Kyllönen Jaakko TTK16SP

Varjostimet ja niiden käyttö visuaalisen tyylin saavuttamiseen Unity-pelimoottorissa

Tradenomi

Tietojenkäsittely

Kevät 2020

KAMK • University of Applied Sciences

Tiivistelmä

Tekijä: Kyllönen Jaakko

Työn nimi: Varjostimet ja niiden käyttö visuaalisen tyylin saavuttamiseen Unity-pelimoottorissa

Tutkintonimike: Tradenomi, tietojenkäsittely

Asiasanat: varjostin, grafiikka, 3D

Opinnäytetyön tavoitteena oli käsitellä 3D-varjostimia ja niiden käyttöä Unity-pelimoottorissa. Tarkoituksena oli antaa varjostimien toiminnasta helposti luettava kuva, jota varten ei tarvitse omata laajaa tietämystä ohjelmoinnista. Tietoa sovellettiin peliprojektissa.

Teoriassa tämä opinnäytetyö kävi 3D-mallinnuksen historiaa lyhyesti läpi, ja miten varjostimet ovat kehittyneet vuosien myötä siten, että pelikehittäjät ovat pystyneet tuottamaan realistisempia tietokonegrafiikalla luotuja ympäristöjä. Varjot, valaistus ja normaalikartoitus ovat 3D-projekteissa alueita, mitä myös käytiin läpi. Tavoitteena oli selventää varjostimien määrityksen vähemmän tietävälle lukijalle.

Unity-pelimoottorissa 3D-varjostimet ovat yksinkertaisesti käytettävissä ja muokattavia, mitä tultiin käyttämään hyödyksi peliprojektissa.

Käytännön osuudessa varjostimien käyttöä sovellettiin visuaalisen ulkonäön rakentamiseen 3D-pelimoottorissa ilman kokemusta ohjelmoinnista. Opinnäytetyö kävi läpi halutun graafisen tyylin ja miten Unity-moottorin kautta pystyi muokkaamaan olemassa olevia varjostimia visuaalisen tavoitteen saavuttamiseen.

KAMK • University of Applied Sciences

Abstract

Author: Kyllönen Jaakko

Title of the Publication: Shaders and Their Use in Achieving a Visual Style in the Unity Game Engine

Degree Title: Bachelor of Business Administration, Information Systems

Keywords: shader, graphics, 3D

The objective of the Bachelor's thesis was to handle 3D-shaders and their usage in the Unity game engine. The goal was to give an easily readable view on the function of shaders without needing a larger knowledge about programming. This theory was applied in a game project.

The theoretical part of the thesis rudimentarily went through the history of 3D-modeling and how shaders have evolved over the years, as game developers have aimed to produce more realistic environments created with computer graphics. Shadows, lighting and normal mapping are areas of 3D-modeling that were also explored. The goal was to clarify the definition of shaders to a less knowledgeable reader.

In the practical section of the usage of shaders was applied to build a visual style in a 3D game engine without prior knowledge in programming. The thesis went over the desired graphic style and how the Unity engine was used to modify existing shaders to achieve the visual goals.

Alkusanat

Aloittelevana 3D-graafikkona yksi isommista haasteista oppia työkuvassani oli tietokonetuotettujen grafiikoiden tekninen puoli. Alussa jo verkkolankamallien tekeminen oli vaikea sisäistää, kun yritin oppia mallintamisohjelmien käytön, ja sitten tuli vastaan tekstuurien ja materiaalien teko. Teksturoinnin ja UV-kartoituksen käsitteet pystyin ymmärtämään nopeasti, mutta kun näin materiaalit ja kaikki niiden mahdolliset toiminnot, ajatukseni kulku katkesi ja tunsin kovan esteen edessäni. Materiaaleista tuli hidaste oppimiselleni.

Opintojeni aikana pääsin työskentelemään 3D-mallien kanssa koko ajan enemmän, samalla oppien pienissä määrissä, miten materiaalit toimivat ja miten käyttää niiden yksinkertaisempia tapoja tuottaa realistisia pintoja. Opin käyttämään varjostimia Unity- ja Unreal Engine-pelimoottoreissa eri tarkoituksia varten, ja aloin sisäistämään tätä toistaiseksi haastavaa puolta 3D-grafiikasta. Ennen pitkää eteeni tuli projekti, missä minun tuli luoda varjostin, millä saisin peliin tietynlaisen visuaalisen ulkonäön. Tämän projektin aikana pääsin näkemään syvälle varjostimien toimintaan, ja oppimiseni parantui huomattavasti.

Päätin tehdä opinnäytetyöni varjostimista ja niiden käytöstä peliprojektissa, koska tahdoin vielä päästä itse sisäistämään lisää varjostimien toiminnasta, ja tuoda muille esille niiden eri toimintoja. Itse graafikkona en usein koske ohjelmoinnin puolelle, mutta varjostimia muokatessani sain hieman opetusta siitäkin, vaikka en pystyisi luomaan omaa varjostinta tyhjästä. Tavoitteeni on esitellä varjostimien peruspiirteet, historia ja käyttö 3D-grafiikassa pinnallisella tasolla, että aloittelevakin graafikko saisi tätä opinnäytetyötä lukiessaan hieman ymmärrystä varjostimien teknisestä puolesta.

Sisällys

Symboliluettelo

Gradientti	Kahden värin välinen siirtymä
Hahmontaminen	Engl. *rendering*, tietokonetuotetun grafiikan ulostulo ruudulle
Normaalikartta	Kartoitus, joka kertoo, miten valo heijastuu 3D-mallissa
Pikseli	Yksittäinen valon piste näytöllä
Pistetulo	Matemaattinen lasku vektoreille
Polygoni	Vähintään kolmen verteksin geometrinen 3D-muoto
Primitiivi	Yksinkertainen geometrinen 3D-muoto
Radiositeetti	Valaistus, joka syntyy valon heijastuessa objektin pinnasta
Rasterointi	Tietokonetuotetun ympäristön muuttaminen ruudun pikseleiksi
Sprite	Kaksiulotteinen graafinen elementti
Vektori	Matemaattinen ilmaisu, jolla on pituus ja suunta
Verteksi	Yksittäinen piste 3D-avaruudessa

1 Johdanto

Tämän opinnäytetyön teko alkoi syksyllä 2019 käydyllä peliprojektikurssilla Kajaanin ammattikorkeakoulussa. Ryhmä nimeltä Reset Phoenix kehitti peliä nimeltä Hilja & The King of the Forest, jonka tuotannossa tämän opinnäytetyön kirjoittaja oli mukana johtavana graafikkona.

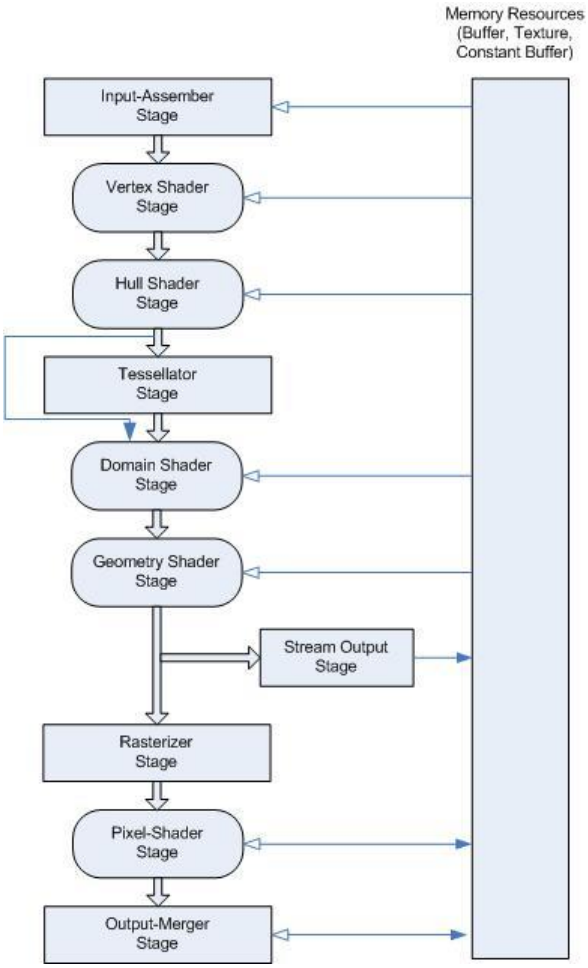
Varjostimien kanssa työskentely oli osa peliprojektia, mistä syntyi pohja opinnäytetyölle. Tietokonegrafiikan varjostimien käytön oppiminen ja ymmärtäminen kehittyi sopivaksi aiheeksi, josta pystyy luomaan yksinkertaisesti luettavan esittelyn graafikoille.

Opinnäytetyö tulee käymään ensimmäisessä osiossa läpi varjostimien määritelmän, kehityksen ja yleisen käytön tietokonegrafiikassa. Projektin vaiheessa annetaan käytännön toteutuksesta yleiskuvaa, miten varjostimen voi muokata tiettyä visuaalista tyyliä varten.

2 3D-grafiikka ja varjostimet

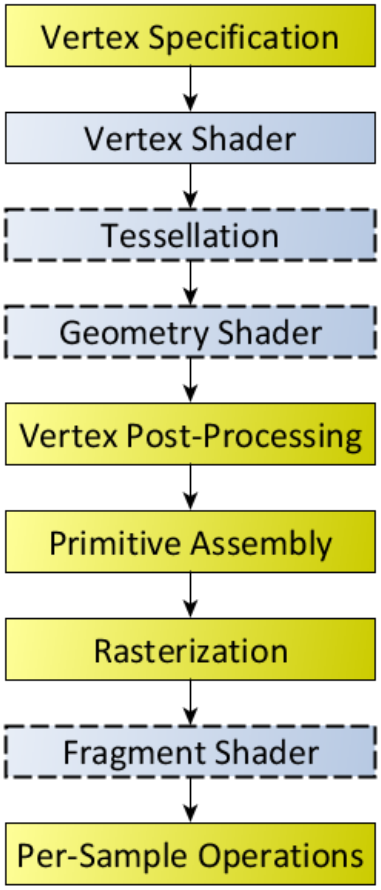
Tietokonegrafiikoiden termeissä shader eli varjostin on erityisellä varjostinkielellä kirjoitettu ohjelma, joka toteutuu grafiikan suorituksen liukuhihnalla kertoen tietokoneelle, miten sen tulee hahmontaa jokainen pikseli. Näitä ohjelmia kutsutaan varjostimiksi, koska niitä sovelletaan eniten 3D-grafiikoiden valaistus- ja varjostusefekteihin, mutta niitä voi käyttää tuottamaan muitakin erikoisefektejä. [1.]

Esimerkkinä grafiikan hahmontamisen liukuhihnalle (engl. pipeline) toimii Microsoftin ohjelmoitava Direct3D 11 -liukuhihna, joka on suunniteltu luomaan grafiikkaa reaaliaikaisille pelisovelluksille [2]. Seuraava diagrammi näyttää, miten tässä liukuhihnassa kolmiulotteinen näkymä hahmonnetaan kaksiulotteiselle ruudulle. (Kuva 1.)



Kuva 1. Direct3d 11 –liukuhihnan toimintavaiheet. [2.]

Ensimmäisenä vaiheena on verteksien määrittely, missä sovellus järjestää listan vertekseistä, jotka määrittävät rajoitukset yksinkertaisille muodoille, kuten kolmioille, viivoille ja pisteille. Tämä lista vertekseistä lähetetään liukuhihnalle. Sen jälkeen alkaa verteksien käsittely, jossa hahmottamisoperaatio käy varjostinoperaatioilla läpi kolme eri vaihetta: verteksivarjostimen, tessellaation ja geometriavarjostimen. (Kuva 2.)



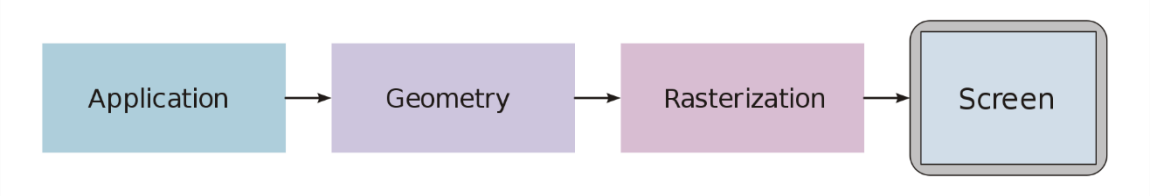
Kuva 2. Hahmottamisliukuhihnan diagrammi. [3.]

Verteksivarjostimet toteuttavat peruskäsittelyt yksittäisille vertekseille, syöttäen ominaisuuksia verteksihahmontamisesta. Verteksivarjostin sitten kääntää jokaisen sisääntulevan verteksin yksittäiseen ulosmenevään verteksiin, seuraten käyttäjän määrittelemää ohjelmaa.

Tessellaatiossa verteksidataa jaetaan pienempiin yksinkertaisiin muotoihin eli primitiiveihin, käyden läpi kaksi varjostinvaihetta ja yhden kiinteäfunktiovaiheen.

Geometriavarjostimessa käyttäjän määrittelemä ohjelma käsittelee jokaisen sisään tulevan primitiivin, palauttaen nolla tai enemmän ulosmeneviä primitiivejä. [3.]

Liukuhihnat käyvät läpi usean vaiheen rakentaessaan varjostinta ja toteuttaessaan sitä peliympäristöön, mutta yksinkertaisimmillaan graafinen liukuhihna voidaan jaotella kolmeen pääosaan: sovellus, geometria ja rasterointi. (Kuva 3.)



Kuva 3. Graafisen liukuhihnan vaiheet yksinkertaistettu. [4.]

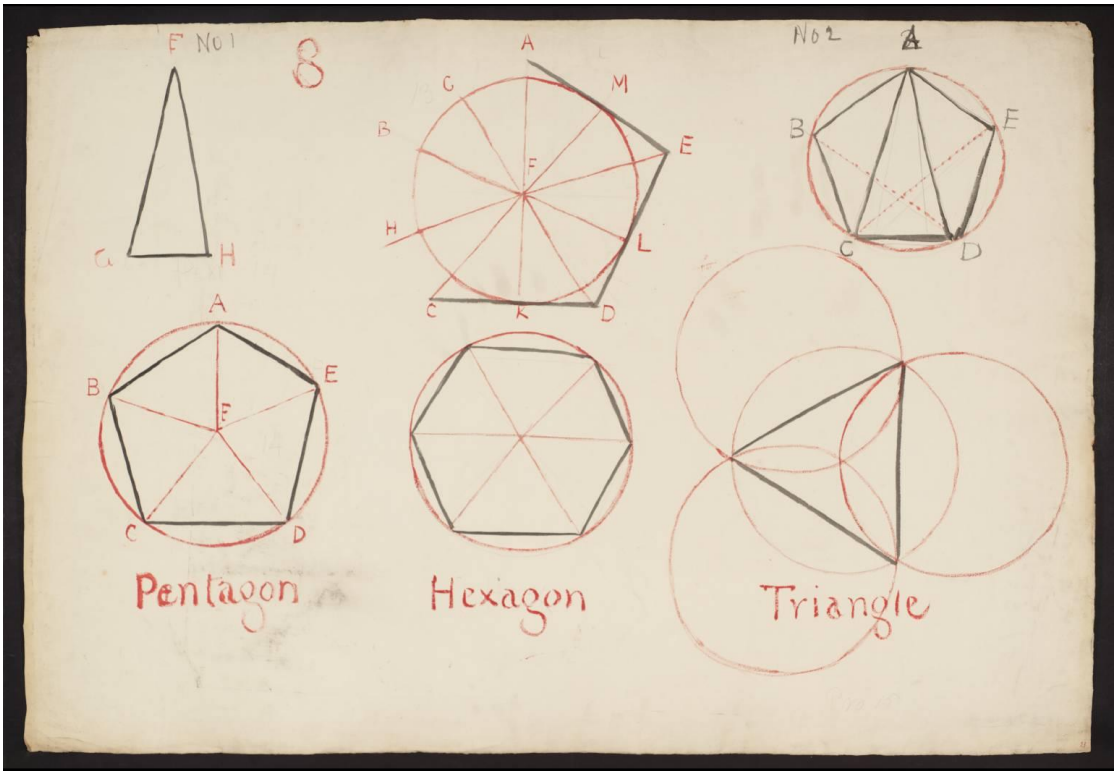
Sovelluksessa peliympäristöön tehdään tarvittavat muutokset joko animaation tai käyttäjän toimintojen tapahtuessa. Uusi ympäristö primitiiveineen järjestetään ja lähetetään seuraavaan vaiheeseen. Ohjelmoijat työskentelevät erityisesti tässä vaiheessa.

Geometrian vaiheessa ympäristöön rakennetaan ja lisätään mallien ja kameroiden muutokset, valaistukset, projektiot, leikkaukset ja ikkuna-näkymämuutokset.

Rasteroinnin vaiheessa erilliset fragmentit luodaan jatkuvista primitiiveistä. Näistä kootaan rasteroitu kuva, jossa kaikki graafiset elementit on muutettu pikseleiksi, jotka tulevat ruudulle. Näin saadaan valmis yksittäinen kuva, joka kohdistetaan näytölle. [5.]

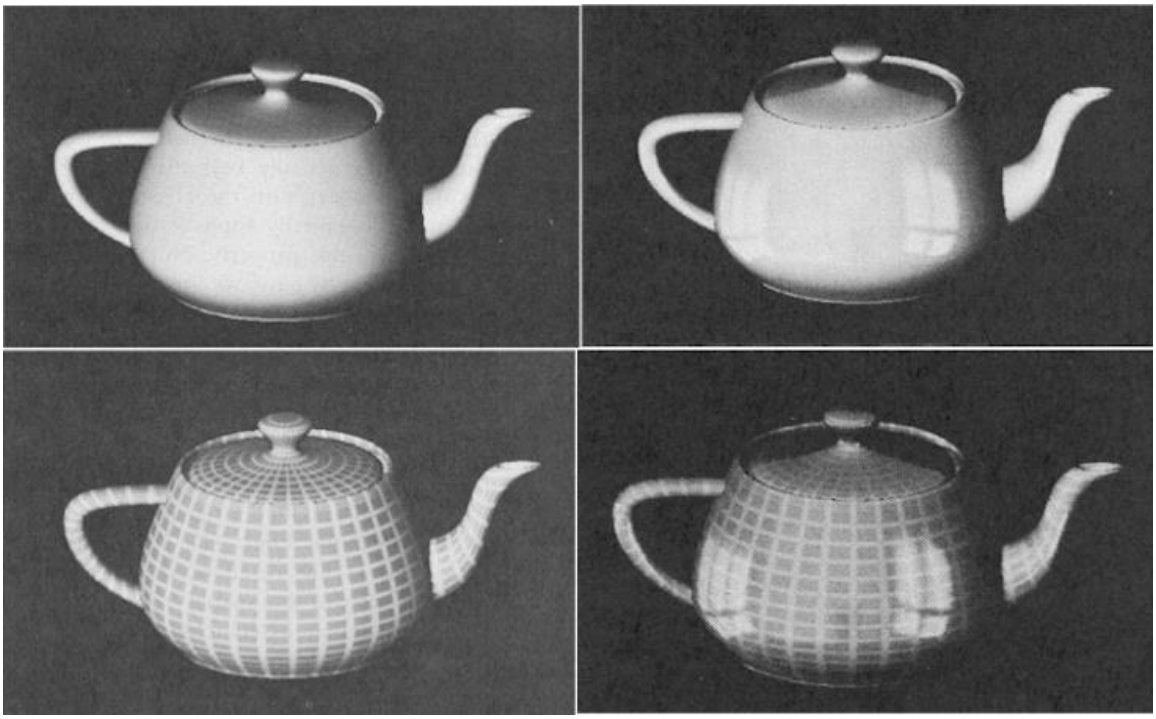
2.1 3D-grafiikan historia

3D-grafiikan käsite oli olemassa jo pitkään ennen tietokoneiden syntyä. Kreikkalainen matemaatikko Eukleides (engl. Euclid) tunnetaan geometrian isänä, ja hän loi pohjan kolmiulotteisen hahmottamisen perusideoille yli 2000 vuotta takaperin. (Kuva 4.) 1600-luvulla René Descartes loi analyyttisen geometrian, millä pystyy mittaamaan etäisyyksiä ja sijainteja. 1800-luvulla englantilainen matemaatikko James Joseph Sylvester keksi matriisimatematiikan, jota sovelletaan nykyään jokaisessa tietokoneella luodussa kuvassa, jossa käytetään heijastuksia tai valon vääristelyä. 1950-luvulla ensimmäiset tietokoneet tulivat käyttöön, ja samalla kolmiulotteinen hahmottaminen niiden avulla matematiikassa. [6.]



Kuva 4. Eukleidesin geometriaa. [6.]

Ensimmäinen tietokoneella luotu 3D-animaatio syntyi 1970-luvulla, kun amerikkalainen Ed Catmull, yksi animaatiostudio Pixarin perustajista, loi videon, jossa hän oli kehittänyt animaation kolmiulotteiselle kädelle ja ihmiskasvoille. [7.] Myöhemmin samalla vuosikymmenellä Martin Newell loi kuuluisan Utahin teepannun, jota käytetään 3D-grafiikassa samanlaisena lähtökohtana kuin ohjelmoinnissa käytetään lausetta ”Hello World!”. (Kuva 5.) Kyseinen teepannu luotiin mallikohdaksi monimutkaisempien 3D-mallien varjojen ja heijastavien pintojen esimerkiksi, ja on kehittynyt sisäiseksi vitsiksi 3D-grafiikassa. [8.]



Kuva 5. Martin Newellin kuuluisa Utahin teepannu. [8.]

2.2 Varjostimien kehittyminen

Aikaisin varjostimen muoto ei ollut vielä muuta kuin yksinkertainen rasteroija, joka pystyi kartoittamaan yhden tekstuurin kolmioprimitiiviin. Varjostimen käsite itsessään oli ollut jonkin aikaa jo olemassa, mutta sellaisen ohjelmoitavuuden soveltaminen laitteiston hahmottamisen ideana ei. Animaatiostudio Pixar loi varjostimen nykyisen käsitteen RenderMan Interface Specification – ohjelmointirajapinnalla, josta syntyi RenderMan-varjostinkieli. [9.]

Videopelien historiassa Id Software oli ensimmäisiä pelintekijöitä, jotka käyttivät varjostimia. Wolfenstein 3D-pelissä käytettiin tekstuureja kolmiulotteisissa seinissä, ja Doom-pelissä käytettiin erikokoisia tekstuureja eri pinnoille ja ympäristövalaistusta. (Kuva 6, 7.)

Kuva 6. Kuvakaappaus Wolfenstein 3D-pelistä. [10.]

Kuva 7. Kuvakaappaus Doom-pelistä. [11.]

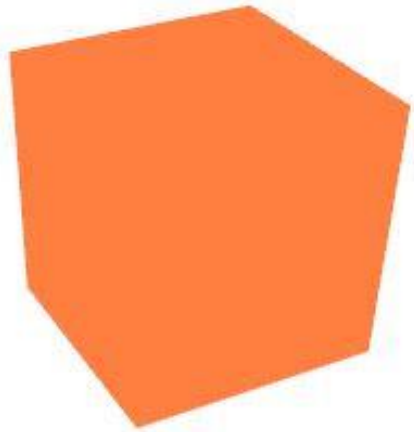
Id Softwaren osa pelivarjostimien kehityksessä ei kuitenkaan loppunut siihen. Heidän myöhempi pelinsä Quake II, julkaistu vuonna 1997, oli ensimmäinen videopeli, joka käytti värillistä valaistusta ympäristöissään. Valo myös pystyi hyppimään ympäristössä, joka tuotti simuloitua radiositeettia. [12.]

3 Varjostimien käyttö

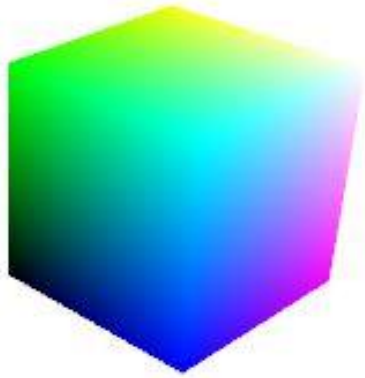
Varjostimet ovat kehittyneet käytettävyydessään niin paljon, että niillä voi nykyään saavuttaa lähes mitä vain grafiikan tuotannossa. Yksinkertaisten väritekstuurien lisäksi niillä tuotetaan melkein kaikki erikoisefektit tietokonetuotetuissa kuvissa. Värit, varjot ja läpinäkyvyys ovat vain alkua. Varjostimista on tullut korvaamaton osa grafiikkaa.

3.1 Värit

Varjostimen yksinkertaisin muoto on väri, jolla saadaan joko yksi sävy tai kokonainen tekstuurikuva objektiin. Objektin väri voidaan määrittää yksinkertaisella fragmenttivarjostimella. (Kuva 8.) Verteksivarjostimella voi taas määrittää jokaiselle verteksille oma värinsä. (Kuva 9.) [13.]



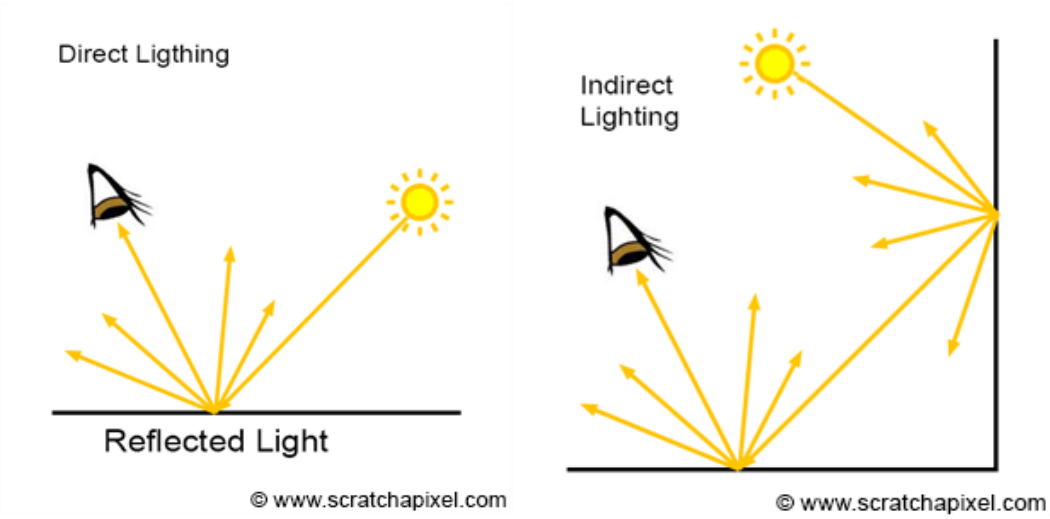
Kuva 8. Yksisävyinen 3D-objekti. [14.]



Kuva 9. Eriväriset verteksit 3D-objektissa. [13.]

3.2 Varjot ja valaistus

Valon uskottava käyttäytyminen grafiikassa on äärimmäisen tärkeää realistisen ympäristön luomiseen. Objektit jaetaan kahteen ominaisuusryhmään: objektin pintojen geometrisiin ominaisuuksiin ja ominaisuuksiin, jotka määrittävät valon ja objektin vuorovaikutukseen keskenään. [15.] Kun näemme esineen, näemme oikeasti valon, joka heijastuu esineestä, joko suorassa tai epäsuorassa valaistuksessa. (Kuva 10.)



Kuva 10. Suora ja epäsuora valon heijastuminen. [15.]

Kun tietokonetuotetussa ympäristössä käytetään valonlähteitä, tämän valon vaikutus objekteihin määritellään varjostimen kautta. [16.] Värien varjostus määritellään seuraavilla termeillä:

- Diffuusio: Valo, joka heijastuu objektista jokaiseen suuntaan. Diffuusiolla määritellään objektin pohjaväri.

- Ympäröivä: käytetään hyppivän valon simulointiin täyttämällä objektin kohdat, missä valoa ei löydy.

- Spekulaarinen: Tämä valo heijastuu vahvemmin yhteen tiettyyn suuntaan, yleensä peilisuuntaan valon tulokulmasta.

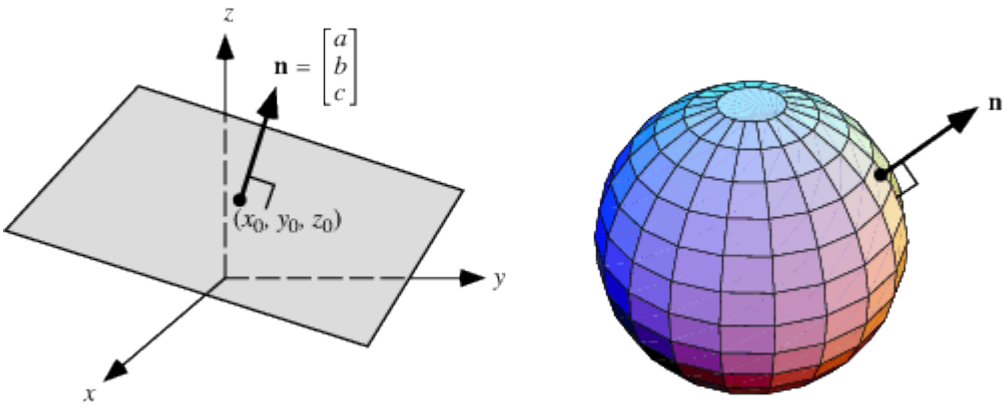
- Emissio: Objekti itse tuottaa valoa.

Kuva 11. Diffuusio, ympäröivä, diffuusio+ympäröivä, diffuusio+ympäröivä+spekulaarinen valaistus objektissa. [16.]

Näitä piirteitä hienovaraisesti yhdistellen 3D-grafiikassa voidaan saavuttaa realistisia valon heijastuksia ympäristössä (Kuva 11.) Tästä alkupisteestä syntyikin varjostimen termi, vaikka varjostimet tekevät paljon muutakin kuin pelkästään valon piirteitä.

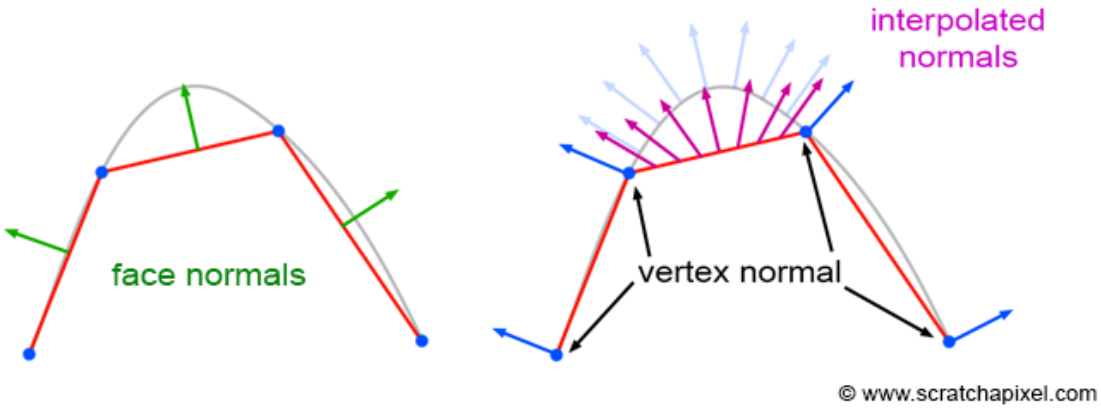
3.3 Normaalikartat

3D-grafiikassa ja geometriassa termillä normaali tarkoitetaan vektoria, joka on kohtisuorassa geometrista pintaa päin. [17.] Tämä normaalivektori lasketaan jokaiselle yksittäiselle pinnalle 3D-malleissa, ja näiden pintojen vektorien yhdistelmällä saadaan valaistuksen käyttäytyminen tarkennettua. (Kuva 12.)

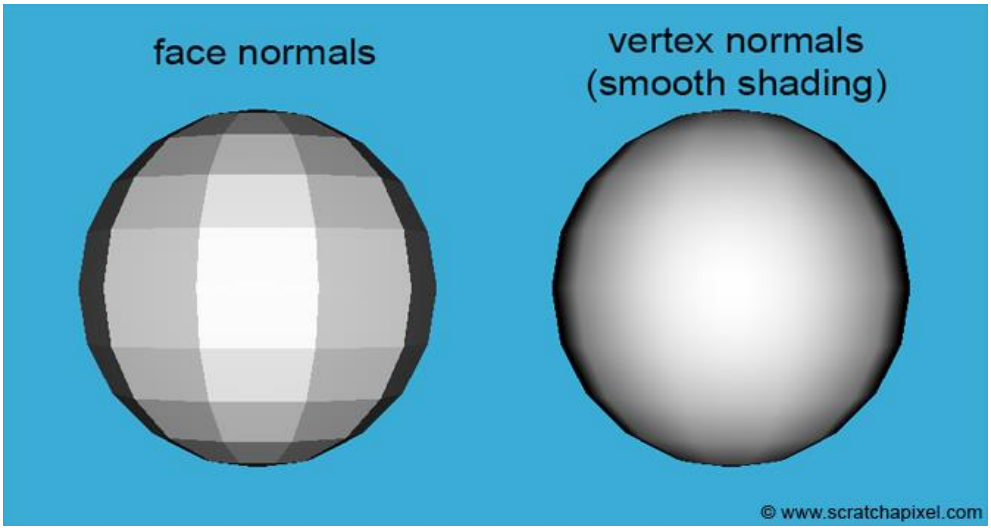


Kuva 12. *N* eli normaalivektorin suunta pinnasta. [17.]

Erikseen lasketuilla normaalivektoreilla syntyisi vain kovapintainen muoto, ja 3D-mallissa näkyisi vain kokoelma litteitä geometrisia pintoja. Vuonna 1971 Henri Gouraud esitteli verteksinormaalin termin ja pehmeän varjostuksen menetelmän. [18.] Sen sijaan, että normaalit laskettaisiin pinnoilta, otetaankin säilöön verkkolankamallin jokaisen verteksin normaali, ja luodaan teeskennellyn pehmeän pinnan verteksien välille lineaarisella interpolaatiolla. (Kuva 13, 14.)



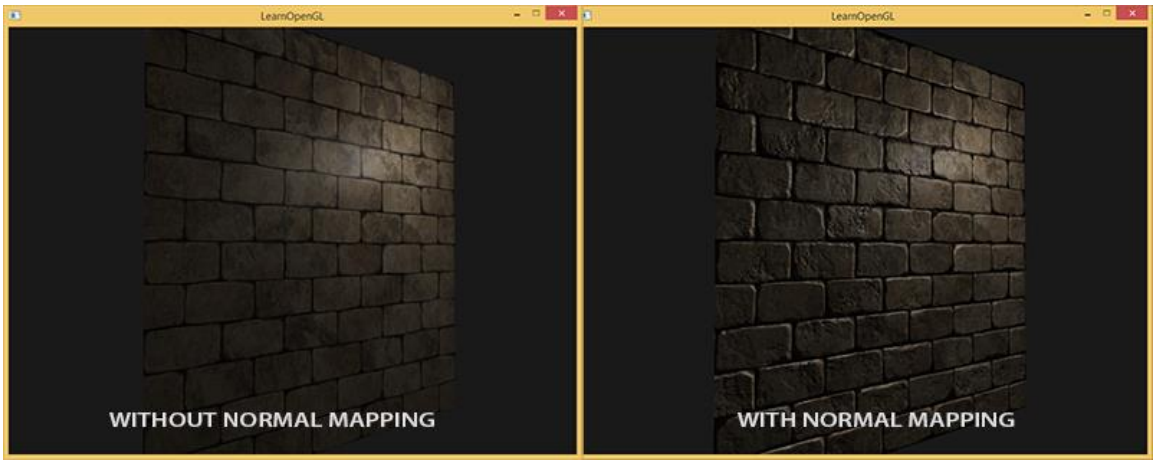
Kuva 13. Pintanormaalin ja verteksinormaalin ero valon laskemisessa. [18.]



Kuva 14. Kovan ja pehmeän varjostuksen ero. [18.]

Normaalien käyttö tulee eniten esille videopelien tuotannossa, sillä varjostimien avulla normaaleista voi tehdä tekstuurikarttoja, jotka tuovat esille illuusion yksityiskohdista, jotka eivät oikeasti ole verkkolankamallissa.

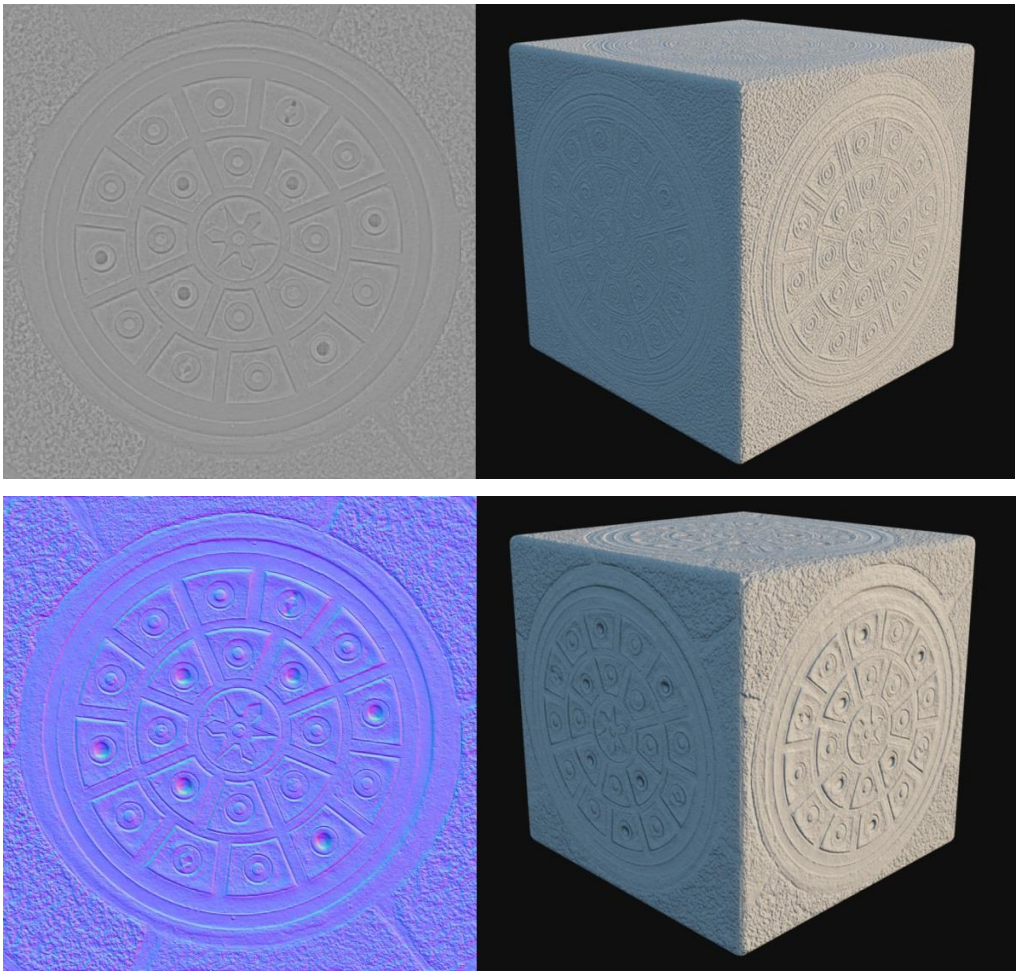
Valaistustekniikka määrittää 3D-mallien muodon kokonaan kohtisuorista normaalivektoreista. [19.] Ottamalla talteen tiedot monikasvoisesta ja yksityiskohtaisesta mallista näitä normaaleja pystytään käyttämään erillisessä tekstuurissa, jolla voi erikseen määrittää, miten valo heijastuu objektissa sen sijaan, että valaistus seuraisi vain objektin omia pintoja. Tätä tekniikkaa kutsutaan normaalikartoittamiseksi. Tällä äärimmäisen käytännöllisellä menetelmällä yksi neljän verteksin neliömuoto muuttuu realistiseksi seinäksi. (Kuva 15.)



Kuva 15. Normaalikartoituksen vaikutus tiiliseinään. [19.]

Normaalikartoituksesta on tullut suuri osa videopelien grafiikkatuotantoa, sillä peliympäristöissä pitää olla säästäväinen 3D-mallien polygonien määrässä. Normaalikartat auttavat tuomaan paljon enemmän realistisuutta peliin käyttämättä kehitysalustan tehoja enemmän kuin tarpeeksi.

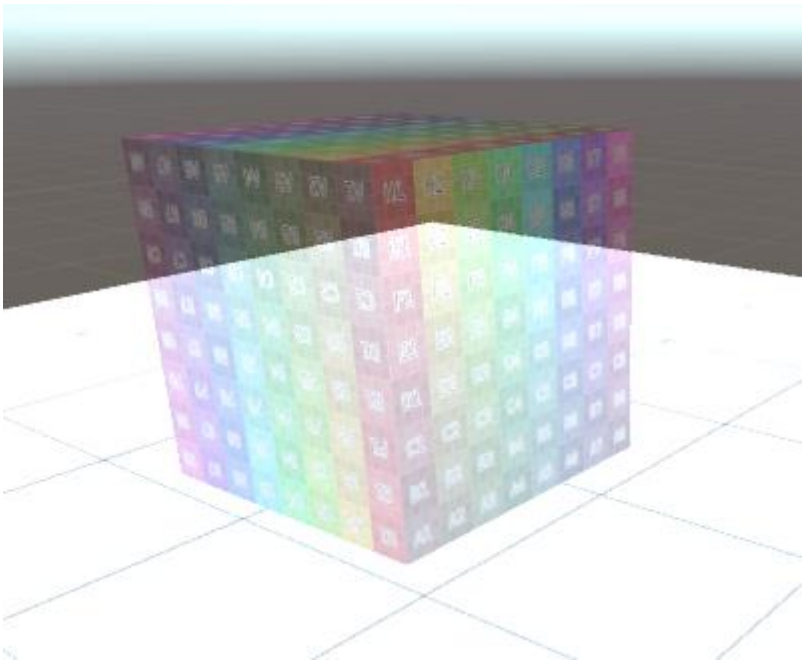
Ennen normaalikarttoja yksityiskohtia varten käytettiin enemmän kohoumakarttoja, joiden ero oli siinä, että kohoumakartoissa pystyi kertomaan valaistukselle vain suoraa syvyyttä ylös tai alas mustavalkoisella skaalalla. (Kuva 16, 17.) Normaalikartassa taas käytetään RGB-tietoja X-, Y- ja Z-akselilla 3D-avaruudessa. [20.] Kohoumakarttoja käytetään vieläkin graafisessa tuotannossa, koska se on välillä helpompi tapa saada pieniä yksityiskohtia esille.



Kuva 16, 17. Kohouma- ja normaalikartoitettu kuutio. [21.]

3.4 Läpinäkyvyys

Tietyt objektit ja materiaalit tarvitsevat läpinäkyvyyttä, kuten lasi ja todella ohuet esineet. Lisäämällä RGB-värimaailmaan alpha-kanavan, valon voi sallia matkustaa koko objektin läpi tietyllä asteella. (Kuva 18.) Läpinäkyvyyttä, spekulaarisuutta, emissiota, ympäröivää valoa ja diffuusiota muokaten voi luoda monimutkaisempia materiaaleja, joiden vuorovaikutus valon kanssa voi olla erittäin vaikea saavuttaa realistiselle tasolle.

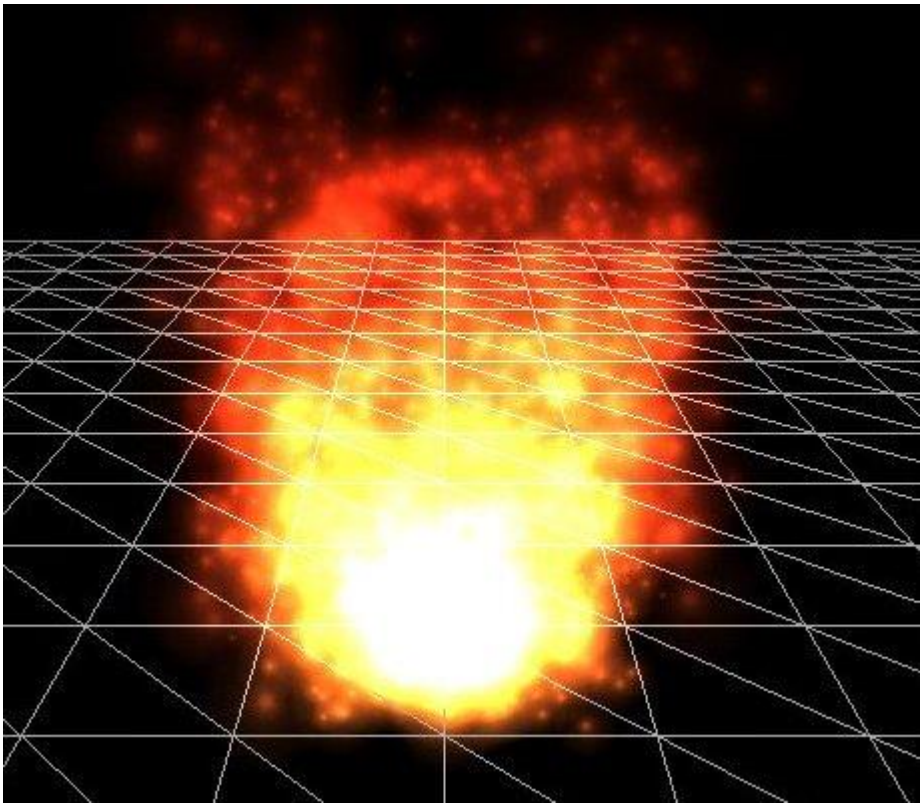


Kuva 18. Läpinäkyvä materiaali. [22.]

3.5 Partikkelit

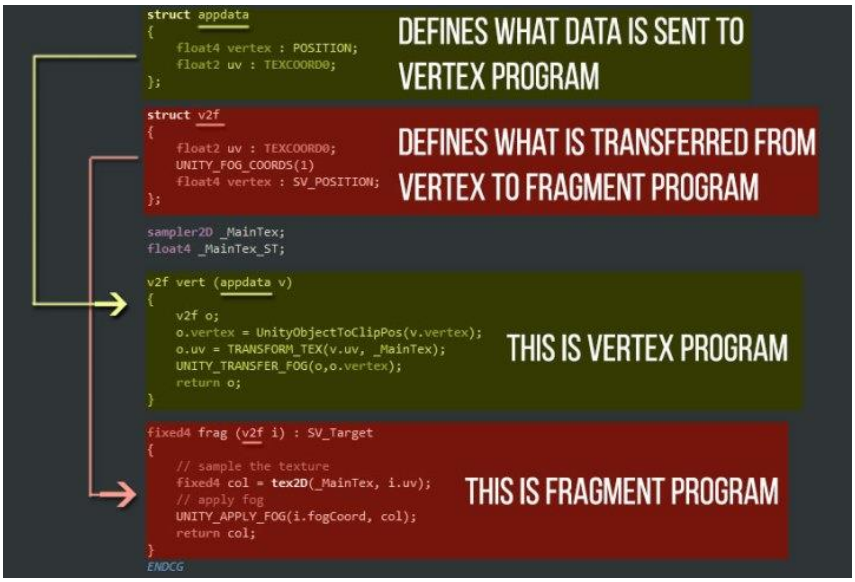
Erikoisefektit, kuten sade, savu, tuli ja pöly 3D-ympäristöissä luodaan partikkeleilla, jotka ovat suuri määrä pieniä neliöitä, jotka ovat aina kohtisuorassa kameraan päin. Näissä neliöissä on hienovaraiset tekstuurit, jotka käyttävät paljon läpinäkyvyyttä. Näitä pieniä tekstuureja luodaan ja tuhotaan sadoissa tuhansissa kappaleissa, jotta syntyy erikoisefekti ympäristöön. (Kuva 19.)

Yleensä partikkeleihin käytetään objekteja, joita kutsutaan partikkeliemittereiksi tai partikkeligeneraattoreiksi, jotka synnyttävät jatkuvasti uusia sprite-grafiikoita. [23.]



Kuva 19. Tuli-efektiä tuottava partikkeligeneraattori. [23.]

Partikkeleiden luomisessa käytetään sekä verteksi- että fragmenttivarjostimia. (Kuva 20.) Verteksivarjostin tekee laskelmat jokaiselle verteksille, mikä tapahtuu neljä kertaa per neliö. Fragmenttivarjostin laskee kaikki verteksit, jotka ovat yhdellä ruudun pikselillä. [24.]



Kuva 20. Partikkelitiedon kulku varjostimessa. [24.]

4 Visuaalisen tyylin saavuttaminen Unity-pelimoottorissa

Syksyllä 2019 olin mukana Kajaanin ammattikorkeakoulun kurssilla, missä ryhmien piti luoda valmis pelikonsepti. Olin johtavan graafikon asemassa art leadinä ryhmässämme, eli pelin laajempi ulkonäkö ja tyylin ohjaaminen oli minun vastuullani, mutta tyylin lopulliset päätökset olivat ryhmän pelisuunnittelijan käsissä. Projektin nimeksi tuli Hilja & The King of The Forest. Pelin tarina ja ympäristö sijoittui vanhanaikaiseen Suomeen, tuoden mukanaan suomalaista kansanuskoa ja myyttejä, kuten saunatonttuja ja metsänhenkiä. Päätimme käyttää Unity-pelimoottoria tämän projektin luomiseen.

4.1 Haluttu visuaalinen ulkonäkö

Alkupäässä lähdimme suunnittelijan idealla saada aikaan peli, joka yhdistää 3D- ja 2D-grafiikkaa. Pelin hahmot olisi mallinnettu ja animoitu 3D-mallintamisohjelmilla, mutta ympäristöt olisivat käsintehtyjä 2D-piirroksia. Malliesimerkkinä tälle toimi Ubisoftin peli Child of Light, joka sai paljon huomiota erikoisella graafisella tyylillään. Hahmoilla olisi myös malliensa ympärillä sarjakuvamainen ääriviiva. (Kuva 21.)

Kuva 21. Kuvakaappaus Child of Light –pelistä. [25.]

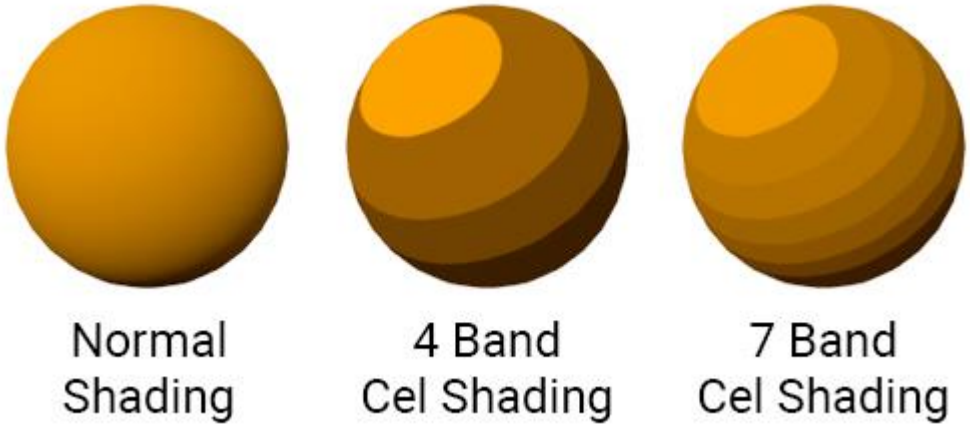
Inspiraationa ympäristön ulkonäköä varten meille osoitettiin internetsarjakuva nimeltä ”Stand Still. Stay Silent”. Graafiseltaan tyyliltään tämä vastaisi digitaalista vesivärimaalausta, joka vaatii vähemmän varjostimien käyttöä. Tulimme lopulta hylkäämään tämän idean ja päätimme käyttää myös ympäristössä 3D-malleja.

3D-mallien ulkonäölle halusimme samansuuntaista tyyliä kuin The Legend of Zelda: The Wind Waker-pelissä. (Kuva 22.) Tämä tarkoitti pelin grafiikkojen toteuttamista cel-shading-tekniikalla, jolla saisimme malleille kovat yksinkertaiset varjot, jotta peli näyttäisi piirretyltä animaatiolta. Lisätoiveena halusimme lisätä malleille myös kovat ääriviivat tehostaakseen piirroksen vaikutelmaa.

Kuva 22. Kuvakaappaus The Legend of Zelda: The Wind Waker HD –pelistä. [26.]

4.2 Cel-varjostimet ja ääriviivat

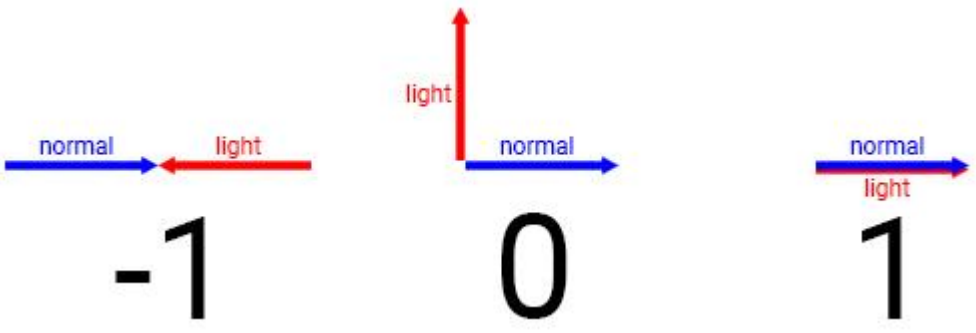
Cel-shading-varjostusmetodilla tarkoitetaan tapaa hahmottaa jotain käyttämällä useita värin juovia jatkuvan gradientin sijasta. (Kuva 23.) Ääriviivat malleissa eivät ole cel-varjostimella tuotettua, vaikka niitä käytetään usein yhdessä videopeliprojekteissa. [27.]



Kuva 23. Tavallinen varjostin ja kaksi cel-varjostinta. [27.]

Cel-varjostin on erinomainen metodi, jos haluaa emuloida 2D-elementtejä 3D-ympäristössä, ja sitä käytetäänkin jo laajalti sekä peli- että animaatioalalla.

Laajin käytetty metodi saada aikaan cel-varjostin on vertailla pinnan normaali ja valon suunta. (Kuva 24.) Esimerkiksi laskemalla pistetulon normaalin ja valon suunnan välillä saa tulokseksi arvon välillä -1 ja 1. Arvo -1 tarkoittaa, että pinta ja valo ovat vastasuuntaisia, 0 tarkoittaa, että ne ovat kohtisuorassa toistensa kanssa ja 1 tarkoittaa, että ne ovat samassa suunnassa. Asettamalla kynnysarvot pistetulolle voi hallita varjostimen tuottamien varjojen juovien määrää. [27.]



Kuva 24. Normaalin ja valon suunnan pistetuloja. [27.]

Ääriviivat saadaan 3D-malleihin esimerkiksi kopioimalla mallin ja kääntämällä kopion normaalit päinvastoin. Suurentamalla mallia hieman se tulee esille ja antaa vaikutelman ääriviivoista. Tämä tosin on aika kömpelö metodi, joka tuplaa verteksien määrän jokaiselle mallille, missä sitä käytetään. Laajemmassa käytössä on reunanilmaisu, joka määritetään pikseli- ja fragmenttivarjostimilla.

4.3 Unityn varjostimet

Unity-pelimoottorilla on omat sisäiset työkalut varjostimia varten, helpottaen sekä ohjelmoijien että graafikkojen työtaakkaa. Näiden työkalujen avulla voi käyttää Unityn perusvarjostimia, luoda uusia tai muokata olemassa olevia varjostimia joko pelimoottorin sisällä tai ohjelmointikoodia käsittelemällä, tuoden uudet muutokset heti käyttöön moottorissa.

Unityn moottorin sisäänrakennettu perusvarjostin itsessään tarjoaa heti laajan valikoiman toimintoja. Sitä voi käyttää hahmontamaan ”oikean maailman” esineitä kuten kiveä, puuta, lasia, muovia ja metallia, tukien laajaa määrää eri varjostintyyppejä ja yhdistelmiä.

Unityn hahmontamisen varjostimet on kirjoitettu pintavarjostimina, verteksi- ja fragmenttivarjostimina tai kiinteinä funktiovarjostimina. Varjostimen tyypistä riippumatta niiden koodi on rakennettu ShaderLab-kielen ympärille. [28.]

4.4 Uuden varjostimen luominen

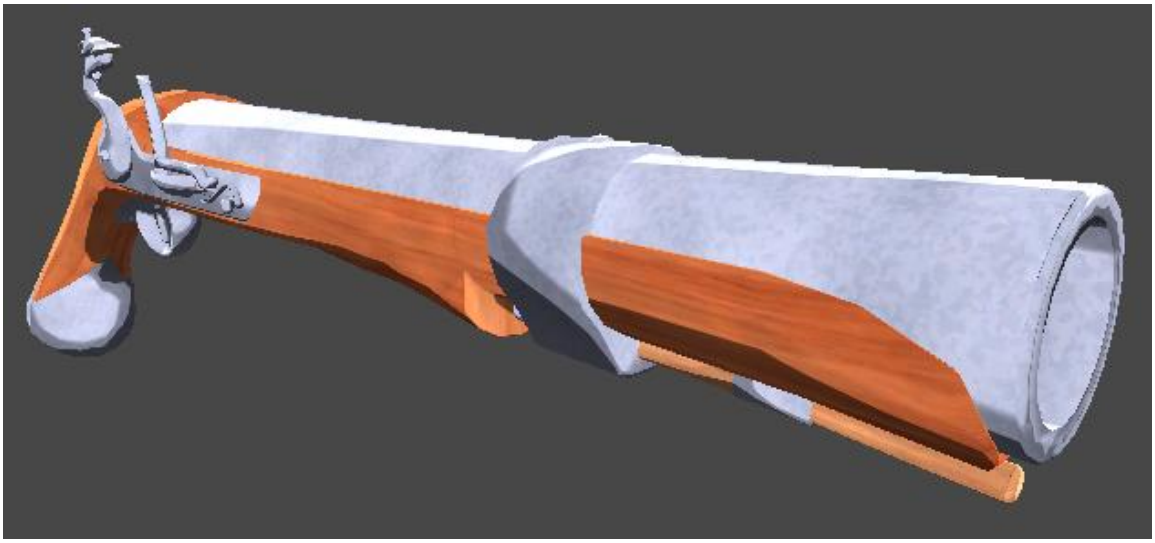
Peliimme haluttu graafinen tyyli vaati cel-varjostusta tukevan varjostimen, jota ei Unityn perusvarjostimella saa toteutettua. Käytän tämän varjostimen projektissa esimerkkinä itsetehtyä pistoolimalliani, jolla aloitin ensin kokeilemalla Unityn perusvarjostinta. (Kuva 25.)

Kuva 25. Unity-moottorin perusvarjostinta käyttävä materiaali 3D-mallissa.

Omalla ohjelmointikyvylläni en pystynyt luomaan varjostinta tyhjästä, joten käytin apunani eri kursseja internetissä.

4.5 Olemassa olevan varjostimen muokkaaminen

Ensimmäisenä yritin muokata Unity-moottorin perusvarjostimen asetuksia, mikä nopeasti paljastui hyödyttömäksi tavoitteessani. Lähdin tekemään uutta varjostinta internetissä löytämäni opastuksen avulla, jolla saisin piirretyn animaation näköisen tuotoksen. [29.] Tällä varjostimella sain aikaan kovat varjot mallilleni, mutta en ollut tyytyväinen valon erittäin vähäiseen vaikutukseen. (Kuva 26.)



Kuva 26. Varjostimen kokeilua materiaalissa.

Tähän asti kokeillut varjostimet eivät tuottaneet tyydyttäviä tuloksia, joten päätin kokeilla olemassa olevaa varjostinta. Latasin Unityn kaupasta ilmaisen varjostimen nimeltä MK Toon Free, jolla saa aikaiseksi hyvän cel-varjostuksen näkymän. (Kuva 27.) Tällä varjostimella sai materiaaleihin sekä normaalikarttoja että ääriviivat mallien ympärille. [30.]

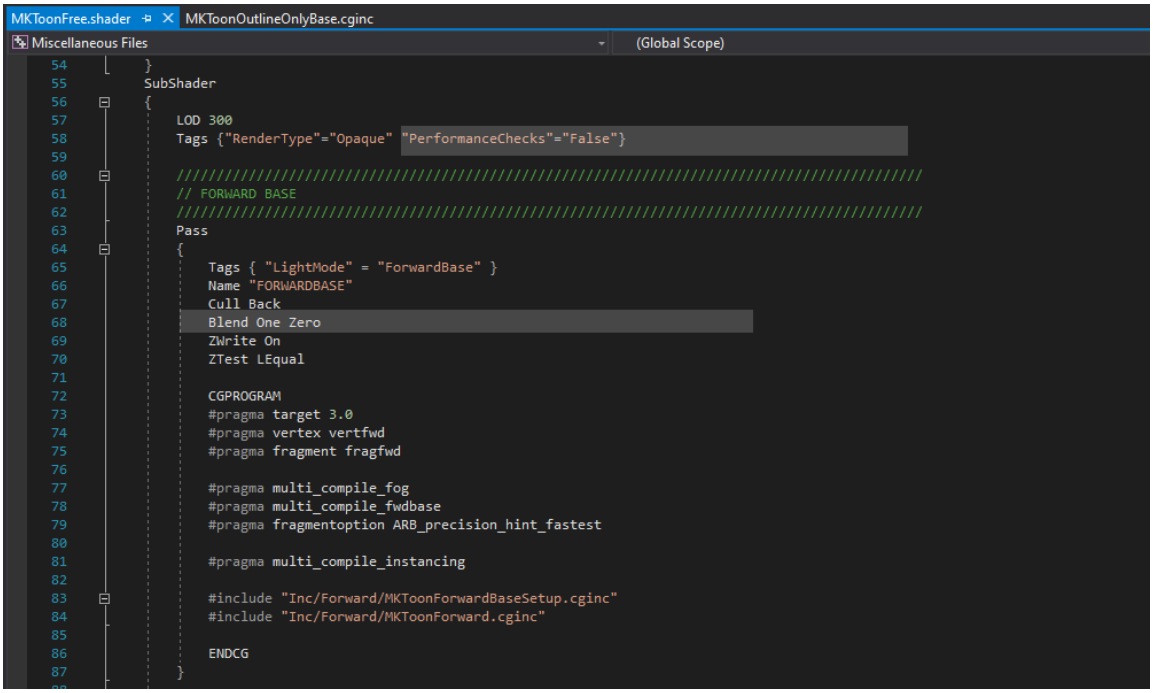
Kuva 27. MK Toon Free –varjostinta käyttävä materiaali.

Tämä varjostin täyttää useimmat vaatimukset halutulle tyylillemme, mutta yksi yksityiskohta puuttuu: Haluamme ääriviivojen värien vaihtelevan mallin värien mukaisesti, mutta tämä varjostin tarjoaa vain yhden säädettävän värin koko ääriviivalle. Minun tuli lähteä muokkaamaan varjostimen koodia saadakseni halutun tuloksen.

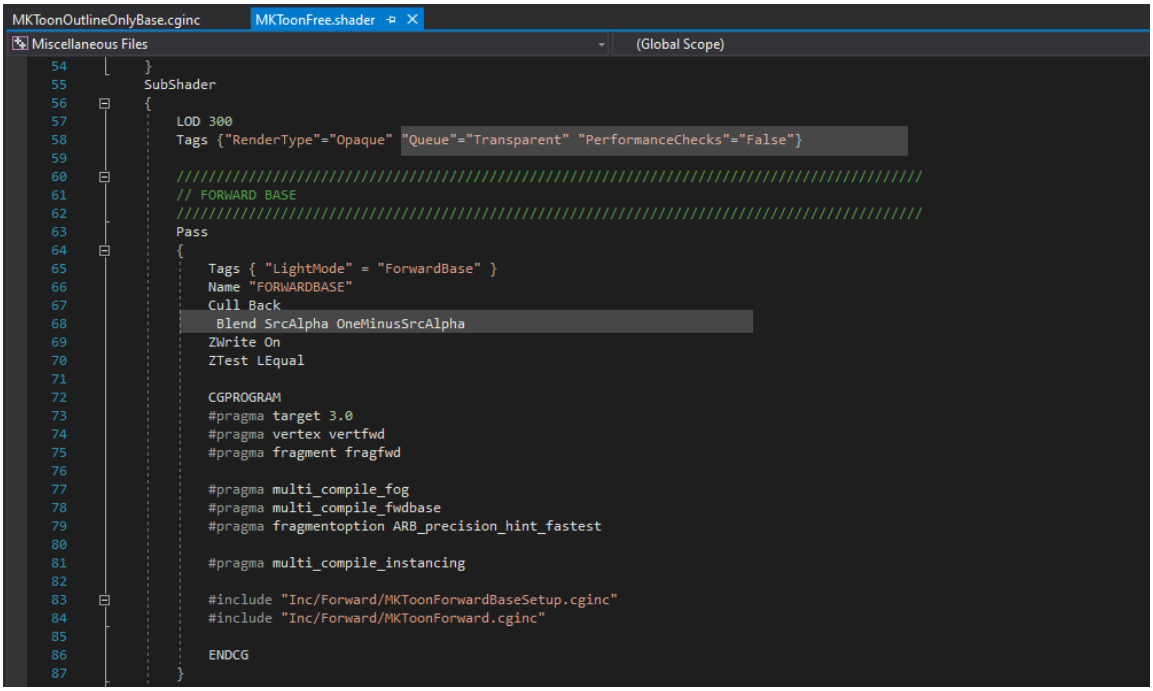
4.6 Lisäyksiä varjostimeen koodissa

Useiden kokeilujen jälkeen löysin osat varjostimien koodista, missä voin tehdä vähäisiä muutoksia saavuttaakseni automaattisen värityksen ääriviivoissa. Unity-moottorin kautta pääsin nopeasti muokkaamaan varjostimen koodia. Ensimmäisenä oli koko varjostimen pääluokka, missä muokkasin kahta koodin osaa: tageja ja Forward Base-osion Pass-metodia. (Kuva 28, 29.)

Tageihin lisäsin jonon läpinäkyvyydelle eli ”Queue”=”Transparent”-tagin. Pass-metodin Blend-kohtaa muokkasin SrcAlpha- ja OneMinusSrcAlpha-tekijöillä. Tämä muokkaus koodissa mahdollistaa ääriviivoille läpinäkyvyyden ja värin lisäyksen alpha-kerrokseen.

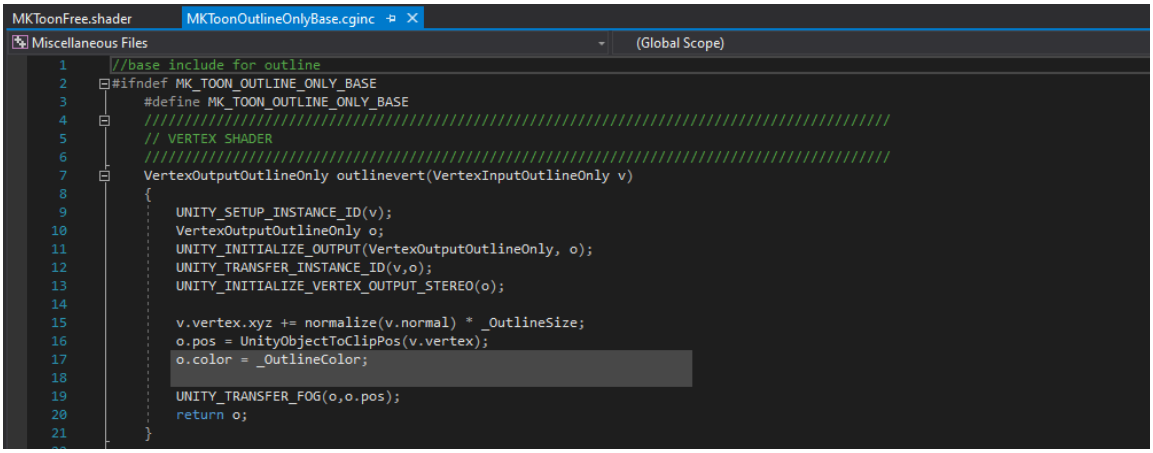


Kuva 28. Alkuperäistä koodia pääluokassa.

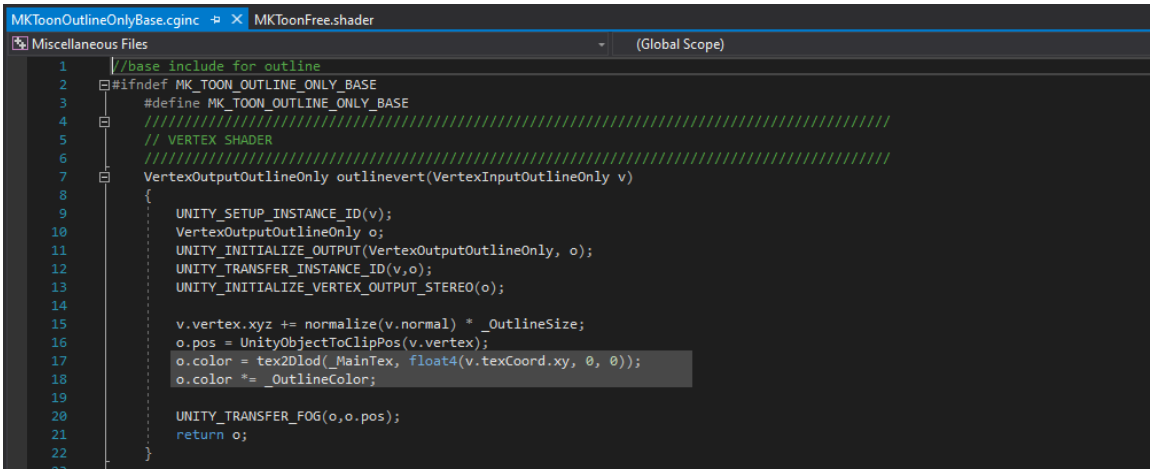


Kuva 29. Muokattua koodia pääluokassa.

Seuraavana minun tuli muokata MKToonOutlineOnlyBase-luokkaa, jossa pääsin ääriviivojen verteksivarjostimeen käsiksi. Lisäämällä tex2Dlod-funktion pystyin määrittämään varjostimen ottamaan värin ääriviivoille objektin tekstuurista. (Kuva 30, 31.) Samalla lisäsin float4-luokan määrittämään varjostimelle tekstuurikoordinaatit, mistä kohti tekstuuria värin tulisi ottaa.

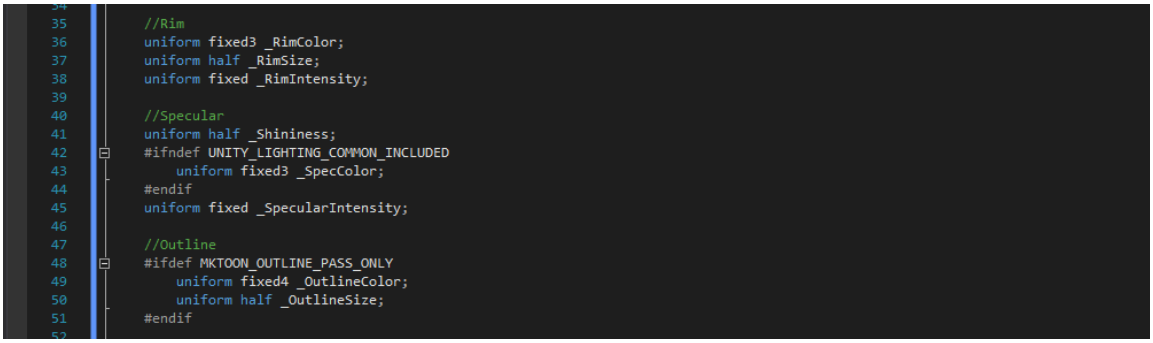


Kuva 30. Alkuperäistä koodia ääriviivojen verteksivarjostimessa.



Kuva 31. Muokattua koodia ääriviivojen verteksivarjostimessa.

Lisäsin vielä muihin luokkiin ”Queue”=”Transparent”-tagin, ennen kuin lähdin MKToonV-luokkaan, missä vaihdoin Outline-osiossa ääriviivavärin määritelmän fixed4-tagista float4-luokkaan, minkä aiemmin lisäsin ääriviivojen verteksivarjostimeen. (Kuva 32, 33.)



Kuva 32. Alkuperäistä koodia ääriviivavärille.

```
35      //Rim
36      uniform fixed3 _RimColor;
37      uniform half _RimSize;
38      uniform fixed _RimIntensity;
39
40      //Specular
41      uniform half _Shininess;
42      #ifndef UNITY_LIGHTING_COMMON_INCLUDED
43          uniform fixed3 _SpecColor;
44      #endif
45      uniform fixed _SpecularIntensity;
46
47      //Outline
48      #ifdef MKTOON_OUTLINE_PASS_ONLY
49          uniform float4 _OutlineColor;
50          uniform half _OutlineSize;
51      #endif
```

Kuva 33. Muokattua koodia ääriviivavärille.

Viimeisenä kävin MKToonOutlineOnlyIO-luokassa lisäämässä verteksivarjostimen syöttöön float3- ja float4-tagit tekstuurikoordinaateille ja niiden värille (Kuva 34, 35.)

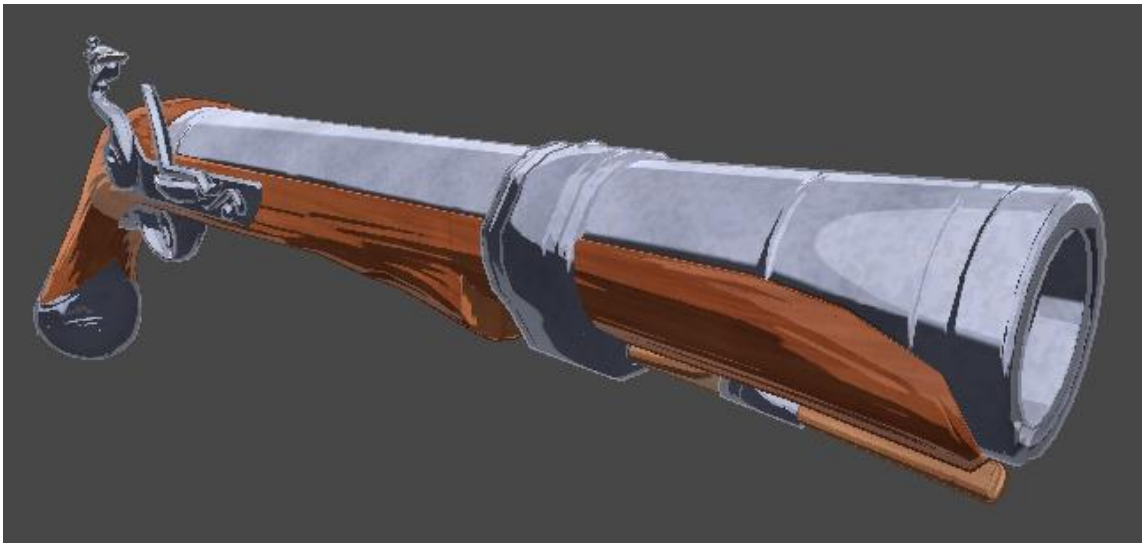
```
MKToonOutlineOnlyIO.cginc*  MKToonV.cginc  MKToonFreeOutlineOnly.shader  MKToonFree.shader  MKToonOutlineOnlyBase.cginc
Miscellaneous Files                                                    (Global Scope)
1   //Vertexshader Input and Output
2   #ifndef MK_TOON_OUTLINE_ONLY_IO
3       #define MK_TOON_OUTLINE_ONLY_IO
4       ////////////////////////////////////////////////////////////////////////////////////////
5       // INPUT
6       ////////////////////////////////////////////////////////////////////////////////////////
7       struct VertexInputOutlineOnly
8       {
9           float4 vertex : POSITION;
10          half3 normal : NORMAL;
11          UNITY_VERTEX_INPUT_INSTANCE_ID
12      };
```

Kuva 34. Alkuperäistä koodia verteksivarjostimen syötölle.

```
MKToonOutlineOnlyIO.cginc  MKToonV.cginc  MKToonFreeOutlineOnly.shader  MKToonFree.shader  MKToonOutlineOnlyBase.cginc
Miscellaneous Files                                                    (Global Scope)
1   //Vertexshader Input and Output
2   #ifndef MK_TOON_OUTLINE_ONLY_IO
3       #define MK_TOON_OUTLINE_ONLY_IO
4       ////////////////////////////////////////////////////////////////////////////////////////
5       // INPUT
6       ////////////////////////////////////////////////////////////////////////////////////////
7       struct VertexInputOutlineOnly
8       {
9           float4 vertex : POSITION;
10          half3 normal : NORMAL;
11          float3 texCoord : TEXCOORD0;
12          float4 color : TEXCOORD1;
13          UNITY_VERTEX_INPUT_INSTANCE_ID
14      };
```

Kuva 35. Muokattua koodia verteksivarjostimen syötölle.

Tällä vähäisellä koodin muokkauksella sain aikaiseksi malleille ääriviivat, joiden jokainen pikseli ottaa itselleen värin osasta tekstuuria, missä lähin nykyisestä näkökulmasta katsottu verteksi sijaitsee. (Kuva 36.)



Kuva 36. Automaattisesti väritetyt ääriviivat 3D-mallin materiaalissa.

4.7 Varjostinta käyttävät 3D-mallit ja testaaminen

Kun olin tyytyväinen varjostimen tuloksiin kokeiluun tarkoitetussa pistoolin 3D- mallissa, oli aika soveltaa sitä peliin tulevissa verkkolankamalleissa. Sain aikaiseksi tyydyttäviä 3D-malleja, joilla on graafiset tavoitteet saatu tyydyttävästi täytettyä. Ääriviivat eivät käyttäydy täydellisesti tietyissä mallien kohdissa, kuten silmien alueella ja lähekkäin olevien osien kesken, mutta varjot ja niiden käyttäytyminen normaalikarttojen kanssa on moitteeton. (Kuva 37, 38.)

Kuva 37. Kettumalli uudella varjostimella.

Kuva 38. Tonttumalli uudella varjostimella, käyttäen normaalikarttoja.

5 Yhteenveto

Oppimiseni varjostimien toiminnasta alkoi käytännön vaiheessa, kun työstin peliprojektia. Lähdin siitä pisteestä taaksepäin teoriaan tätä opinnäytetyötä aloittaessani, ja tämä oppimistapa paljastui toimivaksi. Onnistuin kehittämään omia grafiikan tietojani ja taitojani, saaden hyvän pohjan itselleni tulevaisuuden haasteille, sillä pyrin graafikkona koettamaan monta eri puolta pelien visuaalisen tuotannon liukuhihnalta.

Koskin varjostimien toimintaa tässä työssä pinnallisella ja yksinkertaisella tasolla, mutta aikeeni oli aiheen syvän sisäistyksen sijasta sen ymmärryksen tavoittaminen aloittelevalle grafiikan tuottajalle, jolle tekninen taide voi tuntua uhkaavalta esteeltä. Aloitinkin tämän työn sillä ajatuksella, että itse saisin parempaa pintatason ymmärrystä varjostimista, ja onnistuin omalta kannaltani.

Lähteet

1. Omar Shehata. (2015). A Beginner's Guide to Coding Graphics Shaders. Saatavilla 30.4.2020. https://gamedevelopment.tutsplus.com/tutorials/a-beginners-guide-to-coding-graphics-shaders--cms-23313

2. Mike Jacobs, Michael Satran. (2018). Graphics Pipeline. Saatavilla 30.4.2020. https://docs.microsoft.com/en-us/windows/win32/direct3d11/overviews-direct3d-11-graphics-pipeline

3. OpenGL Wiki. Rendering Pipeline Overview. Saatavilla 30.4.2020. https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview

4. Vierge Marie, Pater McFly. Graphics Pipeline. Saatavilla 20.5.2020. https://en.wikipedia.org/wiki/File:Graphics_pipeline_2_en.svg

5. Kyle Sloka-Fey. (2013). Let's Build a 3D Graphics Engine: Rasterizing Line Segments and Circles. Saatavilla 30.4.2020. https://gamedevelopment.tutsplus.com/tutorials/lets-build-a-3d-graphics-engine-rasterizing-line-segments-and-circles--gamedev-8414

6. UFO 3D. History of 3D Modeling: From Euclid to 3D Printing. Saatavilla 18.4.2020. https://ufo3d.com/history-of-3d-modeling

7. Ranjit Menon. (2011). first ever 3d animation (40 year old 3d computer graphics pixar 1972). Saatavilla 18.4.2020. https://www.youtube.com/watch?v=T5seU-5U0ms

8. Jesse Dunietz. (2016). The Most Important Object In Computer Graphics History Is This Teapot. Saatavilla 18.4.2020. http://nautil.us/blog/the-most-important-object-in-computer-graphics-history-is-this-teapot

9. OpenGL Wiki. History of Programmability. Saatavilla 18.4.2020. https://www.khronos.org/opengl/wiki/History_of_Programmability

10. Id Software. (1992). Wolfenstein 3D. Saatavilla 20.5.2020. https://www.v2.fi/uutiset/pelit/31524/Yli-neljannesvuosisadan-se-otti-mutta-Wolfenstein-3D-on-nyt-sallittu-Saksassa/

11. Id Software. (1993). Doom Classic. Saatavilla 20.5.2020. https://www.techspot.com/news/81172-classic-doom-trilogy-gets-modern-console-re-release.html

12. Rory Milne. (2019). The making of Quake 2. Saatavilla 18.4.2020. https://www.pcgamer.com/the-making-of-quake-2/

13. Lighthouse3d. GLSL Tutorial – Color Example. Saatavilla 18.4.2020. https://www.lighthouse3d.com/tutorials/glsl-tutorial/color-example/

14. Lighthouse3d. GLSL Tutorial – Hello World. Saatavilla 20.5.2020. https://www.lighthouse3d.com/tutorials/glsl-tutorial/hello-world/

15. Scratchapixel. What is Shading: Light-Matter interaction. Saatavilla 19.4.2020. https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-shading/what-is-shading-light-matter-interaction

16. Lighthouse3d. GLSL Tutorial – Lighting. Saatavilla 19.4.2020. https://www.lighthouse3d.com/tutorials/glsl-tutorial/lighting/

17. Eric W Weisstein. Normal Vector. Saatavilla 24.4.2020. https://mathworld.wolfram.com/NormalVector.html

18. Scratchapixel. Normals, Vertex Normals and Facing Ratio. Saatavilla 25.4.2020. https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-shading/shading-normals

19. Joey de Vries. (2015). Normal Mapping. Saatavilla 25.4.2020. https://learnopengl.com/Advanced-Lighting/Normal-Mapping

20. Pluralsight. (2014). Elliminate Texture Confusion: Bump, Normal and Displacement Maps. Saatavilla 24.4.2020. https://www.pluralsight.com/blog/film-games/bump-normal-and-displacement-maps

21. Jonathan Lampel. (2017). Normal vs. Displacement Mapping & Why Games Use Normals. Saatavilla 20.5.2020. https://cgcookie.com/articles/normal-vs-displacement-mapping-why-games-use-normals

22. Ronja Böhringer. (2018.) Basic Transparency. Saatavilla 20.5.2020. https://www.ronja-tutorials.com/2018/04/06/simple-transparency.html

23. Joey de Vries. (2015). Particles. Saatavilla 28.4.2020. https://learnopengl.com/In-Practice/2D-Game/Particles

24. Michał Piątek. (2017). Unity3D Particle Shaders – The simplest shader. Saatavilla 28.4.2020. http://michalpiatek.com/2017/06/08/unity3d-particle-shaders-the-simplest-shader/

25. Ubisoft. (2014.) Child of Light. Saatavilla 20.5.2020. https://www.ubisoft.com/en-gb/game/child-of-light/

26. Nintendo. (2013). The Legend of Zelda – The Wind Waker. Saatavilla 20.5.2020. https://www.nintendo.com/games/detail/the-legend-of-zelda-the-wind-waker-hd-wii-u/

27. Tommy Tran. (2018). Unreal Engine 4 Cel Shading Tutorial. Saatavilla 25.4.2020. https://www.raywenderlich.com/146-unreal-engine-4-cel-shading-tutorial

28. Unity Technologies. Standard Shader. Saatavilla 24.3.2020 https://docs.unity3d.com/Manual/shader-StandardShader.html

29. Erik Roystan Ross. (2019). Toon Shader. Saatavilla 28.4.2020. https://roystan.net/articles/toon-shader.html

30. Michael Kremmel. (2019). MK Toon Free. Saatavilla 24.3.2020. https://assetstore.unity.com/packages/vfx/shaders/mk-toon-free-68972