Tampereen ammattikorkeakoulu

Mobiilialustalle sopiva vesivarjostin

Case Transit King Tycoon

Lahtinen Jarno

OPINNÄYTETYÖ
Marraskuu 2020

Tietojenkäsittely
Pelituotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Pelituotanto

LAHTINEN, JARNO:
Mobiilialustalle sopiva vesivarjostin
Case Transit King Tycoon

Opinnäytetyö 38 sivua
Marraskuu 2020

Tämän opinnäytetyön toimeksiantajana toimi BON Games Oy, joka on pelialan yritys Tampereella. Opinnäytetyössä kehitettiin toimeksiantajan julkaistuun Transit King Tycoon -peliin mobiilialustoilla toimiva ja optimoitu vesivarjostin.

Projektissa havaittiin kohdelaitteilla suorituskykyongelmia, ja yhtenä ongelmakohtana todettiin olevan sen hetkinen vesivarjostin. Tämä ongelma haluttiin korjata, ja siitä syntyi opinnäytetyölle aihe. Opinnäytetyön tavoitteeksi tuli optimaalisemman vesivarjostimen kehittäminen aiemman vesivarjostimen tilalle Transit King Tycoon -peliin.

Tarkoituksena oli selvittää varjostimien optimointitekniikoita sekä yleisiä hyväksi todettuja käytäntöjä varjostinkehityksessä. Lisäksi etsittiin tietoa siitä, miten kohdelaitteiden suorituskykyä voidaan profiloida käyttäen profilointityökaluja.

Varjostimen kehitykseen etsittiin tietoa kirjallisuudesta ja verkosta esimerkeistä. Kirjallisuudesta haettiin tietoa siitä, miten varjostimet ja näytönohjaimet toimivat. Lisäksi selvitettiin yleisiä hyväksi todettuja varjostimien optimointitekniikoita.

Opinnäytetyön lopputuloksena saatiin uusi, paremman näköinen ja suorituskyvyn kannalta optimaalisempi vesivarjostin julkaistuun mobiilipeliin. Suorituskyvyn parantuminen pystyttiin todentamaan käyttäen profilointityökaluja.

Asiasanat: varjostin, unity, mobiilialusta, peli, grafiikka, profilointi

ABSTRACT

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

LAHTINEN, JARNO:
Creating a Water Shader Suitable for Mobile Platform
Case Transit King Tycoon

Bachelor's thesis 38 pages
November 2020

This thesis was commissioned by BON Games Oy, which is a gaming company in Tampere. In this thesis, the goal was to design and develop an optimized water shader for the mobile game Transit King Tycoon published by the client.

The purpose of this thesis was to research shader optimization techniques and good practices in shader development. In addition, research was done regarding shader and performance profiling on target devices using profiling tools.

Information on shader development was sought from literature and from example shaders available in the Internet. The literature was used to gather information on shaders, graphics cards and the best principles regarding shader optimization.

The result of the thesis was a new, improved and more optimized water shader for a published mobile game. Improvement in performance was verified by using profiling tools.

Key words: shader, unity, mobile platform, games, graphics, profiling

SISÄLLYS

ERITYISSANASTO

Cg	Korkean tason C kieleen pohjautuva varjostinohjelmointikieli
Unity	Pelimoottori
Asetelma	Unity asetelma tai työskentely-ympäristö
Polygoniverkko	3D mallin pisteistä muodostettu pisteverkko
Nurkkapiste	Polygoniverkoston piste.
Kehys	Ruudulle piirrettävä kuva
Varjostin	Tietokoneohjelma, jolla sävytetään 3D mallit
Renderöintiputki	Näytönohjaimella tapahtuvat askeleet kuvan piirtämiseksi näytölle
Piirtokutsu	Näytönohjaimelle lähetettävät ohjeet mallin piirtämiseksi
Kuvataajuus	Kuvien piirtotaajuus
Haaroitus	Ohjelman koodissa tapahtuva haaroittuminen ehtolauseissa
Pisteväritys	3D mallin pisteiden väriarvot

1 JOHDANTO

Transit King Tycoon on verkossa pelattava mobiilipeli, jossa pelaajan tarkoituksena on edetä logistiikkauralla ja kehittää menestyvä logistiikkayritys. Peliä kehittää tamperelainen pelialan yritys BON Games Oy, joka on opinnäytetyön toimeksiantajana.

Projektissa todettiin olevan suorituskykyongelmia ja yhtenä ongelmakohtana havaittiin olevan vesivarjostin. Tästä syntyi opinnäytetyölle aihe ja samoin tavoite.

Opinnäytetyön tavoitteena oli kehittää uusi optimaalisempi vesivarjostin Transit King Tycoon peliin. Tarkoituksena oli selvittää, miten varjostimia optimoidaan sekä miten vesivarjostimia yleensä kehitetään mobiilialustalle. Lisäksi etsittiin tietoa siitä, miten kohdelaitteiden suorituskykyä voidaan profiloida käyttäen Unity -pelimoottorin omia työkaluja.

Työssä etsittiin tietoa varjostimista sekä niiden optimoinnista kirjallisuudesta ja tämän lisäksi etsittiin varjostinesimerkkejä verkosta vesivarjostimen toteuttamiseksi. Esimerkit käytiin läpi ennen varsinaisen toteutustavan valintaa, jotta optimoinnin tavoitteet saavutettiin mahdollisimman varmasti.

Opinnäytetyössä käytiin läpi projektin lähtötilanteen vesivarjostin. Sitä analysoitiin suorituskyvyn ja visuaalisuuden näkökulmista, jonka jälkeen sen suorituskyky profiloitiin. Tämä toimi uuden varjostimen suunnittelun lähtökohtana. Työssä käytiin läpi uuden vesivarjostimen toteutus, jonka jälkeen se profiloitiin ja sen suorituskykyä verrattiin aiempaan vesivarjostimeen.

2 TIETOKONEGRAFIIKKA

2.1 Yleisesti

Tietokonegrafiikka terminä käsittää grafiikan, joka luodaan tietokoneella. Tarkemmin ilmaistuna, se pitää sisällään grafiikan esityksen ja manipulaation tietokoneella. Tietokonegrafiikan kehitys on auttanut tietokoneiden grafiikkadatan käsittelyä, ymmärrystä sekä esittämistä. Alan kehitys on myös mullistanut animaatioiden, elokuvien sekä pelien teollisuuden. (Davis 2011, 7.)

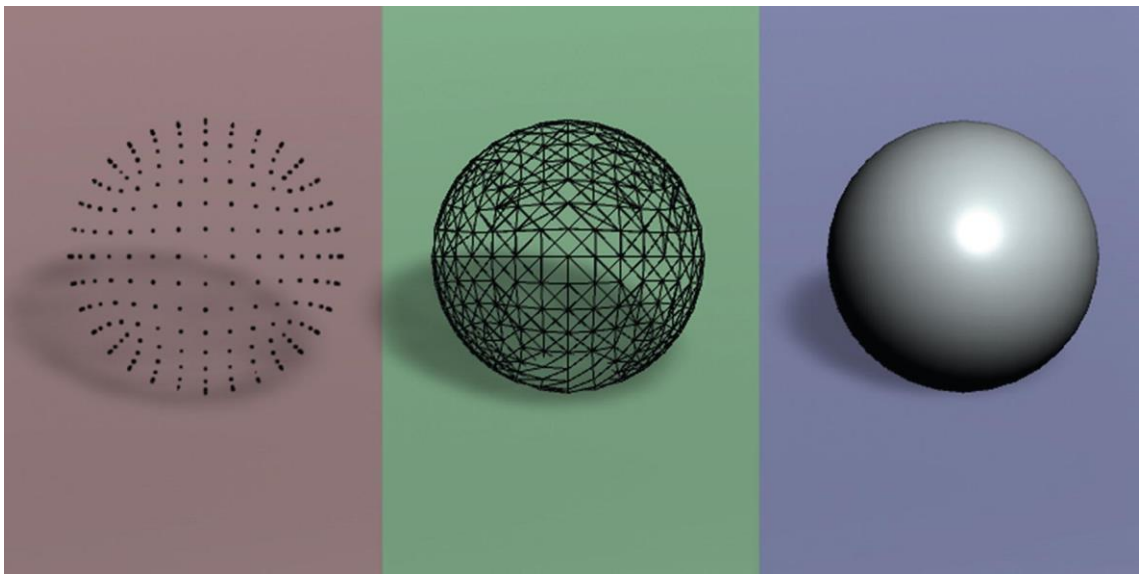
2.2 Renderöinti

Renderöinti on tietotekniikassa termi, joka tarkoittaa kuvan piirtämistä näytölle 2D- ja 3D-malleista. Asetelmassa oleva kamera määrittää kuvassa olevat mallit, valaistukset, katselusijainnin sekä -suunnan. Syntyneestä kuvasta käytetään termiä kehys tai piirros. (Halladay 2019) Yleisesti peleissä näitä kehyksiä piirretään näytölle useita kymmeniä tai jopa satoja kertoja sekunnissa.

2.3 Polygoniverkko

Polygoniverkko on tapa kertoa tietokoneelle tietoa 2D tai 3D mallin muodosta. Muodon kertomiseksi polygoniverkkoon on vähintään kirjoitettava tieto muodossa olevista nurkkapisteistä, reunoista sekä pinnoista. (Halladay 2019.)

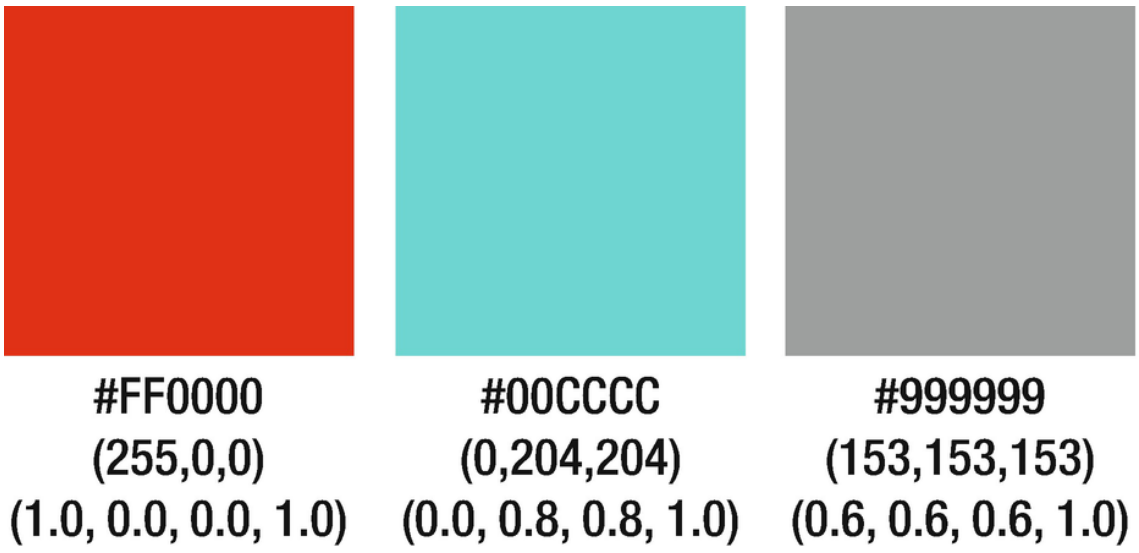
Nurkkapisteet ovat polygoniverkoston pisteitä kolmiulotteisessa avaruudessa. Näitä yhdistävät reunat, jotka määrittelevät kuinka polygoniverkoston muoto muodostuu. Pinnat muodostuvat kolmesta tai useammasta reunasta. Kuvassa 1 on havainnoituna nurkkapisteet vasemmalla puolella, keskellä on esitettynä niihin muodostetut reunat ja oikealla puolella on ilmaistuna reunoista täytetyt pinnat, joka saa kappaleen näyttämään kokonaiselta. (Halladay 2019.)



KUVA 1. Esimerkki polygoniverkoston osista (Halladay 2019.)

2.4 Värien käsittely näytönohjaimella

Yleisesti ohjelmoinnissa väriarvot ilmoitetaan kolmella muuttujalla tai värikanavalla. Värikanavoihin kuuluvat punainen, vihreä ja sininen, ja niitä voidaan ilmaista heksadesimaalein, kokonaisluvuin alueen 0 ja 255 välillä, tai liukuluvulla 0 ja 1 välillä. (Halladay 2019.) Kuvassa 2 on esitettyinä esimerkein näiden eri ilmaisutapojen erot.



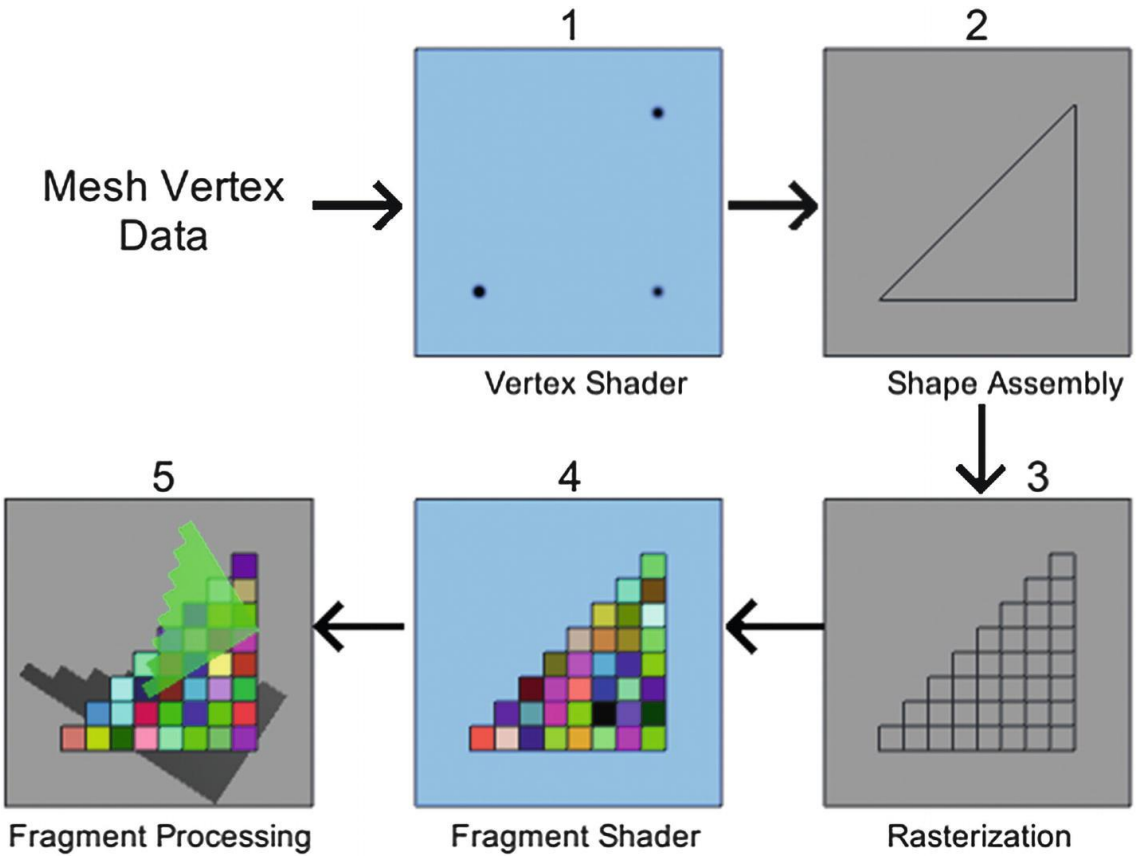
KUVA 2. Väriarvojen ilmaisu esimerkein (Halladay 2019.)

Unityn varjostimissa värikanavoita käsitellään liukuluvuilla, mutta tässä tapauksessa sinne lisätään neljäs arvo viimeiseksi, joka merkitsee värin läpinäkyvyyttä. Tämä arvo määrittelee sen, kuinka läpikuultava kyseinen malli tulee olemaan. (Halladay 2019.)

Tietokonegrafiikassa ja myöskin Unityssä, värit tallennetaan käyttäen vektoria, jossa on neljä komponenttia. Komponenteissa X, Y ja Z tallennetaan värikanavien punaisen, vihreän sekä sinisen väriarvot ja W-komponenttiin tallennetaan värin läpikuultavuuden arvo. Väriarvoja käsiteltäessä, näitä komponentteja X, Y, Z ja W voidaan myös kutsua R-, G-, B- ja A-komponenteiksi koodin selventämisen vuoksi. (Halladay 2019.)

2.5 Renderöintiputki

Renderöintiputki on termi, joka kuvastaa näytönohjaimella tapahtuvia vaiheita, jotta se pystyy piirtämään 3D-mallin näytölle. Se on tärkeä osa varjostinkehittämistä, sillä varjostimet toimivat ja muuttavat arvoja tässä renderöintiputkessa. (Halladay 2019.) Yksinkertaistettu malli renderöintiputkesta on esitettynä kaaviossa 1.



KAAVIO 1. Yksinkertaistettu renderöintiputki (Halladay 2019.)

Renderöintiputkea ajetaan ainoastaan näytönohjaimella, ja se ajetaan jokaiselle 3D mallille, joka pelikameran mukaan tulee olemaan näytöllä. Näytönohjaimet on suunniteltu toistamaan koodia nopeasti sen vuoksi, että renderöintiputki täytyy suorittaa mahdollisimman nopeasti. (Halladay 2019.)

Ensimmäisenä renderöintiputkessa on pisteohjelma, joka on esitettynä kaaviossa 1 kohdassa 1. Sen tehtävänä on laskea mallin pisteiden sijainti, jonka se muuntaa muun muassa pelikameran sijainnin ja katselusuunnan avulla ruudulla esitettäväksi sijainniksi. (Halladay 2019.)

Pisteohjelman suorituksen jälkeen, seuraavana on mallin muodon rakentaminen. Tässä vaiheessa pisteiden väliin muodostetaan viivat, eli käytännössä rakennetaan mallin reunat. (Halladay 2019.)

Rasterointi tapahtuu muodon rakentamisen jälkeen. Tässä vaiheessa näytönohjain selvittää mitkä mahdolliset näyttöruudun pikselit renderöitävä malli saattaa täyttää. Jokaiselle mahdolliselle pikselille näytönohjain rakentaa fragmentin, joka

on datarakenne ja se sisältää kaiken tarpeellisen informaation kyseisen pikselin piirtämiseksi. Rasterointivaiheessa ei kuitenkaan ole tietoa siitä, millainen piirrettävän mallin pinta tulee olemaan, vaan sen määrittämiseksi tarvitaan pikseliohjelmaa. (Halladay 2019.)

Pikseliohjelma suoritetaan rasterointivaiheen jälkeen. Ohjelman tarkoituksena on värittää rasterointivaiheesta saadut pikselit mallin pinnan mukaiseksi. (Halladay 2019.)

Viimeisenä vaiheena on fragmenttien prosessointi, jossa on kaksi tärkeää vaihetta: fragmentin testaus sekä sekoitusoperaatiot. Ensimmäisessä vaiheessa määritetään mitkä fragmentit päätyvät näyttöruudulle ja mitkä eivät. Fragmentit muodostetaan ilman tietoa asetelmasta, joten lopputuloksena näytönohjain yleensä muodostaa huomattavasti enemmän fragmentteja, kuin mitä näytössä on pikseleitä. Fragmentin sekoitusoperaatioissa mahdollistetaan kahden tai useamman eri mallin värien yhdistämisen, jotta saadaan lopputulokseksi läpikuultava materiaali. (Halladay 2019.)

2.6 Varjostimet

Varjostin-termi on syntynyt siitä, että niitä alkuperäisesti käytettiin pääosin simuloimaan realistista valaistusta 3D malleissa. Nykyään varjostimet pystyvät käsittelemään huomattavasti enemmän kuin ainoastaan valaistusta ja varjoja. Niillä pystyy muuttamaan täysin 3D mallin muotoa tai sitä, miltä se tulee lopuksi näyttämään laitteen näytöllä. (Thorn ym. 2018.)

Varjostimet ovat pieniä ohjelmia, joita ajetaan laitteen näytönohjaimella. Varjostimet eroavat yleensä kirjoitetuista skripteistä ja ohjelmista, mikä mahdollistaa varjostimien erityiset käyttötarkoitukset. Tämä ominaisuus toimii myös rajoittavana tekijänä; varjostimilla ei ole mahdollista tehdä kaikkea mitä tavallisilla skripteillä voidaan tehdä. (Halladay 2019.)

Varjostimien käyttö mahdollistaa renderöintiputken tiettyjen osien manipuloimisen halutun näköisen lopputuloksen saamiseksi. Suurin osa renderöintiputkesta

pysyy kuitenkin ennallaan välittämättä siitä, millaista mallia ollaan piirtämässä. (Halladay 2019.)

Renderöintiputken eri osissa ajettavilla varjostinohjelmilla on omat nimensä. Yleisimmin käytettyihin varjostinohjelmiin kuuluvat pisteohjelma sekä pikseliohjelma. Näillä ohjelmilla on pienimmät mahdolliset vaatimukset, jotta malli saadaan renderöityä näytölle. Jokainen vaihe renderöintiputkessa on erilainen, joten ei ole mahdollista käyttää esimerkiksi pisteohjelman tekemään pikseliohjelman prosessointia. (Halladay 2019.)

Varjostimet käyttävät omia varjostinohjelmointikieliään, joita on runsaasti. Perusperiaatteet näillä on kuitenkin hyvin samankaltaisia, joten yhden varjostinkielen opettelun jälkeen toiset varjostinkieletkään eivät tunnu enää vierailta. (Halladay 2019.)

3 VARJOSTINOPTIMOINNIN YLEISIÄ OHJEITA

3.1 Yleisesti

Varjostimien optimoinnissa voidaan käyttää tehokkaiksi todettuja optimoinnin yleisiä perusperiaatteita, jotka ovat hyödyllisiä alustasta riippumatta. Profilointi on varjostinoptimoinnissa erittäin tärkeää, sillä se on hyvin laaja alue. Optimointimetodit riippuvat alustasta, jolle applikaatio on tehty. (Halladay 2019.)

3.2 Laskentojen siirto pisteohjelmaan

Yleisesti kehystä piirrettäessä piirretään huomattavasti enemmän fragmentteja kuin pisteitä. Tämän periaatteen mukaan voidaan olettaa, että siirtämällä laskutoimituksia pikseliohjelmasta pisteohjelmaan ja siirtämään niistä saadut tulokset takaisin pikseliohjelmaan, pystytään säästämään näytönohjaimella tapahtuvia laskentatoimituksia. Tämä metodi on hyödyllinen varsinkin niissä tapauksissa, joissa on mahdollista siirtää trigonometrisia funktioita laskettavaksi pisteohjelmaan. Joissakin tapauksissa saattaa olla myös hyödyllistä lisätä mallien pistemäärää mahdollistaen laskentojen siirtämisen pisteohjelmaan. (Halladay 2019.)

3.3 Haaroitus

Piirtokutsua prosessoidessaan näytönohjaimen täytyy suorittaa laskenta kaikille pisteille sekä fragmenteille, jotka tullaan kirjottamaan siinä piirtokutsussa. Näytönohjaimilla on omat optimointimekaniikat, jotka yhdistävät ruudulla olevia läheisiä pisteitä ja fragmentteja omiin ryhmiinsä ja mahdollistavat näiden ryhmien laskentojen suorittamisen samanaikaisesti. Kyseinen järjestelmä toimii silloin parhaiten, kun varjostin tekee samat laskennat jokaiselle pisteelle ja fragmentille. Tämä johtaa siihen, että yleisesti ehtolauseita ei suositella tehtäväksi varjostimille. Mikäli ehtolausetta käytetään, puhutaan haaroituksesta. (Halladay 2019.)

Haaroitusta on kahta eri tyyppiä: yhtenäinen haaroitus ja dynaaminen haaroitus. Yhtenäinen haaroitus tarkoittaa käytännössä sitä, että haaroittuminen johtuu muuttujasta, joka on yhtenäinen varjostimessa. (Halladay 2019.) Kuvassa 3 on esitettynä esimerkki yhtenäisestä haaroituksesta varjostinkoodissa. Tässä tapauksessa, varjostimen pisteohjelma käyttää samaa haaroitusreittiä, sillä muuttuja on sama jokaisella pisteohjelman suorituskerralla, joten edellä mainittu optimointijärjestelmä pystyy käyttämään samoja laskentoja jokaiselle ryhmitetylle pisteelle (Halladay 2019).

```
Properties {
    [Toggle] _Variable ("Variable", Float) = 0.0
}

            ...

        v2f vert(appdata v)
        {
            if (_Variable > 0) {
                // Do something
            }

        }
```

KUVA 3. Yhtenäinen haaroitus.

Yhtenäisen haaroituksen vastakohtana on dynaaminen haaroitus. Se tapahtuu, kun ehtolause riippuu muuttuvasta muuttujasta, kuten esimerkiksi tekstuurista tai pisteen sijainnista pelimaailmassa. Tämän tyyppinen haaroitus estää edellä mainitun näytönohjaimen optimointijärjestelmän tehokkaan toimivuuden, sillä se ei pysty enää suorittamaan laskentoja yhtenäisillä laskentakaavoilla. (Halladay 2019) Kuvassa 4 on nähtävillä esimerkki dynaamisesta haaroituksesta.

```
// Vertex input
struct appdata
{
    half4 vertex : POSITION;
};

// Vertex output
struct v2f
{
    half4 worldPos : TEXCOORD1;
};


// Vertex program
v2f vert (appdata v)
{
            ...

    o.pos = UnityObjectToClipPos(v.vertex);
    o.worldPos = mul(unity_ObjectToWorld, v.vertex);

            ...
}

// Fragment program
fixed4 frag (v2f i) : SV_Target
{
            ...

    if (i.worldPos.y > 5) {

        // Do something

        }
    }
```

KUVA 4. Dynaaminen haaroitus.

Dynaaminen haaroitus kuitenkin saattaa olla optimointimenetelmä. Tämä riippuu siitä, kuinka paljon koodia on sijoitettuna ehtolausekkeen sisään ja onko se yhdenmukaista. Mitä enemmän koodia sijoitetaan ehtolausekkeen sisään, sen parempi se on suorituskyvyn kannalta. Yhdenmukaisuudella tarkoitetaan sitä, että kuinka todennäköistä on, että ainoastaan yhtä haaraa käytetään optimointijärjestelmän ryhmän keskuudessa. (Halladay 2019.)

3.4 Kerto- ja lisäyslaskennat

Kerto- ja lisäyslaskennat ovat erittäin nopeita näytönohjaimilla, jos ne tehdään juurikin siinä järjestyksessä. Ne ovat kaikista optimaalisimpia laskentatapoja optimisointia ajatellen. Mikäli varjostimessa olevan logiikan pystyy ilmaisemaan tällaista laskentatapaa käyttäen, se on suositeltavaa. (Halladay 2019) Kuvassa 5 on nähtävillä esimerkki kyseisestä laskentatavasta.

```
Properties
    {
        _X ("X", Float) = 1.0
        _Y ("Y", Float) = 2.0
        _Z ("Z", Float) = 3.0
    }

            // Vertex program
            v2f vert (appdata v)
            {
                        float optimized = (_X * _Y) + _Z;
                        float non-optimized = _X * (_Y + _Z);
            }
```

KUVA 5. Kerto- ja lisäyslaskennan esimerkki.

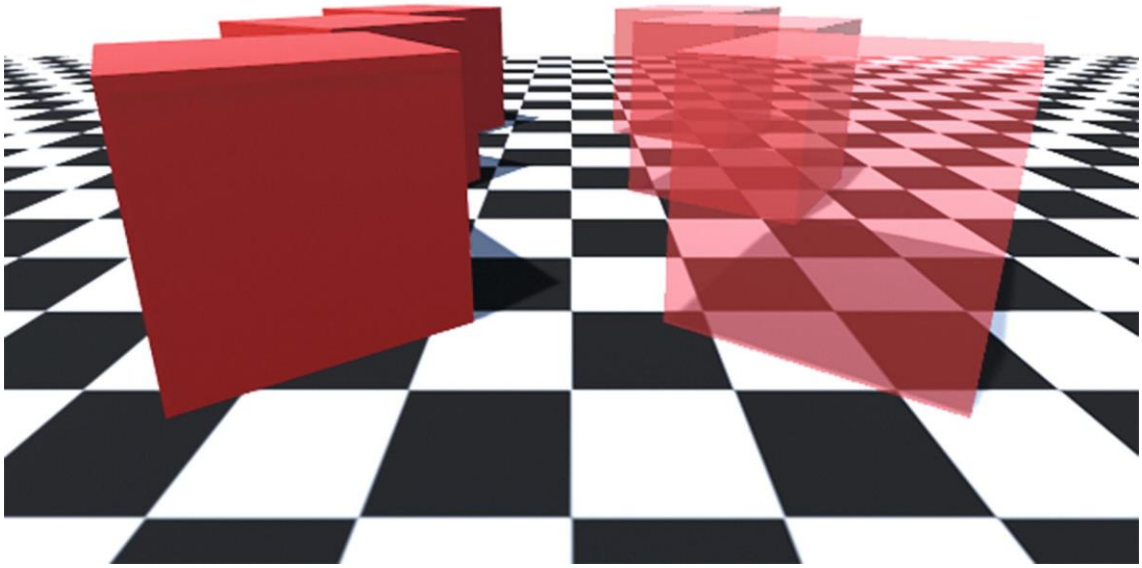
3.5 GLSL funktioiden käyttö

Mikäli on mahdollista käyttää sisäänrakennettuja GLSL funktioita, niitä tulisi käyttää. Sisäänrakennetut funktiot pystyvät käyttämään nopeita reittejä laitteistossa, joita ei ole itse mahdollista käyttää varjostin koodissa. Ne ovat myös enemmän testattuja sekä dokumentoituja kuin mikään koodi, jonka pystyy itse kirjoittamaan. (Halladay 2019.)

3.6 Ylipiirtäminen

Usein peleissä yleinen näytönohjaimen suorituskyvyn pullonkaula on ylipiirtäminen. Se tapahtuu silloin, kun asetelma renderöi jo valmiiksi piirretyn pikselin uudestaan. Tämä on mahdollista tapahtua sekä läpikuultavia että läpinäkymättömiä

materiaaleja käyttäessä. (Halladay 2019) Kuvassa 6 on ylipiirtäminen esitettynä käyttäen kuutioita läpinäkyvällä ja läpinäkymättömällä materiaalilla.



KUVA 6. Läpinäkymättömät ja läpinäkyvät materiaalit (Halladay 2019.)

Läpinäkyvää geometriaa tehdessä on hankalaa estää ylipiirtoa. Siitä huolimatta, näistä syntyvän ylipiirron määrästä on hyvä olla perillä, sillä liian suuri määrä ylipiirtoa saattaa hyvin nopeasti vaikuttaa merkittävästi näytönohjaimen suorituskykyyn. (Halladay 2019.)

Kuvassa 6 vasemmalla puolella on järjesteltynä laatikoita käyttäen läpinäkymätöntä materiaalia. Nämä laatikot saattavat silti aiheuttaa saman verran ylipiirtämistä, mutta se ei ole niin helposti havaittavissa. Tähän vaikuttaa suuresti asetelman piirtojärjestys; mikäli laatikot piirretään taaimmaisesta etummaiseen, näytönohjain ylipiirtää kuvassa lähimpänä olevat laatikot kauimmaisien päälle. Yleensä tätä ongelmaa välttääkseen asetelmat piirretään ruudulle lähimmäistä malleista kauimmaisiin. (Halladay 2019.)

4 PROFILOINTI

4.1 Mittaaminen

Mittaus on optimointiprosessin tärkein vaihe. Sen avulla pystytään varmistamaan suorituskyvyn ongelmien olemassaolo sekä niihin liittyvien korjauksien tulokset. Kaikki mitä pelistä piirretään, vie suorituskykyä ja mittaamisella pyritään havaitsemaan suorituskyvyn mahdollisia pullonkauloja. (Halladay 2019.)

Kuvataajuutta käytetään yleisesti suorituskyvyn mittarina ohjelmoinnin ulkopuolella. Se kertoo pelin tai ohjelmiston piirrettyjen kehysten määrän sekunnissa. Mikäli peli ei pysty pitämään kohdekuvataajuutta, sillä voidaan sanoa olevan suorituskykyongelmia. Kuvataajuutta ei kuitenkaan käytetä mittarina ohjelmoinnissa, sillä se on epälineaarinen asteikko. Tämän ongelman ratkaisemiseksi ohjelmoinnissa käytetään kehysaikaa, eli aikaa, joka kuluu yhden kehyksen piirtämiseksi ja se ilmaistaan yleensä millisekunteina. (Halladay 2019.)

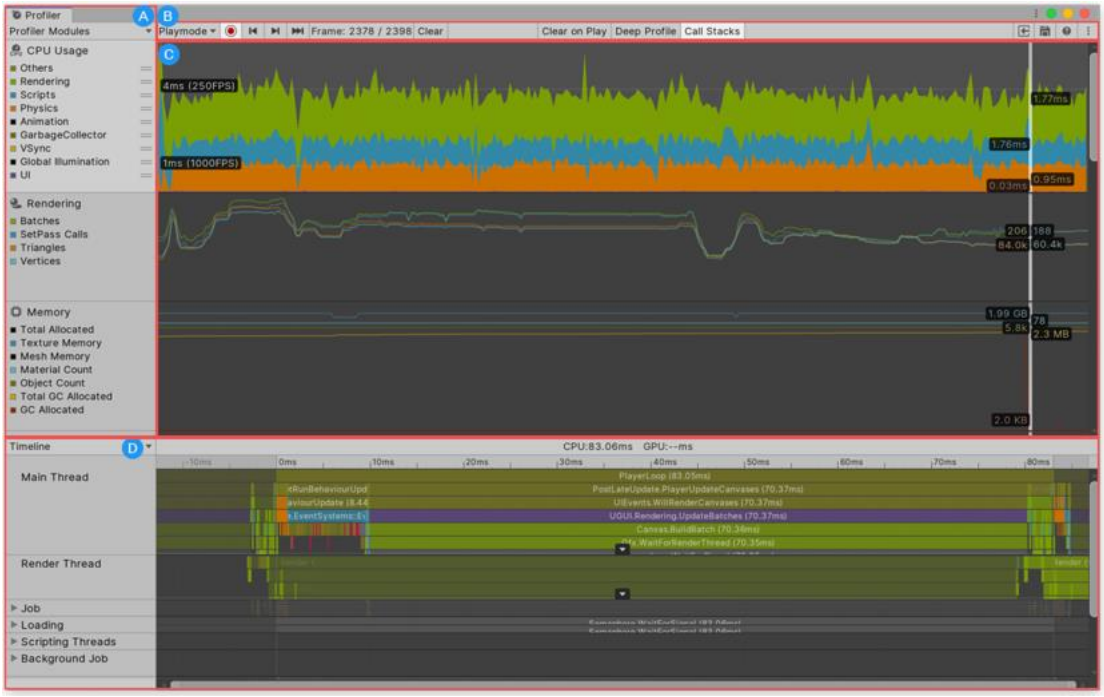
4.2 Profiloinnin työkalut

Profilointityökalu on ohjelmisto, jolla on mahdollista mitata applikaation suorituskykyä. Usein tietotekniikassa profilointityökalut mittaavat ainoastaan applikaation funktioiden kutsumääriä sekä niihin kulutettuja aikoja. Tämän lisäksi on olemassa useita muita profilointityökaluja, joilla on eri käyttötarkoitukset. Näitä ovat muun muassa muistiprofiloija sekä grafiikkaprofiloija. (Introduction to Software Engineering/Testing/Profiling 2011.)

4.2.1 Unity Profiler

Unity Profiler on työkalu, jolla on mahdollista saada suorituskykydataa sille rakennetusta applikaatiosta tai pelistä. Sitä voi käyttää Unityn sisällä ja sen voi myös yhdistää kohdelaitteeseen verkon yli tai kaapelilla yhdistettynä tietokoneeseen. (Unity 2019.4 LTS.)

Unity Profiler pystyy keräämään applikaation käyttämiä resursseja ajon aikana prosessorista, muistista, renderöintimoottorista, fysiikkamoottorista, ääniadapterista, näytönohjaimesta, käyttöliittymästä sekä yleisestä valaistusmallista. Unity Profiler on hyödyllinen työkalu havainnoimaan suorituskyvyn pullonkauloja. (Unity 2019.4 LTS) Kuvassa 7 on ruudunkaappaus Unity Profilerin käyttöliittymästä.

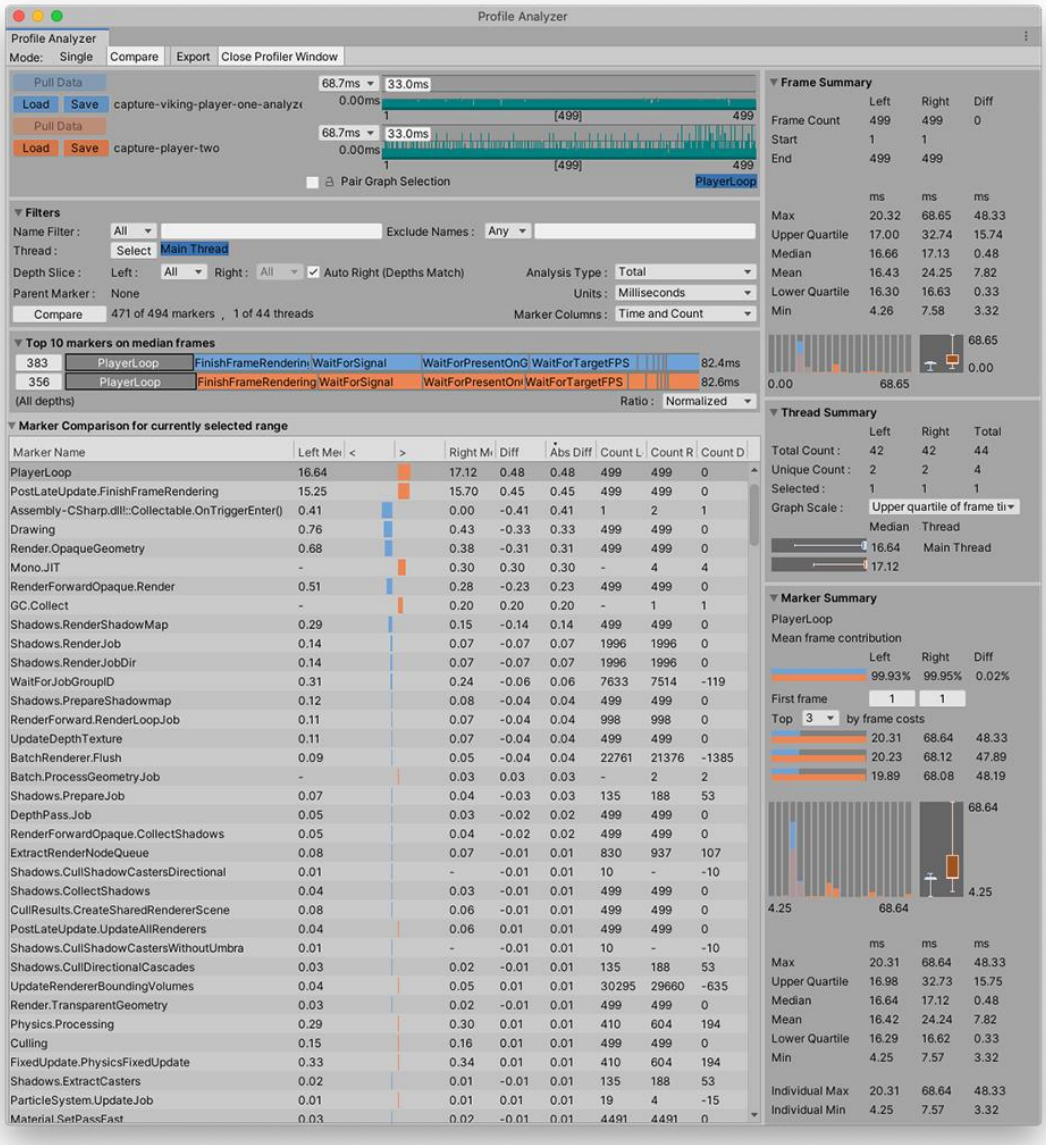


KUVA 7. Unity Profiler -näkymä. (Unity Documentation. 2019.4 LTS)

Ruudunkaappauksessa on merkittynä neljä eri osiota: A, B, C ja D osiot. Osiosta A löytyvät profilointimoduulit, tässä tapauksessa nähtävillä on dataa prosessorista, renderöintimoottorista sekä muistinkäytöstä. B-osiossa ovat profilointityökalun asetukset sekä kontrollit. Tästä osiosta voi säätää mitä laitetta ollaan profiloimassa, millaista profilointia ollaan tekemässä, datan tallentamisen aloitus sekä framejen kelaus. Osio C esittää saadun datan ajanjakson mukaan kaaviomallisesti. Osiossa D on esitettynä tarkemmin tietyn ajanjakson kuormitukset. (Unity 2019.4 LTS.)

4.2.2 Unity Profile Analyzer

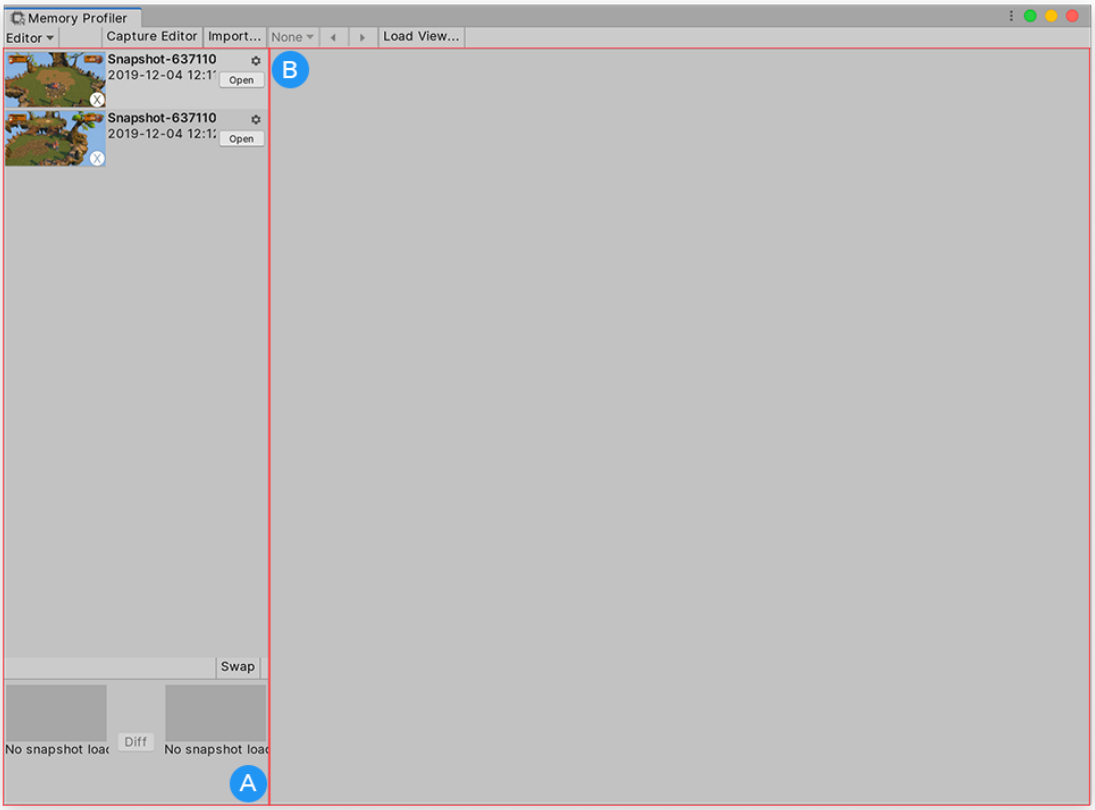
Profile Analyzer on työkalu, jolla voidaan verrata Unity Profilerin tallennettuja datasettejä. Datasetit ja niiden eroavaisuudet ovat visualisoituneena eri väreillä vierekkäin. (Unity Manual: About Profiler Analyzer 1.0.1) Kuvassa 8 on nähtävillä Profiler Analyzer vertaustoiminnon näkymä, jossa verrataan kahta eri datasettiä keskenään.



KUVA 8. Unity Profiler Analyzer käyttöliittymä (Unity Manual: About Profiler Analyzer 1.0.1.)

4.2.3 Unity Memory Profiler

Unity Memory Profiler työkalulla on mahdollista tarkistella applikaation muistin käyttöä. Sillä pystyy tallentamaan, tarkastelemaan sekä vertaamaan tallennettuja muistin kaappauksia. Memory Profiler eroaa Unity Profilerin muistimoduulista, sillä se on suunniteltu antamaan tarkempaa tietoa applikaation muistinkäytöstä. Kuvassa 9 on ruudunkaappaus Unity Memory Profiler -näkymästä. (Unity Manual: Memory Profiler 0.2.4.)

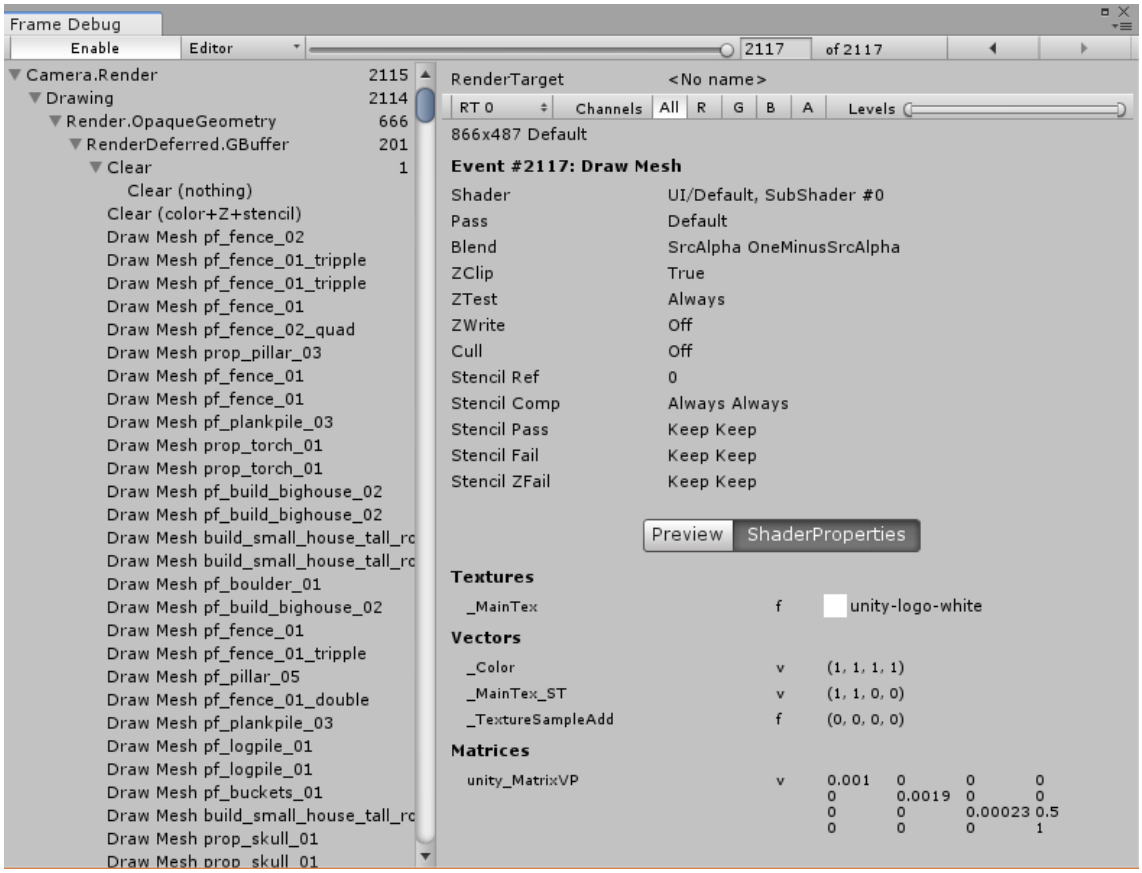


KUVA 9. Unity Memory Profiler. (Unity Manual: Memory Profiler 0.2.4.)

Käyttöliittymä on rajattuna kahteen osioon. Ylimmäisenä löytyvät Memory Profilerin toiminnot, kuten esimerkiksi muistinkaappaus. Osiosta A löytyvät tietokoneelle tallennetut muistikaappaukset. Osio B näyttää valitun muistinkaappauksen tiedot tarkemmin. (Unity Manual: Memory Profiler 0.2.4.)

4.2.4 Unity Frame Debugger

Unity Frame Debuggerilla voidaan pysäyttää pelitila ja tarkastella sen kehyksen piirtokutsuja. Piirtokutsuja on mahdollista selata yksitellen ja siitä näkee, kuinka asetelma rakennetaan piirtoa varten. Frame Debuggerilla pystytään havaitsemaan asetelmassa olevia suorituskykyongelmia. (Unity Tutorial – Working with the Frame Debugger) Kuvassa 10 on esitettynä ruudunkaappaus Frame Debuggerin käyttöliittymästä.



KUVA 10. Unity Frame Debugger. (Unity Tutorial – Working with the Frame Debugger.)

Frame Debuggerin näkymässä vasemmalla puolella on listattuna piirtokutsut järjestettynä ensimmäisestä viimeisimpään. Tässä osiossa näkyvät myös muun muassa jälkiprosessointiefektit. Oikealla puolella on näkyvissä valitun piirtokutsun tarkemmat tiedot, joihin kuuluvat piirrettävän mallin polygoniverkon tiedot sekä sen piirtämiseksi käytetyn varjostimen tiedot. (Unity Tutorial – Working with the Frame Debugger.)

5 VESIVARJOSTIMEN LÄHTÖTILANNE

5.1 Vesivarjostimen kuvaus

Transit King Tycoonin vesivarjostin koostuu kahdesta eri varjostimesta. Yksinkertaisempi varjostin käyttää ainoastaan tummempaa meren väriä koko alueella. Monimutkaisempi toteutustapa käyttää efektin luomiseksi syvyyskarttaa sekä kahta eri tekstuuria aaltoihin. Laitteen suorituskyvyn mukaan valitaan toteutustavoista toinen. Kuvassa 12 on nähtävillä yksinkertaisempi vesivarjostin.

KUVA 12. Transit King Tycoon lähtötilanteen yksinkertainen vesivarjostin.

Yksinkertainen varjostin on erittäin pelkistetty, joka mahdollistaa optimoinnin heikoimmille laitteille. Suorituskyvyn kannalta se on hyvä, sillä pelkkä värivarjostin on tehokas. Tämä kuitenkin maksaa suuren hinnan visuaalisesti verrattuna yksityiskohtaisempaan toteutustapaan. Kuvassa 13 on nähtävillä monimutkaisempi vesivarjostin.

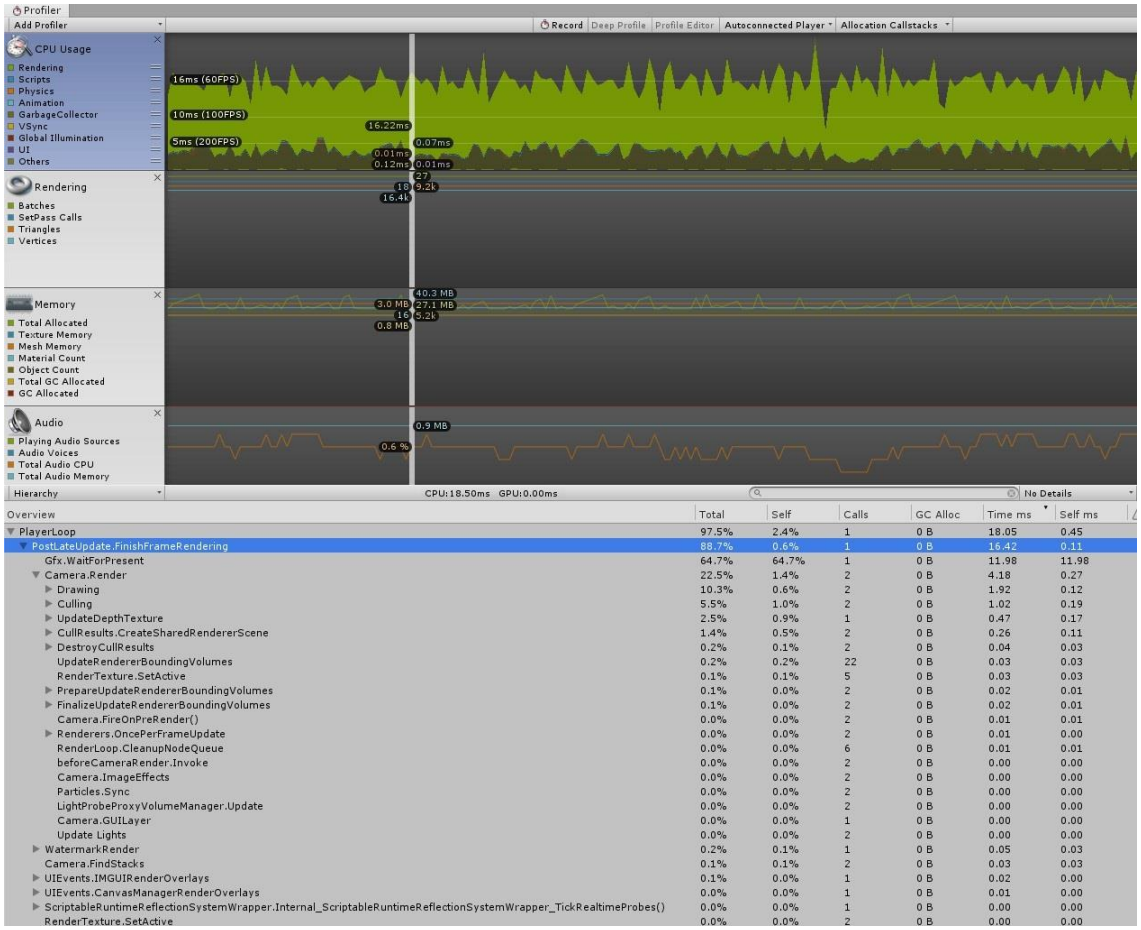
KUVA 13. Transit King Tycoon lähtötilanteen yksityiskohtainen vesivarjostin.

Yksityiskohtaisempi toteutustapa näyttää visuaalisesti hyvältä, mutta se on suorituskyvyn kannalta raskaampi. Tämä tarkoittaa, että heikoimmilla laitteilla hienompaa vettä ei kovin usein nähdä. Varjostinefekti toteutetaan käyttäen dynaamista haaroitusta, syvyyskarttaa sekä kahta eri tekstuuria aaltojen luomiseksi. Suorituskyvyn kannalta nämä ovat mahdollisia ongelmia. Syvyyskartan luomiseksi laitteen prosessorin ja näytönohjaimen täytyy erillisesti prosessoida siihen kuuluvat asetelman objektit, joka puolestaan lisää vaadittavia piirtokutsuja. Dynaaminen haaroitus ei ole suositeltavaa, sillä se heikentää näytönohjaimen natiiveja optimointijärjestelmiä. Tekstuurien käyttö on todennäköisesti näistä mahdollisista ongelmakohdista pienin.

5.2 Suorituskyky

Tässä vaiheessa on suositeltavaa tehdä varsinainen asetelma testaamista ja profilointia varten. Asetelmaan lisätään ainoastaan ne objektit ja varjostimet, joita vertaillaan. Profilointi tulee suorittaa kohdelaitteella datan oikeellisuuden varmistamiseksi. Tässä tapauksessa mittaaminen suoritettiin OnePlus 5 laitteella.

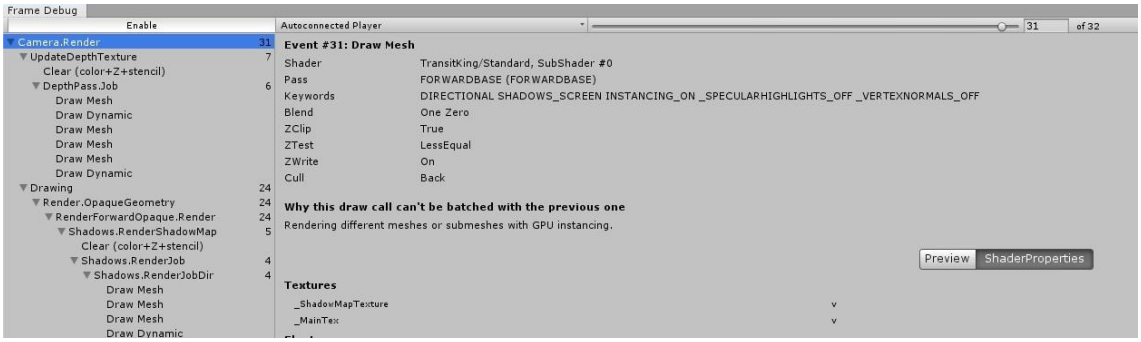
Lähtötilanteen varjostimen suorituskyky on syytä mitata ennen uuden varjostimen tekoa. Tässä työssä lähdettiin liikkeelle Unity Profilerista ja siitä saadut tulokset ovat nähtävissä kuvassa 14.



KUVA 14. Unity Profiler tulokset varjostimen lähtötilanteesta.

Unity Profilerista nähdään, että lähtötilanteessa prosessorilla renderöintiin kului noin 4,18 millisekuntia. Tässä asetelmassa ei kuitenkaan ole kovinkaan paljon objekteja, joten lukema vaikuttaa suurelta, mikäli tähdätään kuvataajuuteen 60.

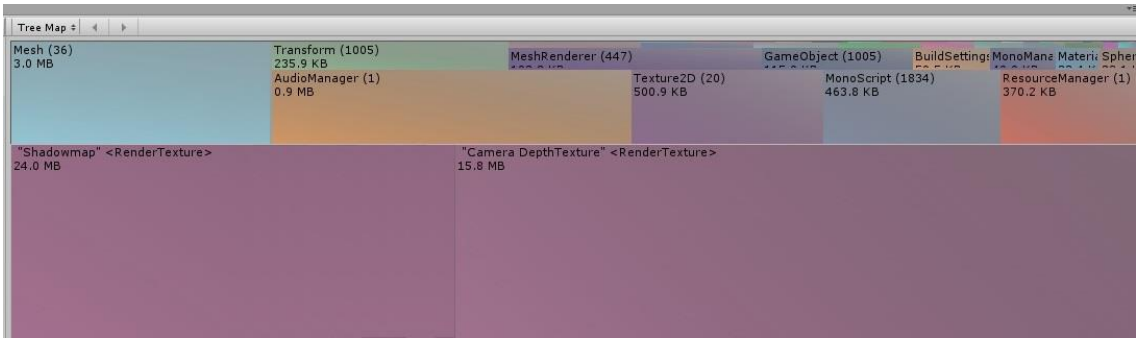
Seuraavaksi katsotaan tilannetta Unity Frame Debuggerilla. Profiloinnista saadut tulokset ovat nähtävissä kuvassa 15.



KUVA 15. Unity Frame Debugger tulokset varjostimen lähtötilanteesta.

Frame Debuggerista saadaan tarkempaa tietoa piirtokutsuista. Kokonaisuudessaan tässä asetelmassa piirtokutsuja tehtiin 32. Tarkemmin havaitaan, että varjoja piirretään kahdessa erässä. Kummassakin erässä piirtokutsuja kertyy viisi kappaletta eli yhteensä niitä on 10 kappaletta. Tämän lisäksi syvyystekstuurin piirtämiseen kuluu piirtokutsuja yhteensä seitsemän kappaletta.

Unity Profilerin ja Frame Debuggerin lisäksi on hyvä tarkastaa käytetty muisti. Memory Profilerin tulokset ovat nähtävissä kuvassa 16.



KUVA 16. Unity Frame Memory Profiler tulokset varjostimen lähtötilanteesta.

Muistinkaappauksesta havaitaan, että Unity tallentaa syvyystekstuurin muistiin ja se käyttää noin 15 megabittiä muistia. Syvyystekstuurin muistinkäyttö ei välttämättä ole ongelma, mutta optimaalisempaa olisi, mikäli sitä ei tarvitsisi käyttää ainoastaan varjostimen vuoksi.

5.3 Uuden varjostimen ominaisuudet

Toimeksiantajalta saaduissa ohjeistuksissa haluttiin keskittyä vesivarjostimen optimointiin sekä ulkonäön säilyttämiseen sellaisena kuin se on. Vesivarjostin todettiin käyttävän liikaa resursseja mobiililaitteilla, mikä aiheutti osaltaan kohdelaitteiden kuumenemisen. Varjostin itsessään ei ollut ainoa syy lämpenemiselle, mutta sen optimoimisella koettiin olevan positiivinen vaikutus ongelman ratkaisemiseksi. Pelissä olevan vesiefektin todettiin olevan pelin taidetyylin mukainen, joten visuaalisesti vesivarjostimella ei koettu olevan tarpeita muutoksille.

Näillä ohjeistuksilla lähdettiin toteuttamaan tiedonhakua siitä, mikä saattaisi olla optimaalisempi toteutustapa vesivarjostimelle mobiililaitteilla.

6 TIEDONHAKU

6.1 Sarjakuvatyylinen vesivarjostin

Roystan.net sivustolta löytyi tiedonhaussa sarjakuvatyylinen vesivarjostin tutoriaali. Visuaalisesti varjostin on hyvännäköinen ja se saattaisi muokkauksien kanssa toimia Transit King Tycoonissa. Kuvassa 17 on kuvakaappaus kyseisestä vesivarjostimesta.

KUVA 17. Sarjakuvatyylinen vesivarjostin (Roystan.net. Toon Water Shader.)

Tässä vesivarjostimen toteutuksessa käytetään syvyystekstuuria ja -kameraa efektin luomiseksi. Lopputuloksessa käytetään useampaa tekstuuria, kuten Perlin Noise- sekä vääristystekstuuria aaltojen luomiseksi. Näiden lisäksi aaltojen yhtenäisen koon varmistamiseksi käytetään näytönohjaimen normaalipuskuria ja aaltojen reunojen tasaamiseksi käytetään varjostimessa olevaa reunanpehmennystekniikkaa. (Roystan.net. Toon Water Shader.)

Tutoriaalissa käytettävät tekniikat vaikuttavat melko raskailta mobiilialustoille. Transit King Tycoonin vesivarjostimessa käytetään ainoastaan syvyyskarttaa ja tekstuureita, joten optimoinnin kannalta tämän kaltainen ratkaisutapa ei välttämättä ole sopiva.

6.2 Yksinkertainen vesivarjostin

Tiedonhaussa löydettiin tutoriaali yksinkertaisesta vesivarjostimen toteutustavasta Lindenreid.wordpress.com sivustolta. Tässä ratkaisutavassa toteutetaan nimensä mukaan yksinkertainen vesiefekti käyttäen erillistä asetelmassa olevaa vesitasoa, syvyyskarttaa sekä vesitason pisteiden sijaintien animaatiota aaltojen tekemiseksi (Lindenreid.wordpress.com. Simple Water Shader in Unity). Kuvassa 18 on nähtävillä vesivarjostimen lopputulos.

KUVA 18. Yksinkertainen vesiefekti (Lindenreid.wordpress.com. Simple Water Shader in Unity.)

Vesivarjostin näyttää yksinkertaiselta ja se saattaisi yksinkertaisuutensa vuoksi soveltua Transit King Tycooniin. Kuitenkin, tässä ratkaisutavassa on samoja ongelmia kuin aiemmassa toteutuksessa; syvyyskarttaa käytetään ja aaltojen animoimiseksi käytetään useampaa tekstuuria. Tämän lisäksi tämä vesiefekti ei välttämättä muistuta tarpeeksi merta, joten tällaisenaan efekti ei välttämättä toimisi Transit King Tycoonissa optimoinnin sekä visuaalisen puolen takia.

6.3 Pistesijaintia käyttävä värimanipulaatio

Unityn foorumeilta löytyi mielenkiintoinen foorumilanka, jossa etsitään mobiilialustalle sopivaa tekniikkaa syvyyden saavuttamiseksi vesivarjostimelle. Keskustelulangassa kehotetaan käyttämään mallin pisteiden globaalia sijaintia veden korkeuden määrittämiseen. (Unity Forums) Tällöin syvyystekstuuria ei tarvitse renderöidä ja tämä tekniikka soveltuu myös varjostimen pisteohjelmassa suoritettavaksi, mikäli tekstuureita ei käytetä esimerkiksi aaltojen luomiseen. Kuvassa 19 on nähtävillä käyttäjän beef-Supremen vesivarjostimen lopputulos.

KUVA 19. Pistesijaintia käyttävä vesiefekti (Unity Forums. Unity Community Discussion. Graphics. Shaders. beef-Supreme.)

Tällaista ratkaisumallia käyttäessä tulee myös ottaa huomioon pelikameran kuvakulma, sekä meren ulkopuolisen alueen värjääminen, jonka pitää olla saman värinen kuin meren syvin kohta. Mikäli näitä seikkoja ei huomioida, tekniikan tekotapa saattaa näkyä pelaajalle, jolloin immersio saattaa rikkoontua vieden pelaajan pois pelimaailmasta.

7 VESIVARJOSTIMEN TOTEUTUS

7.1 Vesivarjostimen suunnittelu

Tiedonhaussa löydetyistä tekniikoista parhaimpana ratkaisuna nähtiin pistesijaintiin perustuva värimanipulaatio. Tässä toteutustavassa ei tarvita erillisiä syvyyskarttoja eikä myöskään tekstuureja, joka tekee siitä lupaavimman toteutustavan optimoinnin kannalta.

7.2 Vesivarjostimen toteutus

Vesivarjostimen toteutus aloitettiin monistamalla yksinkertaisin projektissa oleva pistevärityksvarjostin, jotta saatiin tarpeelliset varjostimen tuet kohdealustoille vähäisellä resurssimäärällä. Tämän jälkeen aloitettiin varjostimen teko tärkeimmästä ominaisuudesta lähtien.

Tärkeimmäksi varjostinominaisuudeksi nähtiin vesirajan tekeminen. Sen yläpuolella mallit pitää värjätä kuten aiemminkin, tässä tapauksessa käyttäen joko pisteväritystä tai tekstuureja. Vesirajan alapuolella tätä väritystä ei käytetä vaan mallit värjätään meren väreillä. Vesiraja saatiin aikaiseksi käyttäen ehtolausetta, joka määritettiin pistesijainnista. Ehtolauseen ensimmäisessä osiossa malli värjättiin käyttäen tekstuuria tai pisteväritystä ja toisessa osiossa käyttäen meren värejä. Vesiraja animoitiin käyttäen aikamuuttujaa sekä varjostimien sin -funktiota.

Varjostimessa syvyyttä ei lasketa käyttäen kameran muodostamaa syvyystekstuuria vaan se lasketaan maailman koordinaatistosta ja mallin pistesijainnin suhteesta. Meren syvyyden visualisoimiseksi käytettiin Lerp -funktiota, johon parametreina syötettiin kaksi eri väriarvoa ja mallin pisteen laskettu syvyys. Laskettu syvyys animoitiin käyttäen sin -funktiota. Lopputuloksena saatiin värigradientti syvyyden mukaan.

Aaltojen vaahto toteutettiin käyttäen ehtolausetta ja Lerp -funktiota. Ehtolause muodostettiin luomaan aalloille variaatioita asetelmasijainnin mukaan. Lerp -funktiota käytettiin muodostamaan gradientti vaahdolle.

Vesirajan yläpuolelle lisättiin vielä märkää hiekkaa efektinä, joka toteutettiin käyttäen ehtolausetta. Ehtolauseen sisällä laskettua väriarvoa kerrottiin muuttujalla, jotta saatiin aikaan tummempi väriarvo. Se animoitiin käyttäen sin -funktiota aikamuuttujalla ja se skaalattiin samoin kuin vesirajassa, jotta ne pysyivät synkronoituneina.

Lopullinen vesivarjostin on nähtävissä kuvassa 20.

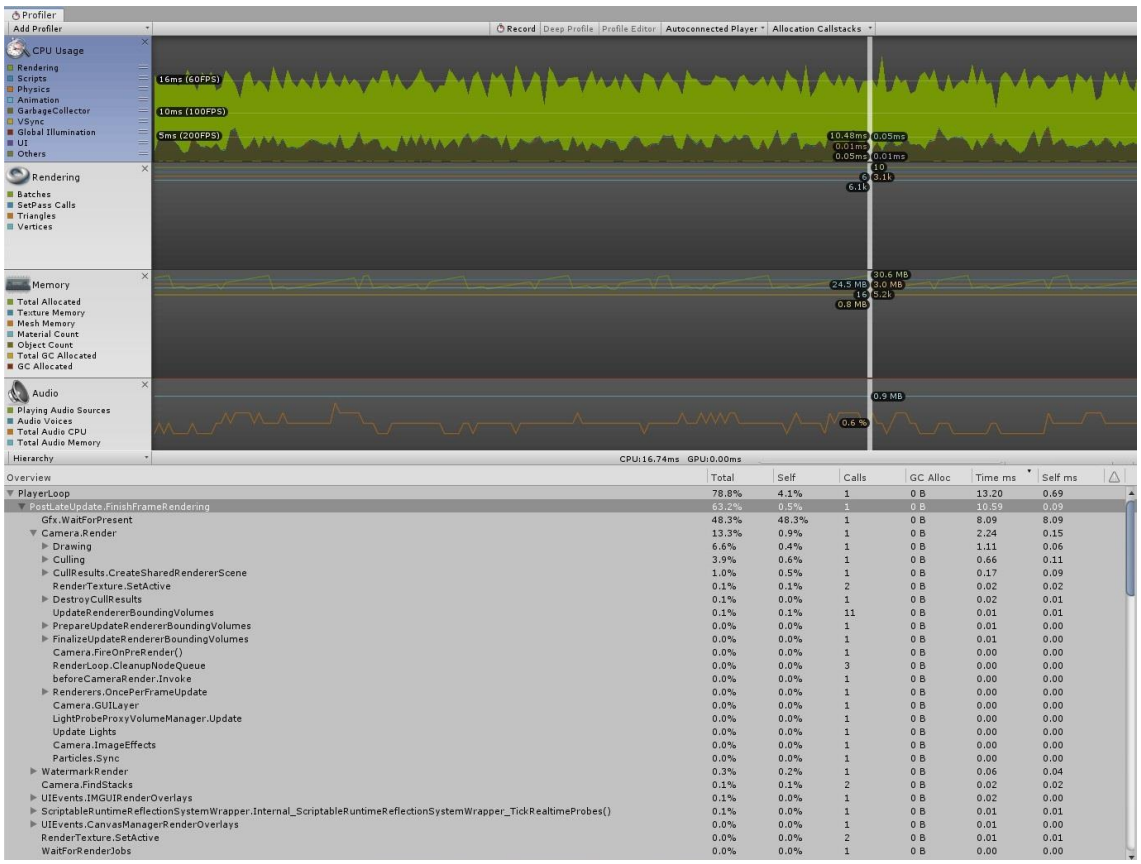
KUVA 20. Vesivarjostimen lopputulos.

Ongelmia syntyi geometrian kanssa, sillä meressä on nähtävissä selkeä raja missä matala meri vaihtuu syväksi. Tämän ongelman ratkaiseminen varjostimella olisi ollut turhan työlästä ja visuaalisesti meri olisi näyttänyt hyvin äkkisyvältä, joten artistia pyydettiin muokkaamaan geometriaa soveltuvaksi. Kuvassa 21 on nähtävillä sama varjostin viimeistellyllä geometrialla.

KUVA 21. Vesivarjostin viimeistellyllä geometrialla.

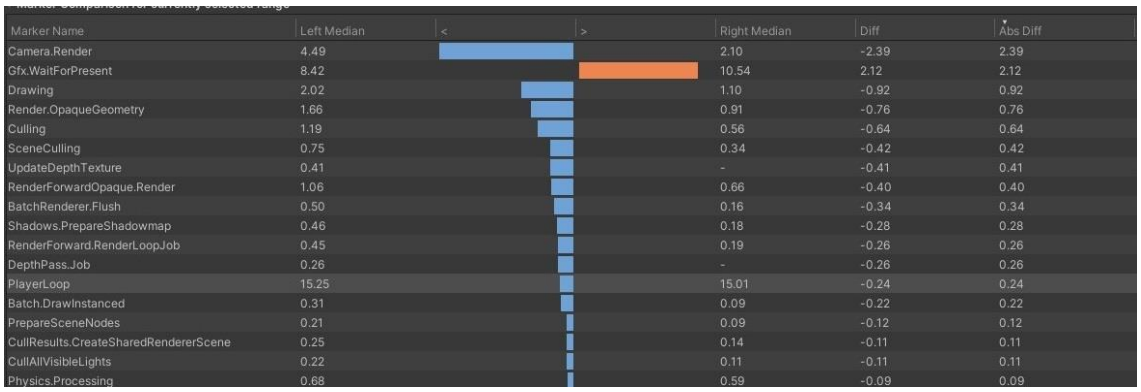
7.3 Vesivarjostimien vertailu

Unity profilerilla uuden vesivarjostimen suorituskykyä tarkasteltaessa nähdään, että asetelman renderöinti Camera.Render:ssä vei 2,24 millisekuntia, joka on 1,94 millisekuntia vähemmän aiempaan nähden. Kuvassa 22 on nähtävissä profilerin ruutukaappaus. Pelkällä profilerilla ei ole suositeltavaa verrata suorituskykyä, vaan parempi on tarkastaa tilanne Profiler Analyzerilla, joka on tehty juuri tätä varten.



KUVA 22. Unity Profiler

Profiler Analyzeriin syötettiin kahden eri asetelman suorituskykydata. Kuvassa 23 on sinisellä nähtävissä vanhemman vesivarjostimen suorituskyky ja oranssilla uudemman version suorituskyky. Tärkeimpinä huomioina muutokset Camera.Render ja Gfx.WaitForPresent. Camera.Render pieneni 4,49 millisekunnista 2,10 millisekuntiin, tarkoittaen että asetelma renderöidään nykyään nopeammin. Gfx.WaitForPresent funktio nousi puolestaan 8,42 millisekunnista 10,54 millisekuntiin. Varmaa tietoa Gfx.WaitForPresent:stä ei ole, mutta Unityn foorumeiden mukaan se on odotusaika, jolloin laite on suorittanut kuvan piirtämisen. (Unity Forums. Unity Community Discussion. Editor & General Support.)



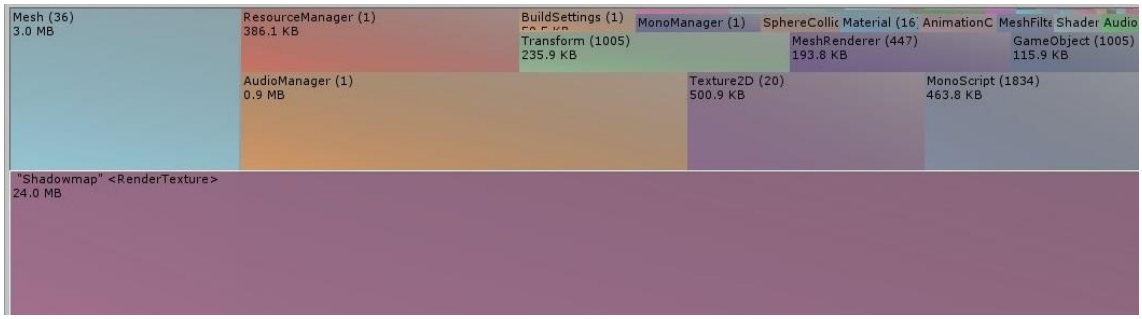
Marker Name	Left Median	Right Median	Diff	Abs Diff
Camera.Render	4.49	2.10	-2.39	2.39
Gfx.WaitForPresent	8.42	10.54	2.12	2.12
Drawing	2.02	1.10	-0.92	0.92
Render.OpaqueGeometry	1.66	0.91	-0.76	0.76
Culling	1.19	0.56	-0.64	0.64
SceneCulling	0.75	0.34	-0.42	0.42
UpdateDepthTexture	0.41	-	-0.41	0.41
RenderForwardOpaque.Render	1.06	0.66	-0.40	0.40
BatchRenderer.Flush	0.50	0.16	-0.34	0.34
Shadows.PrepareShadowmap	0.46	0.18	-0.28	0.28
RenderForward.RenderLoopJob	0.45	0.19	-0.26	0.26
DepthPass.Job	0.26	-	-0.26	0.26
PlayerLoop	15.25	15.01	-0.24	0.24
Batch.DrawInstanced	0.31	0.09	-0.22	0.22
PrepareSceneNodes	0.21	0.09	-0.12	0.12
CullResults.CreateSharedRendererScene	0.25	0.14	-0.11	0.11
CullAllVisibleLights	0.22	0.11	-0.11	0.11
Physics.Processing	0.68	0.59	-0.09	0.09

KUVA 23. Unity Profiler Analyzer

Seuraavaksi vilkaistaan tilanne Frame Debuggerista. Aiemmin asetelmassa piirtokutsuja tehtiin 32 kappaletta ja nyt uudella varjostimella niitä tehtiin ainoastaan 12. Prosentuaalisesti piirtokutsujen määrä tippui 62,5%. Kuvassa 24 on nähtävillä piirtokutsut Frame Debuggerissa.

KUVA 24. Unity Frame Debugger

Viimeiseksi tarkistetaan Memory Profiler. Aiemmin syvyystekstuuri tallennettiin laitteen väliaikaismuistiin ja se tarvitsi tilaa 15,8MB. Uudemman varjostimen muisti on nähtävissä kuvassa 25 ja siitä nähdään, että syvyystekstuuria ei enää tallenneta muistiin, jolloin se säästää sitä 15,8MB.



KUVA 25. Unity Memory Profiler

8 POHDINTA

Opinnäytetyön tavoitteena oli tehdä suorituskyvyn kannalta optimaalisempi ja visuaalisesti saman tasoinen vesivarjostin Transit King Tycoon mobiilipeliin. Profilointi Unityn omilla työkaluilla onnistui ongelmitta, mutta tarkkaa dataa näytönohjaimesta sillä ei valitettavasti ole mahdollista saada. Näillä työkaluilla kuitenkin pystyttiin profiloimaan kohdelaitetta ja todentamaan optimointitavoitteiden onnistuminen, joka testausasetelmassa vaikutti hyvältä. Tämä antoi ajatusta siitä, että vanhempi toteutustapa saattoi sotkea piirtojärjestystä ja täten kuormittaa kohdelaitteita tarpeettomasti.

Visuaalisesti varjostin pysyi aiemman kaltaisena, mutta siihen lisättiin pieniä lisäominaisuuksia kuten esimerkiksi märän hiekan efekti, joka tekivät siitä entistä paremman näköisen. Varjostimen visuaalisena ongelmana ilmeni hyvin yksitoikkoinen ulompi meri, joten jatkokehityksen kannalta siihen voisi lisätä aallokkoa.

Varjostimesta ei kuitenkaan tullut optimaalisin. Suurin osa laskennasta tapahtui pikseliohjelmassa, mutta teoriassa sen olisi voinut siirtää pisteohjelmaan näytönohjaimen lisäoptimoinnin vuoksi. Tämän lisäksi meren varjostinkoodissa käytettiin dynaamista haaroitusta, jota ei ole suositeltavaa käyttää varjostinkoodissa. Näitä ongelmakohtia ei kuitenkaan lähdetty korjaamaan, sillä verrattaessa aiempaan vesivarjostimeen uudessa varjostimessa nähtiin huomattavaa parannusta.

LÄHTEET

Davis, Martin J. 2011. Computer Graphics.

Halladay, Kyle. 2019. Practical Shader Development Vertex and Fragment Shaders for Game Developers.

Thorn, Alan & Doran, John P. & Zucconi, Alan & Palacios, Jorge. 2018. Complete Unity 2018 Game Development.

Unity Documentation. 2019.4 LTS. Analysis: Profiler overview. Luettu 21.7.2020 https://docs.unity3d.com/Manual/Profiler.html

Unity Manual. About Profiler Analyzer 1.0.1. Luettu 25.7.2020. https://docs.unity3d.com/Packages/com.unity.performance.profile-analyzer@1.0/manual/index.html

Unity Manual. Memory Profiler 0.2.4-preview. Luettu 21.7.2020. https://docs.unity3d.com/Packages/com.unity.memoryprofiler@0.2/manual/index.html

Unity Tutorial. Working with the Frame Debugger. Luettu 25.7.2020. https://learn.unity.com/tutorial/working-with-the-frame-debugger#5ce57ea8edbc2a0ebeae75d7

Wikibooks. Introduction to Software Engineering/Testing/Profiling. Luettu 25.7.2020. https://en.wikibooks.org/wiki/Introduction_to_Software_Engineering/Testing/Profiling

Roystan.net. Toon Water Shader. Tutorial. Luettu 29.7.2020. https://roystan.net/articles/toon-water.html

Lindenreid.wordpress.com. Simple Water Shader in Unity. Tutorial. Luettu 31.7.2020. https://lindenreid.wordpress.com/2017/12/15/simple-water-shader-in-unity/

Unity Forums. Unity Community Discussion. Graphics. Shaders. Mobile water with “Depth Alpha” or “Depth Opacity”? Luettu 29.7.2020. https://forum.unity.com/threads/mobile-water-with-depth-alpha-or-depth-opacity-or-darker-stuff-deeper.486107/

Unity Forums. Unity Community Discussion. Editor & General Support. Device.Present in profiler….what’ the deal. Luettu 27.10.2020. https://forum.unity.com/threads/device-present-in-profiler-what-the-deal.86312/