Tampereen yliopisto

Okko Saarinen

UNITYN SOVELTUMINEN DAYDREAM - ALUSTALLA TOIMIVAN VR-SOVELLUKSEN TOTEUTUKSEEN

Tekniikan ja luonnontieteiden tiedekunta
Kandidaatintyö
Kesäkuu 2019

TIIVISTELMÄ

Okko Saarinen: Unityn soveltuminen Daydream-alustalla toimivan VR-sovelluksen toteutukseen
Kandidaatintyö
Tampereen yliopisto
Automaation tietotekniikka
Kesäkuu 2019

Virtuaalitodellisuussovellusten kehityksen mahdollistavat teknologiat ovat kehittyneet viime vuosina nopeasti. Lisäksi kuluttajilla olevien laitteiden laskentateho on siinä pisteessä, että virtuaalitodellisuuden kokeminen onnistuu jo älypuhelimillakin. Virtuaalitodellisuussovellusten kehitys vaatii myös erityistietämystä sillä varomaton toteutus saattaa aiheuttaa käyttäjille pahoinvointia.

Työn tavoite on tutkia Unityn soveltuvuutta virtuaalitodellisuussovellusten kehitykseen ja tuottaa pohjamateriaalia muille virtuaalitodellisuussovellusten kehityksen aloittaville henkilöille. Työn kokeellisessa osassa toteutetaan sovellus, jossa voidaan tarkastella Tampereen teknillisen yliopiston tiloista löytyvää tislauskolonnia virtuaalisessa ympäristössä.

Työssä toteutettiin kokeellinen virtuaalitodellisuussovellus Unityllä, sekä kirjattiin hyväksi todettuja käytäntöjä. Lisäksi dokumentoitiin kokeellisessa työssäkin käytettyjä optimointimenetelmiä, joilla voidaan täyttää puhelinpohjaisten virtuaalitodellisuussovellusten suorituskykyvaatimukset.

Avainsanat: Virtuaalitodellisuus, Unity, Tislauskolonni, Daydream

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

VR	Virtual Reality, virtuaalitodellisuus
OPC	OLE for Process Control, tiedonsiirtostandardi
OPC UA	OPC Unified Architechture, laajennos OPC standardiin
HMD	Head-Mounted Display, kasvoilla pidettävä näyttölaite
FPS	Frames Per Second, kuinka monesti ruutu päivitetään sekunnissa
3DOF	3 Degrees of Freedom, kolme vapausastetta
6DOF	6 Degrees of Freedom, kuusi vapausastetta
RGBA	Red Green Blue Alpha, väriarvon kuvaustapa
PBR	Physically Based Rendering
4K	4096 x 2160 resoluutio
LOD	Level of Detail, 3D-mallin tarkkuuden taso
SDK	Software Development Kit
DOTS	Data-Oriented Technology Stack

1. JOHDANTO

Virtuaalitodellisuudella tarkoitetaan tietokoneella luotua keinotekoista ympäristöä, joka yrittää visuaalisesti ja äänimaailmaltaan matkia todellisuutta. Kolmiulotteisen ympäristön vaikutelman saavuttamiseksi käytetään yleensä stereoskopiaa eli tekniikkaa, jossa kummallekin silmälle syöttämällä eri kuvat saadaan aikaiseksi vaikutelma syvyydestä. Kuvat näytetään silmille asettamalla kasvoille lasit, jotka sisältävät kaksi näyttöä eriteltynä kummallekin silmälle linsseineen. Näyttöjä ympäröi kehys, joka suojaa näkökenttää ulkoisilta valoilta ja muilta häiriöiltä. Stereoäänillä saadaan äänimaailmaan realistinen tilantuntu.

Virtuaalitodellisuuden sovelluksia löytyy viihdeteollisuuden lisäksi paljon myös opetuskäytöstä. Virtuaalitodellisuutta voidaan käyttää erilaisien käytännön tilanteiden simuloinnissa, joiden järjestäminen todellisuudessa olisi vaarallista, kallista tai muulla tavalla epäkäytännöllistä, kuten esimerkiksi lentosimulaatioissa, armeijan harjoituksissa tai terveydenhuoltoalan harjoittelussa.

Virtuaalitodellisuuden mahdollistava teknologia on viime vuosina kehittynyt huomattavasti ja kiinnostus teknologiaa kohtaan sekä kuluttajien, yritysten että kehittäjien puolesta on samalla lisääntynyt. Oculus VR -yhtiön kehittämät Oculus Rift ja HTC:n ja Valve Corporationin yhteistyössä kehittämät HTC Vive -virtuaalilasit toivat virtuaalitodellisuuden suuren yleisön tietoisuuteen. Lisäksi Google ja Samsung ovat tuoneet omilla puhelimiin perustuvilla ratkaisulla sen lähemmäksi tavallista kuluttajaa.

Koska laitteistovaatimukset ovat virtuaalitodellisuussovelluksilla korkeat, voivat alan laitteet olla tavallisen kuluttajan saavuttamattomissa. Älypuhelinten suorituskyky on kuitenkin saavuttanut pisteen, joka mahdollistaa niiden käytön halvempana vaihtoehtona virtuaalitodellisuussovelluksiin. Käyttämällä puhelimen asentosensoreita ja näyttöön kiinnitettyä virtuaalisovelluksia varten suunniteltua koteloa saadaan kustannustehokas vaihtoehto monille käyttäjille, joille kalliit alan laitteet eivät ole saatavilla. Puhelimella toimivaan virtuaalitodellisuuskokoonpanoon riittää vain tarkkanäyttöinen puhelin sekä silmille asetettava kotelo linsseineen sekä mahdollisesti Bluetoothiin välityksellä toimiva ohjain. Puhelimen laitteistoon perustuvia ratkaisuja ovat esimerkiksi Googlen kehittämät Cardboard- ja Daydream-alustat, Samsungin kehittämä Gear VR sekä Microsoftin Windows Mixed Reality. Työssä pyritään toteuttamaan sovellus, jota on mahdollista ajaa Googlen Daydream-alustaa tukevilla älypuhelimilla.

Unity on aloittelijaystävällinen työkalu, joka on vakiinnuttanut paikkansa yhtenä suosituimmista pelimoottoreista. Unitylle löytyy valmiiksi toteutettuna paljon VR- kehityksessä vaadittuja komponentteja, kuten Daydream-ohjaimen käyttäjäsyötteiden käsittely sekä stereoskooppisen renderöinnin ja äänimaailman toteutukset [1]. Laajan käyttäjäkunnan vuoksi internetistä löytyy paljon opetusmateriaalia ja kehittäjäfoorumit ovat aktiiviset. Sovellus julkaistaan alustavasti ainakin Androidille, mutta Unity mahdollistaa julkaisujen tekemisen helposti myös muille alustoille.

Työn tavoitteena on tutkia Unityn soveltuvuutta virtuaalitodellisuussovellusten kehitykseen käyttäen Googlen Daydream-alustaa. Kokeellisessa osuudessa toteutetaan Tampereen teknillisen yliopiston tiloista löytyvä tislauskolonni virtuaalisena ympäristönä käyttäen hyödyksi Unitylla rakennettua sovellusta ja node.js -palvelintoteutusta, joka tuottaa kuvitteellista dataa tislauskolonnin prosessista. Sovelluksessa pyritään noudattamaan alalla hyväksi havaittuja käytäntöjä ja täyttää virtuaalitodellisuussovellusten kehityksen korkeat suorituskykyvaatimukset. Työssä käydään läpi ensin virtuaalitodellisuussovelluksen vaatimuksia ja yleisiä ongelmia sekä hyviä käytäntöjä. Tämän jälkeen käydään läpi eri optimointimenetelmiä. Lopuksi esitellään lyhyesti toteutunutta sovellusta ja mahdollisia parannuksia siihen. Työn on tarkoitus toimia pohjamateriaalina virtuaalitodellisuussovellusten kehittäjille.

2. VIRTUAALITODELLISUUDEN TOTEUTTAMINEN PUHELIMELLA

Aivot tulkitsevat syvyyttä silmien eri näkökenttien perusteella. Tätä käytetään hyväksi virtuaalitodellisuuden luomisessa. Virtuaalitodellisuuden kokemiseen tarvitaan tätä varten suunniteltu kasvoille asetettava näyttölaite (engl. Head-Mounted Display, HMD). Tyypillinen HMD sisältää joko kaksi tarkkaa näyttöä tai yhden tarkan näytön jaettuna kahteen osaan siten että kummallekin silmälle voidaan piirtää oma kuva. Linsseillä avustettuna saadaan koko näkökentän peittävistä kaksiulotteisista näytöistä heijastettua verkkokalvoille illuusio syvyydestä. Virtuaalisen maailman havainnoimiseen tarvitaan näytön lisäksi anturit, jotka havaitsevat pään kääntymisen ja kallistumisen. Kehittyneemmät laitteistot sisältävät anturit havaitsemaan kaikki kuusi liikkeen vapausastetta ja siten mahdollistavat myös liikkumisen virtuaaliympäristössä, joskin pienessä tilassa.

Tarkoitukseen rakennetut laitteet ovat usein hintavia ja vaativat tehokkaan tietokoneen tai pelikonsolin. Suosittuja kaupallisia malleja ovat esimerkiksi Oculus Rift, Sony Playstation VR ja HTC Vive. Useat tahot, kuten Google ja Samsung, ovat alkaneet kehittää vaihtoehtoista tapaa tuoda kuluttajalle virtuaalitodellisuuskokemuksia. Google julkaisi Google Cardboardiksi nimetyn pahvisen HMD:n vuonna 2014, joka mahdollisti Android- ja IOS-laitteiden käytön virtuaalilaseina. Samsung julkaisi oman Samsung Gear VR -virtuaalilasinsa vuonna 2015, joka mahdollistaa joidenkin Samsungin älypuhelinten käytön virtuaalilaseina. Google julkaisi toisen Daydreamiksi nimetyn monipuolisemman virtuaalitodellisuusalustan vuonna 2016.

Google Daydream -alusta määrää laitevalmistajille tietyt vaatimukset, jotka laitteen on täytettävä ollakseen Daydream-yhteensopiva [2]. Koska eri laitevalmistajat toteuttavat näyttölaitteita hieman eri tavalla, ei mittasuhteille ole asetettu tarkkoja sääntöjä. Sen sijaan lasit tulevat NFC-merkinnän kanssa, jonka laite osaa lukea ja asettaa kyseistä kokoonpanoa vastaavat sisäiset asetukset. Daydream tukee ainoastaan Android-laitteita alkaen Android 7.0 Nougat -versiosta. Daydream asettaa laitteistovaatimusten lisäksi vaatimuksen Bluetooth-ohjaintuelle, ja ohjaimen on tultava lasien mukana.

3. VR-SOVELLUSTEN YLEISET ONGELMAT JA NIIDEN RATKAISUT

Ennen kehityksen aloittamista on hyvä tutkia virtuaalisten ympäristöjen kehityksen hyviä käytäntöjä. Oman näkökentän korvaaminen virtuaalisella näkymällä johtaa moniin ongelmiin ihmiskehossa. Tällaisia ongelmia ei juurikaan ilmene muussa ohjelmistokehityksessä, joten aloittelevan VR-kehittäjän tulee käydä läpi mahdolliset virtuaalitodellisuuden aiheuttamat ongelmakohdat.

Yleisiä ongelmia ovat silmien rasitus, sekavuus ja pahoinvointi. Vaikutukset voivat näkyä eri ihmisisissä eri tavalla, mutta näihin voidaan vaikuttaa hyvällä suunnittelulla ja toteutuksella. Myös käyttäjäsyötteet poikkeavat normaaleista sovelluksista ja saattavat vaatia erityistä suunnittelua. Virtuaalitodellisuus on tekniikkana vielä suhteellisen tuore, ja tutkimusta aiheesta tehdään vielä, joten ratkaisut ja käytännöt muuttuvat nopeasti. [3]

3.1 Käyttäjäsyötteet

Kehittyneimmät HMD:t toimivat kuudessa vapausasteessa, kun taas mobiililaitteilla toimivat HMD:t käyttävät ainoastaan kolmea vapausastetta. 6DOF-laitteet voivat kääntymiskulmien lisäksi liikkua kolmessa ulottuvuudessa. 3DOF-ohjaimien tulee toteuttaa kolmiulotteisessa avaruudessa liikkuminen jollain muulla tavalla kuin ympäristössä kävelemällä. Tämä tarkoittaa yleensä erillistä ohjainta liikkumista varten. [4]

Luontevinta on tuoda käyttäjäsyötteet erillisiltä ohjaimilta, jotka tunnistavat oman asentonsa ja synkronoituvat virtuaaliseen näkymään, kuten esimerkiksi laserohjaimella. Laserohjain tulee piirtää näkymään ja laserohjaimen osoittimen suunta visualisoida käyttäjälle. Ainoastaan jos asentoon perustuvia ohjaimia ei ole mahdollista käyttää, tulee käyttäjän valintoja tarkastella pään suunnan perusteella esimerkiksi tuijotustoiminnolla (engl. gaze). Tämä tarkoittaa esimerkiksi painikkeen painamista katsomalla sitä tietyn aikaa. [4]

Kuten liikkuvassa autossa, pahoinvointia voi lieventää se, että käyttäjä on itse vastuussa liikkeestä ja osaa siten reagoida muutoksiin. Ohjelmallisesti aiheutettu kiihtyvyys aiheuttaa käyttäjälle herkästi epämukavuutta. Tämä tarkoittaa kaikkea perspektiivin muutoksen aiheuttamaa liikettä, joka ei ole lähtöisin pään asennosta tai laserohjaimen syötteestä. Koska liikkumista kolmiulotteisessa ympäristössä ei voi 3DOF-ohjaimella toteuttaa, tulee tämän toteutuksen kanssa olla tarkkana. [5]

3.2 Näyttö

Virtuaalisovelluksen ei välttämättä tarvitse olla realistinen ollakseen miellyttävä silmälle, mutta sen on toimittava mahdollisimman korkealla näyttötarkkuudella ja pystyttävä ylläpitämään korkea FPS ilman pudotuksia. Latenssi käyttäjien syötteiden ja kuvan päivityksen välillä tulisi pitää mahdollisimman pienenä sekä mahdollisimman vakiona. [6]

Kirkkaat kuvat voivat myös ärsyttää silmiä, joten tummemmat näkymät ovat suositeltavampia, ja luonnollisesti kaikki, mikä normaalisti häiritsee ihmisnäköä, on minimoitava virtuaalisessa ympäristössä, kuten valojen välkkymiset ja nopeat koko näytön päivitykset [6]. Siten esimerkiksi eri siirtymiset, kuten pelin käynnistys, pelin lopetus ja eri valikot, on hyvä häivyttää ja tuoda esiin mahdollisimman tasaisesti esimerkiksi pimentämällä ja himmentämällä ruutua.

Näkymään vaikuttavat jälkiprosessointitekniikat tulee lisätä molemmille silmille, ja eri silmien näkymät tulisi poiketa ainoastaan perspektiivinsä puolesta. Monet perinteiset jälkiprosessointitekniikat eivät toimi stereonäössä, ja joitain haittavaikutuksia voi syntyä. [7]

Näkökenttään staattisesti liitetyt käyttöliittymäelementit koetaan näkökenttää häiritseviksi ja epämiellyttäviksi, joten usein valikot ja muut käyttöliittymäelementit on hyvä tuoda mahdollisuuksien mukaan osaksi virtuaaliympäristöä. Elementit tulisi kuitenkin asetella ja ryhmitellä siten, että käyttäjä ei joudu ponnistelemaan pään asennon kanssa päästäkseen valikosta toiseen. Näkymää ei tule lukita paikoilleen siten, että näkymä ei enää reagoi käyttäjän pään asentoon. Tämä tarkoittaa, että esimerkiksi latausruudut tulee suunnitella erilliseksi kolmiulotteiseksi ympäristöksi, jossa käyttäjä voi vielä liikutella päätään ja vaihtaa kameran perspektiiviä odottaessaan. Lisäksi toteuttajan tulee huolehtia, että ympäristön horisontti pysyy aina tasaisesti paikallaan, sillä tämä voi aiheuttaa käyttäjälle sekavuutta. [7, 8]

3.3 Laitteistovaatimukset

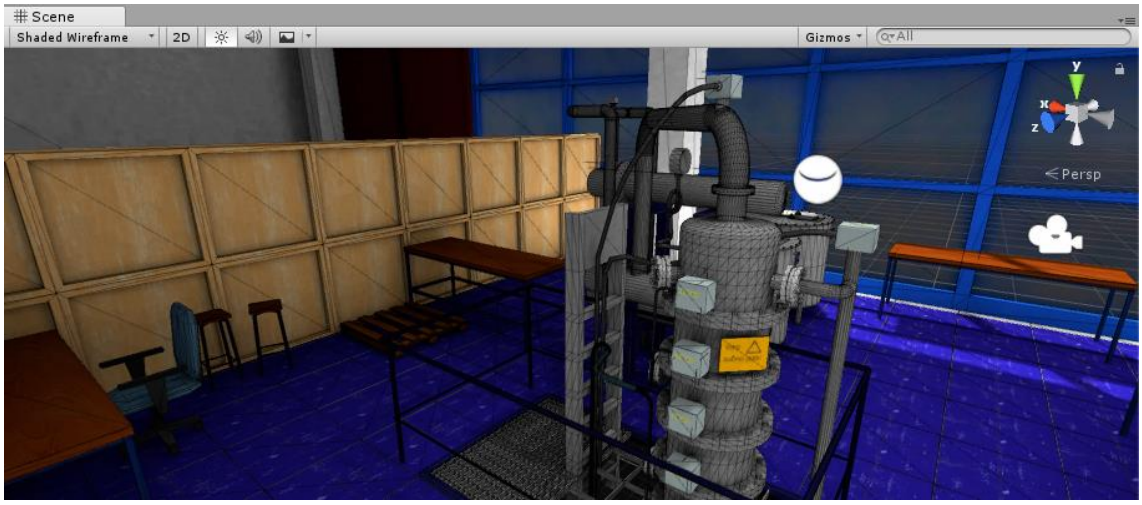
Koska molemmille silmille täytyy piirtää ympäristö erikseen, korkea vaadittu näyttöresoluutio ja korkea FPS-vaatimus aiheuttavat virtuaalitodellisuussovelluksille korkeat laitteistovaatimukset, joita voidaan pitää huomattavina jopa nykyaikaisille tietokoneille. Älypuhelinten suorituskyky ei riitä kuin varsin yksinkertaisiin ja hyvin optimoituihin sovelluksiin. Lisäksi olisi hyvä jos sovellus ei kuormittaisi puhelinta täydellä teholla jatkuvasti, sillä puhelimen ollessa poissa käyttäjän käsistä on ylikuumenemista vaikea havaita.

4. UNITY

Unitystä julkaistaan tyypillisesti muutama uusi versio vuodessa ja jotkut esitellyistä konsepteista ja optimointimenetelmistä saattavat olla vanhentuneet. Unity tarjoaa sivuillaan versioidun dokumentaation. Projektissa on käytetty Unityn versiota 2017.2.

4.1 Näkymä

Näkymä (engl. Scene) on kokonaisuus, joka kokoaa yhteen joukon elementtejä. Näkymän laajuus on mielivaltainen ja se riippuu käyttötarkoituksesta ja toteuttajasta. Koko sovelluksen voi halutessaan tehdä yhteen näkymään mutta sovelluksen eri osa-alueiden kuten kenttien ja valikoiden toteuttaminen omiin *.scene*-tiedostoihinsa helpottaa ylläpidettävyyttä. Näkymää voidaan muokata ja tarkastella erillisestä näkymä-ikkunasta (kuva 1).



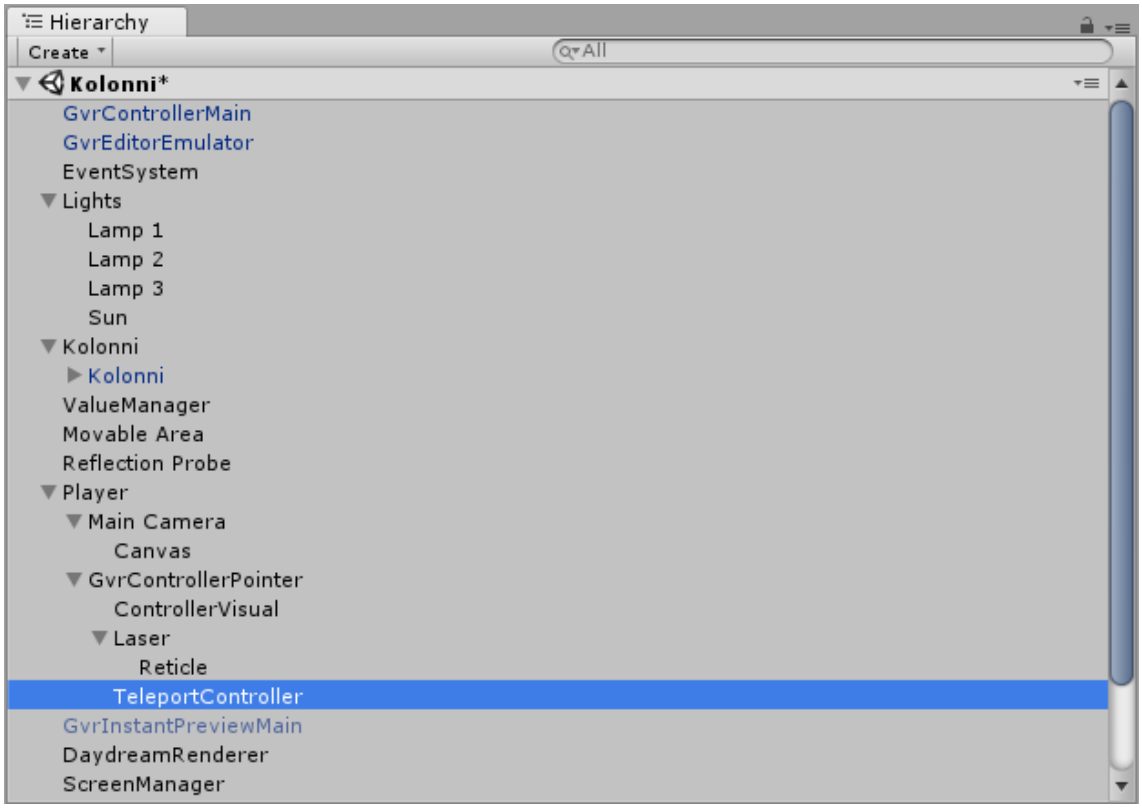
Kuva 1. *Scene-ikkuna*

Sovelluksen kääntäminen vaatii vähintään yhden näkymän, joka lisätään näkymälistan ensimmäiseksi projektin *Build*-valikosta. Tämä on yleensä sovelluksen päävalikko. Sovellus käynnistyy tähän näkymään, jonka jälkeen näkymiä voidaan ladata ruudulle edellisen tilalle tai lisätä edelliseen näkymän lisäksi toinen näkymä. Asynkroninen näkymän lataus mahdollistaa toiminnallisuuden lataamisen ympäristöön vasta sitä tarvittaessa, jolla saadaan aikaseiksi tehohyötyjä tietyissä tilanteissa. Esimerkiksi ympäristön ollessa kaupunki voidaan näkymän alustuksessa ladata ainoastaan rakennusten ulkoseinät ja jälkeenpäin ladata asynkronisesti rakennusten sisältö, helpottaen ensisijaista latausaikaa.

Näkymät tallennetaan omiin tiedostoihinsa projektin kansiorakenteessa, jonka avattua Unity päivittää projektissa esillä olevan projektihierarkian. *Scene*-näkymä päivittyy vastaamaan näkymän objektihierarkiaa ja *Game*-ikkuna näyttää sovelluksen pääkameran sisällön. Uusi näkymä tiedosto sisältää oletuksena kameran ja valonlähteen. Näkymä tiedoston rinnalle syntyy saman niminen kansio, joka sisältää Unityn luomia näkymään liittyviä resursseja.

4.2 Peliobjekti

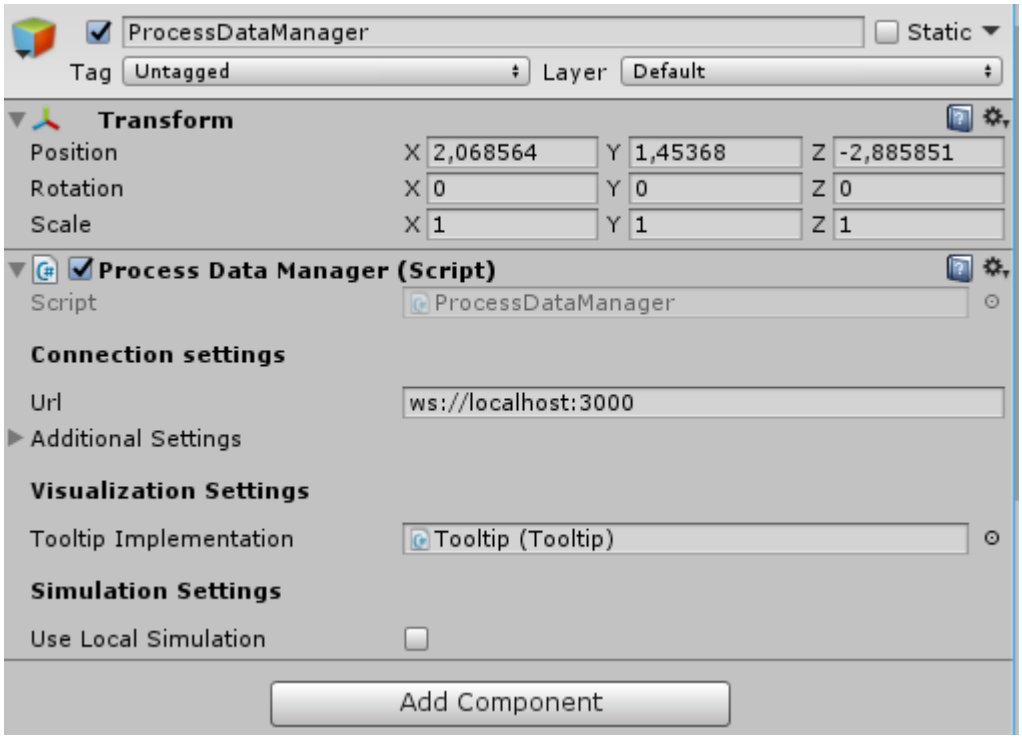
Näkymähierarkia muodostaa puumaiseen rakenteen, jossa ylimmässä tasossa ovat avatut näkymät ja näiden alle kasatut peliobjektit (engl. GameObject). Kuvassa 2 on esitetty ”Kolonni” niminen näkymä ja sen sisältämät peliobjektit. *GameObject* on Unityn perusluokka, joka toimii pohjana näkymähierarkiassa oleville objekteille. Peliobjektiin voidaan liittää eri komponentteja, jotka määrittävät miten peliobjekti toimii näkymässä. Peliobjektilla on aina vähintään *Transform*-komponentti, joka osoittaa objektin sijainnin, mittakaavan ja kierron 3D-tasossa sekä peliobjektin vanhemman hierarkiassa.



Kuva 2. *Näkymän peliobjektihierarkia*

Peliobjekti voi sisältää yhden tai useamman komponentin kuten kameran, polygonimallin, valonlähteen, äänilähteen tai komentosarjan. Komponenttien arvoja voi säätää kom-

ponentin sisältämän peliobjektin valitsemisen jälkeen *Inspector* -näkymästä kuten kuvattu kuvassa 3. Peliobjekteja voi tallentaa myös tiedostojärjestelmään, jolloin peliobjekti ja sen sisältämät komponentit arvoineen ja lapsi peliobjekteineen tallentuvat projektissa yleisesti käytettäviksi. Näitä kokonaisuuksia voi muokata keskitetysti päivittäen kaikki viittaukset kyseiseen peliobjektiin.



Kuva 3. *Tarkastelijanäkymä*

4.3 Komentosarjat

Unity tukee kahta ohjelmointikieltä: C# ja UnityScript. UnityScript on JavaScriptin pohjalta luotu kieli, jonka tuki loppuu Unityn versiossa 2018.2. UnityScript muistuttaa syntaksiltaan JavaScriptia, mutta on luokkineen ja suojaustasoineen enemmän oliomallinen kuin JavaScript. JavaScript ei suoraan käänny joten ulkoiset JavaScript kirjastot eivät suoraan toimi Unityssä. Projektiin voi myös sisällyttää muita .NET kielillä käännettyjä kirjastoja .dll muodossa. [9]

Peliobjekteille kirjoittaessa omia komponentteja tulee kehittäjän komponenttitoteutuksen periä *MonoBehaviour*-luokka, joka mahdollistaa koodin ajamisen Unityn elinkaarimetodeissa. Normaalisti koodia ajetaan esimerkiksi peliobjektin alustuksessa tai joka ruudunpäivityksellä kutsuttavasta elinkaarimetodista. Komponenttien julkiset ominaisuudet ovat alustettavissa Unityn kautta.

Skriptien ajojärjestyksen voi määrittää editorista, mutta Unity määrää tietyt säännöt tietyn nimisille kansioille. Oleellisimpina *Resources*-kansio, joka pakataan aina sovelluksen mukana ja *Editor*-kansio, jota ei sisällytetä rakennettuun versioon. Resources-kansion sisältö sisällytetään aina rakennettuun sovellukseen, joten sen sisältöä voi siten ladata ajonaikaisesti. Editor-kansion skripteihin voi määrittää toiminannallisuutta, joka laajentaa Unityn editorin toiminnallisuutta, jota ei tarvita enää ohjelman käännösvaiheessa. *Editor*-kansioiden ohjelmakoodit karsitaan aina automaattisesti lopullisesti sovelluksesta.

4.4 Materiaali

Materiaali (engl. Material) on Unityssä komponentti, joka määrittää miten eri objektien pinnat renderöidään. Materiaalin lähtökohtana on varjostin (engl. Shader), johon määritetään tekstuurit ja eri parametrit, kuten esimerkiksi pinnanmuotokartat (engl. Normal map), yksityiskohtakartat (engl. Detail map) sekä valaisuun ja heijastuksiin liittyvät asetukset. Varjostin on oma näytönohjaimelle kirjoitettu koodi, joka määrittää miten jokin kappale piirretään. Materiaalit koostuvat yleensä useasta tekstuurista, jotka vievät tietoa varjostimelle.

Varjostimen valinta on oleellista suunniteltaessa suorituskykyistä sovellusta. Unityn mukana tuleva PBR-varjostin (Physically Based Rendering) on moderni tapa tehdä pintoja pelikehityksessä. Se voi olla liian raskas mobiilisovelluksille, sillä monimutkaiset varjostin toteutukset lisäävät näytönohjaimen kuormitusta.

Projektissa on käytetty Daydream-alustalle erityisesti suunnattua varjostinta, joka on parametroitu tehokkaaksi. Mahdollisuuksia on esimerkiksi käyttää tekstuuripohjaisten varjojen sijasta nurkkapistepohjaisia varjoja, jotka voidaan asettaa paikalleen suoraan 3D-malliin. Lisäksi tehoa on säästetty käyttämällä eri RGBA-tekstuurien läpinäkyvyys-arvoja varjostimen parametrina, erillisen harmaasävykartan sijaan. *Daydream Shader* on Unityn kehittämä avoimen lähdekoodin projekti, jonka lähdekoodi löytyy Unity VR GitHub sivulta.

4.5 3D-ympäristö

Polygonimalli *(*engl. polygon mesh), koostuu nurkkapisteistä (engl. vertex) 3D-koordinaatistossa, jotka yhdistetään toisiinsa kolmioilla (engl. triangle face) muodostaen pinnanmuodon. Pisteeseen voidaan tallentaa myös muuta tietoa kyseiseen sijaintiin pinnalla liittyen, kuten pisteen normaali (engl. normal), uv-koordinaatti (engl. uv coordinate) ja väri (engl. vertex color).

Pinnan normaalit määrittävät miten valo heijastuu mallin pinnasta renderöitäessä ja niiden automaattiseen määrittämiseen on mallinnusohjelmissa useita työkaluja. Normaalien avulla voidaan huijata pyöreämuotoinen polygonimalli näyttämään todellisuutta tarkemmalta, asettamalla pinnanmuodot sileäksi normaalien avulla sekä lisäämään materiaaliin pinnanmuotoja lisäämällä päälle erillinen pinnanmuotokartta.

Uv-koordinaatit määrittelevät joukon pisteitä 2D-koordinaatistossa, joita käytetään erilaisten tekstuurien sijoittamiseen polygonimallin pinnalle. Uv-koordinaattien määrittäminen on oleellinen osa mallinnusprosessia. Näiden harkittu sijoittelu helpottaa tekstuurien piirtämistä sekä auttaa piilottamaan saumakohtia tekstuurin reunoissa. Mallinnusohjelmissa löytyy työkaluja näiden automaattiseen luontiin esimerkiksi pinnanmuotojen mukaan.

Nurkkapisteen väri on pisteeseen liittyvä RGBA-arvo. Kyseessä ei kuitenkaan ole välttämättä mallin tekstuurin väri kyseisessä pisteessä, vaan väri tietoa voidaan käyttää varjostimessa esimerkiksi tekstuurien yhdistämiseen, varjoihin tai suuntavektorina eri operaatioille.

Polygonimallin toteuttaminen hyvien käytäntöjen mukaisesti vähentää mahdollisuuksia erilaisten graafisten artefaktien syntymiseen sovelluksessa. Toteuttajan tulee erityisesti huomioida, että polygonimalli on suljettu, eli normaalien toisella puolella olevia tasoja ei ole näkyvissä. Tämä aiheuttaa valojen esilaskennan virheitä Unityssä. Lisäksi uv-karttaa määritellessä tulee pitää huoli, että eri uv-koordinaatti saarien välillä on tarpeeksi tilaa sillä tämä voi aiheuttaa saumoja varjoihin. [10]

4.6 Valot

Unity tarjoaa kahden tyyppisiä valoja: reaaliaikaisia ja esilaskettuja. Reaaliaikaiset valot päivittyvät joka piirtokomennolla. Tämä tarkoittaa, että varjojen sijainti ja voimakkuus päivittyvät näkymän valojen ja varjoja luovien objektien mukaisiksi reaaliaikaisesti. Tämä vaatii huomattavasti enemmän laskentatehoa ja optimointia sovellukselta.

Esilasketut valot luodaan valaistuskartoiksi ennen pelin rakennusvaihetta. Valojen esilaskenta suoritetaan ainoastaan varjoja vastaanottaville objekteille, jotka ovat merkitty staattisiksi. Objektin merkintä staattiseksi on lupaus siitä, että objekti ei vaihda sijaintiaan, asentoaan tai kokoaan. Esilasketuille valoille on yleiset asetukset, joihin voidaan määritellä resoluutio, voimakkuus ja useita eri tarkkuuteen liittyviä parametreja. Tarkkuuteen liittyvät parametrit ovat yleensä kompromissi laadun ja laskenta-ajan välillä, joten kehityksenaikana on hyvä käyttää matalalaatuisia valaistuskarttoja ja säätää asetukset

korkeammalle vasta grafiikka-asetuksia viimeisteltäessä. Asetuksia voi säätää myös objektikohtaisesti esimerkiksi keskittäen tarkimmat valaistusasetukset keskeisille objekteille.

5. OPTIMOINTI

Optimointi on tärkeää virtuaalitodellisuussovelluksissa ja vielä tärkeämpää tähdätessä puhelinalustoille. Se mikä optimointitekniikka on tehokkain riippuu sovelluksesta ja yleinen mielipide on, että optimointi tulisi jättää sovelluksen kehityksen loppuvaiheille. Tehokkaimpia optimointimenetelmiä on hankala ennustaa etukäteen koska se riippuu toteutettavasti sovelluksesta. Monet optimointitekniikat vaativat kuitenkin huomioonottamista jo aikaisessa kehitysvaiheessa. Tästä johtuen optimointitekniikat on hyvä tietää etukäteen.

5.1 Tekstuurikartasto

Tekstuurikartasto (engl. Texture Atlas) on tekstuurikokoelma, johon on kerätty useita eri tekstuureja, muodostaen yhden isomman tekstuurin. Tämä voidaan tehdä käsin tai ohjelmallisesti. Lisäämällä mallin käyttämä tekstuurikartalle ja säätämällä mallin uv-koordinaatit vastaamaan mallin tekstuuria uudella tekstuurikartalla, voidaan luoda uusi tekstuuri, josta kaikki tähän liitetyt mallit käyttävät vain osaa. Toistamalla tämä kaikille varjostimen käyttämille tekstuureille, voidaan luoda yhteinen materiaali komponentti usealle mallille. Tämä helpoittaa piirtokutsujen yhdistämisessä.

Hyvänä käytäntönä voidaan pitää resoluutioiden käyttöä, jotka ovat neliöitä ja resoluutioltaan kahden potenssia. Tämä käytäntö helpottaa myös tekstuurikokoelmien tekemistä, sillä tekstuureja on silloin helpompi jakaa kartalle. Kaikkia tekstuureja ei kuitenkaan kannata yhdistää sillä syntyneistä tekstuureista voi tällöin syntyä liian suuria. Näkymästä riippuen voi sovellus joutua lataamaan suuren 4K -resoluutioisen tekstuurikartan tarvitessaan vain pientä osaa siitä. Lisäksi joitain malleja ei ole helppoa käyttää tekstuurikartassa johtuen uv-koordinaattien päällekkäisyydestä tai sijoittumisesta [0-1, 0-1] alueen ulkopuolelle.

5.2 Näkökenttä- ja näkyvyysleikkaus

Näkökenttäleikkaus (engl. Frustum culling) on Unityn automaattisesti suorittama optimointiprosessi, joka poistaa käytöstä polygonimallit, jotka ovat kamera kuvakulman ulkopuolella. 3D-ympäristö saattaa lisäksi sisältää paljon päällekkäisiä objekteja, joita piirtää ruudulle. Näkyvyysleikkaus (engl. Occlusion culling) on optimointiprosessi, jossa sovellus ajonaikaisesti tarkastelee sovelluksen piirtämää aluetta ja jättää toisten objektien taakse kokonaan jäävät objektit piirtämättä. Tätä voidaan käyttää näkökenttäleikkaus

operaation kanssa samaan aikaan. Näkökenttäleikkauksen Unity hoitaa itse, mutta näkyvyysleikkaus operaatio vaatii kehittäjältä asetuksia. [11]

Jotta Unityn näkyvyysleikkaus voi päätellä mitkä objektit se voi piilottaa, täytyy staattisille polygonimalleille määrittää ominaisuudet *Occluder Static* tai *Occludee Static*. Occluder Static -ominaisuus on pääsääntöisesti käytetty merkintä näkyvyysleikkauksen toteuttaville objekteille. Tämä tarkoittaa, että tämän kappaleen takana olevien polygonimalleja ei tarvitse piirtää. Pienille ja läpinäkyville objekteille, jotka eivät todennäköisesti peitä mitään, voidaan merkitä *Occludee Static*. Tämä tarkoittaa, että tarkastuksia näiden takana oleville objekteille ei tarvitse tehdä, mutta kappale voi itse olla muiden kappaleiden takana. [11]

5.3 Piirtokutsujen yhdistys

Piirtokutsut (engl. Draw call) tarkoittavat prosessorin käskyä näytönohjaimelle piirtää jotain ruudulle. Jokainen uusi materiaali näkymässä täytyy piirtää näytönohjaimella erikseen ja valot sekä varjot voivat kasvattaa määrää entisestään. Piirtokutsuja tehdään useita jokaisen ruudunpäivityskerran aikana ja jokainen näistä vie jonkin verran aikaa prosessorilta. Koska stereoskopiaa hyödyntävien sovellusten on renderöitävä kaikki kaksi kertaa sekä pystyttävä ylläpitämään korkea kuvataajuus, on näiden minimointi tärkeää. Piirtokutsujen minimointiin on useita optimointitekniikoita, jotka liittyvät pääosin eri kerralla nähtävien materiaalien vähentämiseen.

Piirtokutsujen yhdistys *(*engl. Draw call batching) tarkoittaa Unityn suorittamaa optimointiprosessia, jossa yritetään automaattisesti yhdistää useita piirtokutsuja yhdeksi. Tämä voi tapahtua kahdella eri tavalla, riippuen piirrettävän objektin ominaisuuksista. Tapoja on staattinen yhdistys (engl. static batching), joka tapahtuu paikalla oleville staattisille kappaleille ja dynaaminen yhdistys (engl. dynamic batching), joka yhdistää tietyt ehdot täyttävät dynaamiset polygonimallit yhdeksi. [12]

Saman materiaalin jakavat polygonimallit, jotka ovat määritelty sovelluksessa staattisiksi, Unity yhdistää automaattisesti yhdeksi polygonimalliksi. Polygonimallit voi yhdistää myös 3D-mallinnusohjelmalla tai ohjelmakoodissa. Mallien yhdistämisessä ohjelmallisesti tai mallinnusohjelmassa tulee ottaa huomioon näkökenttä -ja näkyvyysleikkaus tekniikat. Laajan alueen kattava yhdistetty polygonimalli saattaa laskea suorituskykyä sen ollessa näkyvissä jatkuvasti. [12]

Unityn automaattinen polygonimallien yhdistys toiminnallisuus vaatii jonkin verran muistia, mutta toimii näkökenttä ja näkyvyysleikkaus -tekniikoiden kanssa sekä tekee objektien liikuttelusta Unityn sisällä helpompaa.

Dynaamisia polygonimalleja, jotka jakavat saman materiaalin voidaan yhdistää samaan piirtokutsuun, jos polygonimallit täyttävät tietyt ehdot. Polygonimalli tulee sisältää maksimissaan 300 nurkkapistettä ja 900 nurkkapisteattribuuttia. Esimerkiksi polygonimalli, jonka materiaalille on määritelty varjostin, joka käyttää sijainnin lisäksi normaalia, uv-koordinaattia sekä väriä voisi sisältää maksimissaan 180 nurkkapistettä. [12]

5.4 LOD

Unityn LOD-järjestelmä (engl. Level of Detail) mahdollistaa polygonimallien tarkkuuden säätämisen etäisyyden perusteella. Samasta kappaleesta tehdään monta polygonimallia, jotka ovat toinen toistaan yksinkertaisempia. Jokaisen mallin voi asettaa tiettyyn LOD-ryhmään, joka vastaa tiettyä etäisyysaluetta kamerasta. Siten malleja voidaan vaihtaa ajon aikaisesti tarkemmiksi kameran lähestyessä niitä. Tarkin malli asetetaan LOD 0 -ryhmään ja renderöidään ainoastaan kameran ollessa lähellä kappaletta. Ryhmien määrää lisäämällä saadaan tasaisempi häivytys eri mallien välillä mikä mahdollistaa sen, että käyttäjä ei välttämättä huomaa mallien vaihtumista. Toisaalta ryhmien määrän lisääminen lisää tarvittavaa mallintamistyötä.

Jotkin proseduraaliset mallin muokkaus työkalut osaavat tehdä approksimaatioita mallista automaattisesti, mutta lopputulokset eivät aina ole suoraan käytettävissä ja vaativat käsin muokkaamista. Jos polygonimallit nimetään käyttäen nimeämiskäytäntöä, jossa tiettyä tarkkuutta vastaavan polygonimalli nimetään päätteellä _LOD0, _LOD1 jne., Unity asettaa polygonimallit omiin ryhmiinsä nimen perusteella. [13]

5.5 Ohjelmakoodin optimointi

Monesti ohjelmakoodin tulee toimia jatkuva-aikaisesti sovelluksen ollessa käynnissä. Jotkin operaatiot kuten käyttäjän syötteet, kameran liikuttaminen ja liikkuminen tulee suorittaa jokaisella näytön päivityksellä. Tällöin kyseinen operaatio käydään läpi useita kymmeniä kertoja sekunnissa, joten on tärkeää, että prosessorin käyttämä aika näissä operaatioissa pysyy matalana. Tämä tarkoittaa monimutkaisen laskennan yksinkertaistamista ja uusien muistipaikkojen varaamisen välttämistä, sillä c#-ohjelmointikielessä automaattinen roskien keruu saattaa aiheuttaa piikkejä prosessorin käytössä. [14]

Jotkin operaatiot, jotka eivät vaadi reaaliaikaista päivitystä, voidaan määrittää laskettavaksi tietyin väliajoin, säästäen prosessorin kuormitusta. Unityn API ei takaa rinnakkaisuuden toimivuutta, mutta rinnakkaisia operaatioita voi edelleen käyttää omassa toteutuksessa, kunhan niissä ei kutsuta Unityn rajapintoja. [15]

Ohjelmakoodia kirjoittaessa on tärkeää kiinnittää huomiota muistin käyttöön, sillä useasti ajettava ohjelmakoodi, joka varaa paljon muistia joutuu vapauttamaan muistipaikat jossain vaiheessa, eli suorittamaan roskienkeruun. Roskienkeruun ja sovelluksen muistin käyttöä voi tutkia helposti nauhoittamalla sovellusta ajon aikaisesti *Unity profilerilla*. Tarvitut muistipaikat tulisi mahdollisuuksien mukaan varata sovelluksen ladatessa ja sen jälkeen käyttää uudestaan.

Jotkin Unityn sisäiset operaatiot ovat hitaita, ja näiden karsiminen usein ajettavasta koodista on vaatimus tehokkaalle sovellukselle. Näistä esimerkkinä toimivat monet peliobjektien sekä peliobjektien komponenttien löytämiseen tarkoitetut apufunktiot.

5.6 Objektiallas

Joskus on tarpeellista luoda ohjelmakoodissa kopioita ennalta määrätyistä peliobjekteista tai luoda ohjelmallisesti uusia. Tämä on suhteellisen raskas operaatio, joten tämän tekeminen ohjelman suorituksen aikana voi vaikuttaa negatiivisesti suorituskykyyn. Objektiallas (engl. Object Pooli) on menetelmä, jolla tämä ongelma voidaan ratkaista siirtämällä objektien alustus pelin alkuun, odottamaan myöhempää käyttöä. Tämä menetelmä vaatii arvion lisättävien resurssien samanaikaisesta maksimimäärästä. Menetelmää voidaan käyttää myös tavallisiin c# objekteihin, joiden tallentaminen dynaamiseen muistiin ja vapauttaminen muistista halutaan optimoida. Menetelmä estää myös muistin fragmentoitumisen muistipaikkojen osalta. [16, luku 19]

Resurssit luodaan sovelluksen käynnistäessä oletusarvoilla ja asetetaan pois päältä. Tällöin ne eivät vaikuta prosessorin suorituskykyyn, mutta ovat valmiina olemassa aktiivisessa näkymässä. Resursseja tarvittaessa objekti otetaan käyttöön objektialtaasta ja asetetaan tarvittavat arvot kuten sijainti. Vastaavasti kun resurssia ei enää tarvita, sen sijaan että se tuhottaisiin näkymästä ja poistettaisiin muistista, se asetetaan taas pois päältä ja asetetaan takaisin objektialtaaseen käytettäväksi. Tämä menetelmä vaatii kuitenkin ylimääräistä muistia riippuen siitä, kuinka paljon resursseja pidetään altaassa. Lisäksi resurssien alustaminen sovelluksen alussa lisää jonkin verran latausaikaa. [16, luku 19]

5.7 Profilointityökalut

Sovellusta kehittäessä on tärkeää tarkastella sovelluksen suorituskykyä ajonaikaisesti. Tätä varten on olemassa paljon eri näytönohjainvalmistajien ja muiden tahojen kehittämiä profilointityökaluja, joiden avulla voidaan tutkia Android laitteen suorituskykyä. [17]

Unityn mukana tulee myös tehokas suorituskyvynmittaustyökalu, joka voi analysoida suorituskykyä sekä suoraan kehitysympäristöstä ajettuna että yhdistettynä laitteeseen. *Unity Profiler* kerää tilastoa eri sovelluksen suorituskyvyn osa-alueista, jotka näytetään kehittäjälle omissa graafeissaan. Sen avulla löytää helposti suorituskyvyn ongelmakohdat ja voi keskittää optimoinnin siihen osa-alueeseen, joka on pullonkaulana suorituskykyisen sovelluksen kehityksessä (kuva 4).

Kuva 4. *Unity Profiler helpottaa löytämään piikit prosessorin, muistin ja näytönohjaimen käytössä*

Profiler toimii taustalla sovellusta ajettaessa ja kerää reaaliajassa kuvaaja sovelluksen suorituskyvystä. Kuvaajista voi sen jälkeen tarkkailla piikkejä suoritusajassa ja tarkkailla mikä on aiheuttanut piikin kyseisellä hetkellä. Unity Profilerin voi asettaa nauhoittamaan myös kaikkien komentokehotteiden suoritusta. Tämä kuitenkin vaatii ylimääräistä laskentatehoa eikä anna profiloinnin aikana enään oikeata kuvaa sovelluksen suorituskyvystä. Tämä on kuitenkin hyvä työkalu tutkiessa oman ohjelmakoodin suhteellista tehokkuutta.

5.8 Frame Debug

Frame Debug -ikkuna tarjoaa tarkempaa tietoa siitä, miten ja missä järjestyksessä näkymä piirretään. Sovelluksen voi pysäyttää kesken ajon ja tarkastella ruudunpäivitys kerrallaan, mistä näkymän muodostavat piirtokutsut syntyvät.

Näkymä muodostaa listan operaatioista, jotka aiheuttavat piirtokutsun siinä järjestyksessä, kun ne ilmenevät. Jos operaatioon liittyy näkymässä esiintyvä peliobjekti, se korostetaan projektin hierarkia -ikkunassa. Näkymässä myös selitys miksi operaatiota ei

voitu ohjelmallisesti yhdistää edelliseen, joka on hyödyllinen tieto tutkiessa miten eri piirtokutsut muodostuvat.

6. TOTEUTUS

Google tarjoaa ohjeet projektin aloittamiselle Unityllä suoraan VR kehitystä koskevilla sivuillaan. Unity sisäänrakennetusti tukee mobiilialustoille rakentamista mutta virtuaalitodellisuus ominaisuuksia varten tulee ladata Google VR Unity SDK. Lisäksi Unityn Asset Storesta ja Google VR Github repositoriosta löytyy paljon hyödyllisiä työkaluja VR kehitykseen. Daydreamille löytyy esimerkiksi valmiit toteutukset Daydream-ohjaimen käyttöönotolle ja Daydreamille räätälöityjä varjostin toteutuksia.

6.1 Tislauskolonni

Tislauskolonnia käytetään toisiinsa liuenneiden aineiden erottamiseen. Erottelu perustuu erotettavien aineiden eri haihtuvuuksiin. Höyry nousee kolonnia ylöspäin ja tiivistyy välikerroksissa. Alhaisemmassa lämpötilassa kiehuva neste erottuu höyryn mukana ja jatkaa seuraavaan välikerrokseen. Näin pitoisuus nousee ylöspäin mennessä ja päätyy lopulta tislesäiliöön.

Kolonni sisältää antureita, joiden arvoja voidaan havainnollistaa virtuaalisessa ympäristössä numeerisesti antureiden kohdalla. Tislauskolonnia on mahdollista käydä tutkimassa paikan päällä ja ottaa mittoja sekä referenssikuvia. Referenssikuvan ja toteutuneen mallin ero on havainnollistettu kuvassa 5.

Kuva 5. *Tilauskolonni ja siitä kehitetty 3D-malli*

6.2 3D-ympäristön toteutus

Sovelluksen kehityksessä on hyödyllistä, jos kehityksen iteraatioaika pysyy pienenä. Sovellusta voi ajaa muokkaustilassa suoraan Unityn sisällä. Tämä on riittävää suurimmassa osassa kehitystä, mutta virtuaalitodellisuuteen liittyvä kehitys, kuten ohjaimen toiminnallisuus sekä sovelluksen suorituskyky, tulee pystyä testaamaan oikealla laitteella.

3D-ympäristö on mallinnettu Blender-ohjelmistolla. 3D-mallin toteutus on oleellinen osa sovelluksen suorituskykyä, joten sovellusta toteuttaessa on vuoroteltu usein Unityn ja Blenderin välillä. Blenderillä on luotu eri LOD-tasojen versiot kolonniympäristöstä, joskin ne eivät pääse täysin oikeuksiin nykyisessä valmiiksi ahtaassa ympäristössä. LOD-tasojen merkitys kuitenkin kasvaa, jos ympäristö jatkokehitetään suuremmaksi. LOD-tasoja on luotu käyttäen Blenderin *decimate*-työkalua, jonka avulla voidaan tarkasta polygonimallista helposti luoda suorituskykyisempiä versioita muutaman parametrin avulla.

Tekstuurit on toteutettu Substance Painter -ohjelmistolla, jolla pystyy proseduraalisesti generoimaan tekstuureja polygonimallin datan perusteella. Tämä on maksullinen ohjelmisto mutta ei pakollinen sovelluksen jatkokehityksessä. Tekstuureja voi ladata internetistä ilmaiseksi tai tehdä itse suosimallaan piirtosovelluksella. Myös normaalikarttoja voi luoda ilmaisilla ohjelmilla ja laajennuksilla eri piirtosovelluksiin.

Tekstuurikartasto on toteutettu siirtämällä uv-koordinaatit Blenderin sisällä käsin ja yhdistämällä Substance Painterilla luodut tekstuurit vastaaville koordinaateille Photoshopilla. Tämän tekeminen käsin on jossain määrin työlästä, joten automatisointi on suositeltavaa. Tähän tarkoitukseen on olemassa työkaluja Unityn Asset Storessa.

6.3 Unity

Koska sovelluksen ympäristö ei sisällä liikkuvia kappaleita, on koko ympäristö merkattu staattiseksi. Sovellus käyttää laajalti hyväksi esilaskettuja valoja. Koska esilasketuilla valoilla saadaan tuotettua jo suhteellisen uskottava ympäristö ilman suurta vaikutusta sovelluksen tehokkuuteen, voidaan laskentatehoa käyttää hyväksi piirtämään korkean resoluution tekstuureja tai käyttää tehokkaampia jälkiprosessointiefektejä.

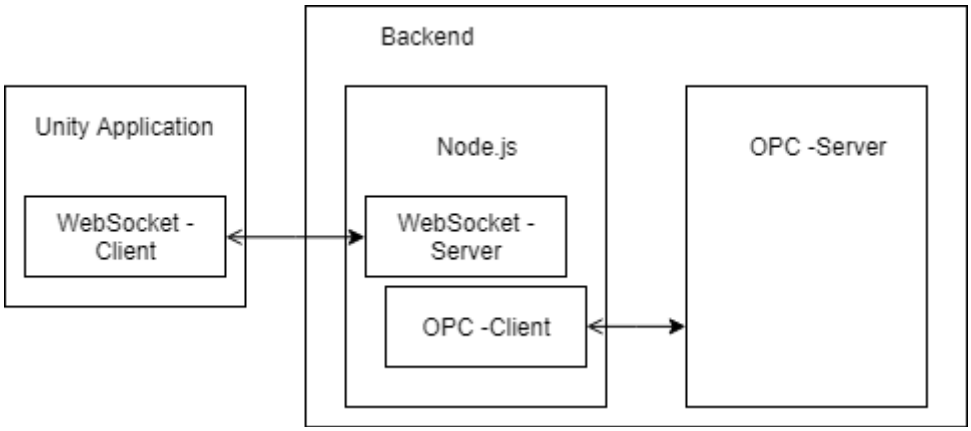
Omia komponentteja projektissa toteutettiin tiedonsiirtoa ja näkymän hallintaa varten. Suurin osa kehitystyöstä keskittyi Unityn ulkopuolelle mallinnukseen ja erilaisten optimointimenetelmien vertailuun. Työn manuaalisen toistamisen välttämiseksi projektissa on toteutettu myös yksinkertainen komentokehoite asettamaan tiettyjä parametreja automaattisesti kuten esimerkiksi LOD-ryhmien kynnysarvot.

Projektissa hyödynnettiin Googlen kehittämiä valmiita toteutuksia stereoskooppisen renderöinnin osalta. Käyttäjäsyötteiden vastaanottamista, laser-ohjaimen renderöintiä sekä liikkumista varten käytettiin valmiita toteutuksia. Lisäksi käytettiin erityisesti Daydreamille-suunnattua valmista varjostin toteutusta. Nämä ja monet muut komponentit ovat varsin monimutkaisia kehittää yksin joten kehittäjän tulisi suosia valmiita toteutuksia ennen oman kehittämistä.

Kehityksen apuna on mahdollista käyttää Googlen *Instant Preview* -komponenttia, joka mahdollistaa nopeamman iteraatioajan Unityn kehitysversion ja puhelinapplikaation välillä. Komponentin avulla rakennus prosessin voi jättää tekemättä ja suoratoistaa sovelluksen suoraan puhelimen näytölle USB-johdon tai WLAN:in välityksellä. Laskenta tapahtuu kuitenkin tietokoneen komponenteilla ja ainoastaan kuvata lähetetään puhelimeen, joten tämä ei sovellu suorituskyvyn testaukseen.

6.4 Palvelinympäristön toteutus

Applikaatio yhdistää käynnistyessään Node.js palvelimeen WebSocket-asiakassovelluksella ja synkronoi applikaation ja palvelimen välisen datan. WebSocket on tiedonsiirto protokolla, joka mahdollistaa reaaliaikaisen kaksisuuntaisen tiedonsiirron TCP-yhteyden yli. OPC UA -palvelin simulaatio muuttaa sisältämiään arvoja satunnaisesti lyhyin väliajoin. Arvot lähetetään muuttuessaan Node.js palvelimelle, joka välittää ne Unity applikaatiolle. Asiakas-palvelin-arkkitehtuuria on havainnollistettu kuvassa 6.



Kuva 6. Sovelluksen asiakas - palvelin ympäristö

Node.js palvelin sisältää simuloitavien arvojen lisäksi yksikön, joka näytetään arvojen perässä, sekä sijainnin 3D-koordinaatistossa, jolloin datapisteitä voidaan lisätä helposti ympäristöön tekemättä muutoksia puhelimessa ajettavaan ohjelmakoodiin.

OPC UA -asiakassovelluksen voisi toteuttaa myös suoraan .NET ympäristössä ja ajaa Unityllä. Sen toimivuus vaikuttaa kuitenkin olevan riippuvainen käytetystä versiosta, valitusta OPC UA -kirjastosta sekä .NET ajoympäristöstä. Uudemmissa Unity versioissa lisätään tuki myös .NET Standard 2.0: lle. [18]

6.5 Sovellus

Projektin lopputuloksena syntyi toimiva Android sovellus, joka asentuu ja toimii odotetusti puhelimella. Sovelluksessa käyttäjä pystyy Daydream-ohjaimen avulla liikkumaan kolonnin sisältämässä salissa ja tutkimaan taustalla olevan palvelimen lähettämiä arvoja reaaliajassa.

Tehokkuutta tutkittiin ajamalla sovellusta puhelimella yhdistettynä USB-johdolla tietokoneeseen profilointidatan keräämiseksi. Sovelluksesta etsittiin kohta, jossa näytönohjain ja prosessori joutuivat suurimman kuorman alle. Piirtokutsujen maksimimäärä jäi huomattavasti alle hyväksyttävänä pidetyn rajan (28, kun tavoite on 50-100 tai vähemmän) ja kolmioiden määrä oli myös rajojen sisällä (88 000, kun tavoite on 50 000 – 100 000 tai vähemmän). WebSocket-asiakastoteutuksen arvojen hakeminen ja päivittäminen ympäristöön muodostivat välillä pienen piikin prosessorin käytössä, mutta ei laskenut FPS alle halutun rajan. Tavoite arvoina käytettiin Oculus-yhtiön dokumentaatiossa julkaisemiaan arvoja. [19]

FPS pysyy tasaisesti korkeana eikä pudotuksia ole havaittavissa. Puhelin ei myöskään kuumene pitkänkään käyttöajan jälkeen. Tämä on saatu aikaiseksi pitämällä piirtokutsujen määrä mahdollisemman pienenä. Tähän vaikuttaa eniten päätös käyttää ainoastaan staattisia objekteja läpi sovelluksen sekä käytetyt tekstuuri kartastot. Lisäksi käytetty varjostin oli laskennaltaan yksinkertainen mikä vähensi näytönohjaimen kuormitusta entisestään.

Liikkuminen ympäristössä tapahtuu laserosoittimen avulla, joka mahdollistaa käyttäjän siirtymisen osoittimen päähän napin painalluksella. Lisäksi käyttäjän asentoa voidaan säätää ohjaimella, joka mahdollistaa sen, että käyttäjän ei halutessaan tarvitse kääntyä vastakkaiseen suuntaan liikkuakseen taaksepäin.

Sovellus käyttää SD-kortille tallennettua XML-tiedostoa, johon voidaan säätää mille palvelimelle sovellus yhdistää hakeakseen kolonnin datan. Tämän muokkaamiselle ei kuitenkaan löydy sovelluksesta itsestään käyttöliittymää, vaan sen muokkaaminen jää käyttäjän vastuulle androidin tiedostojärjestelmästä käsin.

6.6 Tulevaisuus

Projektiin on tulevaisuudessa tarkoitus jatkaa toteuttamalla 3D-ympäristö tarkempana. Lisäksi ympäristöstä puuttuu laboratorion alakerta. OPC UA -asiakassovellus on tarkoitus yhdistää OPC UA -palvelimeen, joka tarjoaa oikeaa dataa kolonni ympäristön antureista. Numeeristen arvojen lisäksi olisi mahdollista myös kuvata tietovirtoja. Projektiin voitaisiin myös toteuttaa toimintoja, jotka mahdollistaisivat interaktion virtuaalisen kolonnin kanssa.

Grafiikan kannalta projektissa voitaisiin tutkia mahdollisuuksia käyttää pintojen heijastuksia varjostimessa, sillä ympäristö sisältää paljon metallia ja kiiltäviä pintoja. Tämä voisi lisätä sovelluksen näyttävyyttä. Jälkiprosessointiefektejä voitaisiin myös lisätä parantamaan ympäristön uskottavuutta. Unity mahdollistaa sovelluksen rakentamisen monelle alustalle. Sovellusta ja sen ohjain toiminnallisuutta voisi jatkokehittää eri VR-alustoille, joissa on enemmän laskentatehoa kuin puhelimessa. Unityn uudemmat versiot voisivat tuoda projektille uusia ominaisuuksia ja tehokkuusparannuksia.

Kolonnin 3D-mallinnus toteutettiin käsin hyödyntäen paikan päältä otettuja referenssi kuvia. Mallin tarkkuutta voitaisiin parantaa käyttämällä fotogrammetriaa. Fotogrammetria tarkoittaa mallin luomista digitaalisten kuvien perusteella automaattisesti. Fotogrammetriasta on tullut varteenotettava vaihtoehto 3D-sisällön luontiin ohjelmistojen kehittyessä. Fotogrammetria ohjelmistolle syötetään mahdollisimman paljon kuvia kohteesta eri kulmista, joista ohjelmisto yrittää päätellä mallin muodon ja muodostaa siitä 3D-mallin automaattisesti. Prosessiin riittää huonokin kamera, tärkeintä on saada paljon kuvia kohteesta ja mahdollisemman monesta eri kuvakulmista. Lopputulosta saattaa joutua hieman jälkikäsittelemään 3D-mallinnus ohjelmalla jälkikäteen, jos lopputuloksessa on virheitä tai jos malli vaatii optimointia.

Unityn uudemmissa verioissa on tullut ominaisuuksia, jotka voivat olla hyödyllisiä projektissa. Ohjelmoitavat renderöinti putki (engl. Scriptable Rendering Pipeline) mahdollistaa Unityn syvemmän renderöinti logiikan muokkauksen, joka ei ollut aikaisemmin mahdollista. Tämän mukana tulee mahdollisuus käyttää Unityn luomia valmiita renderöintiputkia. Valitsemalla renderöintiputki, joka on luotu kevyiden ja graafisesti vaatimattomien sovellusten luomiseen, saadaan aikaiseksi tehohyötyjä. Lisäksi uusi DOTS-järjestelmä (Data-Oriented Technology Stack) mahdollistaa Unityn sisäistenkin toimintojen säikeistämisen ja tuottaa rakennusvaiheessa tehokkaampaa ohjelmakoodia. Uuden renderöintiputken käyttöönotto vaatii kuitenkin varjostimien uudelleentoteutuksen ja DOTS-järjestelmä ei ole vielä virallisesti valmis.

7. YHTEENVETO

Unity on hyvä työkalu virtuaalisovellusten toteutukseen. Hyvien käytäntöjen noudattaminen on tärkeää miellyttävän kokemuksen takaamiseksi sekä kehityksen helpottamiseksi. Näiden seuraaminen polygonimalleja toteuttaessa auttaa kehityksen siirtyessä mallinnusohjelmasta Unityyn. Silmiä rasittavia ja pahoinvointia aiheuttavia ongelmia syntyy helposti virtuaalisissa sovelluksissa ja teoria niiden takana ei ole vielä täysin selvillä. Tiheä testaus ja nopea sovelluksen iteraatioaika on tärkeää.

Huonosti toteutettu virtuaalitodellisuus sovellus voi aiheuttaa käyttäjälle pahoinvointia ja silmien rasitusta. Optimointiin tulee panostaa, jos tarkoituksena on julkaista virtuaalisovellus puhelimelle, sillä laitteistovaatimukset ovat korkeat. Huonosti optimoitu sovellus voi kuumentaa puhelinta, kuluttaa akkua tai tehdä sovelluksesta käyttökelvottoman. Paikoittainkin huonosti toimiva sovellus voi aiheuttaa silmien rasitusta.

Unity on erittäin aloittelijaystävällinen ja alkuun pääsee vähällä vaivalla. Unitylle on toteutettu paljon ilmaiseksi jaossa olevia työkaluja ja komponentteja, joita kannattaa käyttää hyödyksi sovellusta kehittäessä. Kehittäjäfoorumit ovat aktiiviset ja dokumentaatio on hyvä ja niitä kannattaa silmäillä aktiivisesti. Unityn laajennettavuus kannattaa ottaa huomioon tuottavuuden maksimoimiseksi. Monet käsin tehdyt usein toistuvat toiminnot voi mahdollisesti automatisoida. Laitetuki on kattava ja kehitys on helppoa ja nopeaa. Profilointityökalut helpottavat tehokkaan sovelluksen toteuttamista paljastamalla prosessoinnin pullonkauloja.

Työn kokeellisessa osassa toteutettiin virtuaalitodellisuussovellus Unityllä sekä Node.js-palvelinympäristö. Sovellusta testattiin Daydream -alustaa tukevalle puhelimella. Sovellus toimi hyvin käyttötarkoitukseensa. Testauksen ja profilointityökalujen perusteella sovellus myös toimi tehokkaasti. Ohjelmakoodin ja 3D-mallien lisäksi projektista syntyi tämä kandidaatintyö, joka toimii pohjana muille virtuaalitodellisuussovelluksien toteuttajille tai mahdollisille projektin jatkajille.

LÄHTEET

[1] GitHub, Google VR, 2019, verkkosivu, saatavissa (viitattu 4.3.2019): https://github.com/googlevr

[2] Alex Dobie, 9.11.2016, Google reveals hardware requirements for Daydream V, Mobile Nations, verkkosivu, saatavissa (viitattu 4.3.2019): https://www.androidcentral.com/google-reveals-hardware-requirements-daydream-vr

[3] Oculus VR Best Practices, Instruction to Best Practices, verkkosivu, saatavissa (viitattu 4.3.2019): https://developer.oculus.com/design/latest/concepts/book-bp/

[4] Oculus VR Best Practices, User Input, verkkosivu, saatavissa (viitattu 4.3.2019): https://developer.oculus.com/design/latest/concepts/bp-userinput/

[5] Oculus VR Best Practices, Locomotion, verkkosivu, saatavissa (viitattu 4.3.2019): https://developer.oculus.com/design/latest/concepts/bp-locomotion/

[6] Oculus VR Best Practices, Rendering, verkkosivu, saatavissa (viitattu 4.3.2019): https://developer.oculus.com/design/latest/concepts/bp-rendering/

[7] Oculus VR Best Practices, Vision, verkkosivu, saatavissa (viitattu 4.3.2019): https://developer.oculus.com/design/latest/concepts/bp-vision/

[8] Google VR Design requirements, päivitetty 14.8.2018, verkkosivu, saatavissa (viitattu 4.3.2019): https://developers.google.com/vr/distribute/daydream/design-requirements

[9] Unity Techonlogies, Managed Plugins, 2019, verkkosivu, saatavissa (viitattu 4.3.2019): https://docs.unity3d.com/Manual/UsingDLL.html

[10] Polycount wiki: Edge padding, päivitetty 10.6.2017, verkkosivu, saatavissa (viitattu 4.3.2019): http://wiki.polycount.com/wiki/Edge_padding

[11] Unity Technologies, Occlusion Culling, 2019, verkkosivu, saatavissa (viitattu 4.3.2019): https://docs.unity3d.com/Manual/OcclusionCulling.html

[12] Unity Technologies, Draw call batching, 2019, verkkosivu, saatavissa (viitattu 4.3.2019): https://docs.unity3d.com/Manual/DrawCallBatching.html

[13] Unity Technologies, Level of Detail, 2019, verkkosivu, saatavissa (viitattu 4.3.2019): https://docs.unity3d.com/Manual/LevelOfDetail.html

[14] Unity Technologies, Optimizing garbage collection in Unity games, 2019, verkkosivu, saatavissa (viitattu 14.4.2019): https://unity3d.com/learn/tutorials/topics/performance-optimization/optimizing-garbage-collection-unity-games

[15] A. Davis, 19.6.2017, Threads, promises and Unity, verkkosivu, saatavissa (viitattu 14.4.2019): http://www.what-could-possibly-go-wrong.com/threads-promises-and-unity

[16] R. Nystrom, 2009–2014, Game Programming Patterns, E-Book, saatavissa (viitattu 4.3.2019): http://gameprogrammingpatterns.com/contents.html

[17] Unity Technologies , Unity Android Profiling Tools, verkkosivu, saatavissa (viitattu 4.3.2019): https://unity3d.com/learn/tutorials/topics/best-practices/android-profiling-tools

[18] Unity Technologies , Unity .NET Profile Support, verkkosivu, saatavissa (viitattu 4.3.2019): https://docs.unity3d.com/Manual//dotnetProfileSupport.html

[19] Oculus, Debugging and Performance Analysis in Unity, verkkosivu, saatavissa (viitattu 4.3.2019): https://developer.oculus.com/documentation/unity/latest/concepts/unity-integration-perf/