unity.postprocessing@2.3/manual/Quick-start.html>. Luettu 8.4.2020.

23 Linear or Gamma workflow. 2020. Verkkoaineisto. Unity. < https://docs.unity3d.com/Manual/LinearRendering-LinearOrGammaWorkflow.html>. Luettu 14.4.2020.

24 Unity. 2019. Versio 2019.3.0a5. Unity Technologies.

25 Unity. 2019. Versio 2019.3.4f1. Unity Technologies.