TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

ILKKA MÄÄTTÄ

LISÄTYN TODELLISUUDEN SOVELLUS WEBGL-YMPÄRISTÖSSÄ

Diplomityö

Tarkastaja: professori Kari Systä
Tarkastaja ja aihe hyväksytty
8. kesäkuuta 2016

TIIVISTELMÄ

ILKKA MÄÄTTÄ: Lisätyn todellisuuden sovellus WebGL-ympäristössä
Tampereen teknillinen yliopisto
Diplomityö, 43 sivua
Toukokuu 2017
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Ohjelmistotuotanto
Tarkastaja: professori Kari Systä

Avainsanat: WebGL, Lisätty todellisuus, AR

Tämän työn tavoitteena oli toteuttaa prototyyppijärjestelmä automaatiojärjestelmän informaation visualisointiin hyödyntäen lisättyä todellisuutta. Prototyyppijärjestelmältä vaadittiin, että sen käyttöliittymä olisi selainpohjainen, jonka perusteella päädyttiin valitsemaan WebGL-kirjasto lisätyn todellisuuden toteutusta varten. Prototyypin toteuttamista varten oli käytettävissä testauskäyttöön tarkoitettu koekenttä, jonne oli ennalta asennettu automaatiojärjestelmä ja IP-kamerat. Selainympäristöstä johtuen ei haluttu päätellä videokuvasta kameran orientaatiota. Tämän pohjalta päädyttiin kyselemään kameroilta näiden orientaatiotietoa, sekä hyödyntämään ennalta tunnettua tietoa näiden sijainneista. Toteutuksen lisäksi tässä tekstissä esitellään lyhyesti toteutukseen liittyvää teoreettista taustaa.

Toteutettu järjestelmä rakentui asiakas-palvelin mallisesta selainsovelluksesta, jakautuen kahdeksi palvelinsovellukseksi ja asiakkaanpään selainsovellukseksi. Palvelinsovelluksilla muunnettiin automaatiojärjestelmän tietoa ja kameroiden videokuvaa muotoon, joka voitaisiin välittää käyttäjälle. Nämä sovellukset toteutettiin ensisijaisesti hyödyntämällä C#- ja Python-kieliä. Selaimessa vastaanotetuista tiedoista muodostettiin videokuvan päälle lisätyn todellisuuden näkymä, hyödyntämällä Javascript-kieltä sekä WebGL-kirjastoa. Palvelin välitti tietoa käyttäjille reaaliajassa, päivittäen lisätyn todellisuuden näkymää.

Lopullinen prototyyppijärjestelmä tarjosi käyttäjälle selainnäkymän, jossa lisätyn todellisuuden näkymä oli nähtävissä kamerakuvassa. Videokuvasta käyttäjä pystyi valitsemaan kuvassa näkyviä laitteita ja objekteja, jolloin visualisoitiin näiden päälle geometriaa ja tuotiin esiin reaaliaikaisesti päivittyvää tilatietoa. Toteutettu prototyyppi sisälsi joitain epätäydellisyyksiä lisätyn todellisuuden kohdistamisessa videokuvaan. Tämä johtui epätarkkuuksista kameroilta saaduissa orientaatiotiedoissa, sekä näiden suhteesta asennuksesta johtuviin poikkeamiin. Epätäydellisyyksistä huolimatta se täytti siltä vaaditut ominaisuudet ja toiminnallisuudet, ja kykeni visualisoimaan kuvassa näkyviä laitteita ja kohteita. Toteutus jäi prototyyppihenkiseksi, ja jatkokehityksessä sitä ei tulla hyödynnetä sellaisenaan. Se kuitenkin tarjoaa pohjan jatkokehitystä varten.

ABSTRACT

ILKKA MÄÄTTÄ: Application of Augmented reality with WebGL
Tampere University of Technology
Master of Science Thesis, 43 pages
May 2017
Master's Degree Programme in Information Technology
Major: Software Engineering
Examiner: Professor Kari Systä

Keywords: WebGL, AR, Augmented reality

The goal for this thesis was to implement a prototype system for visualizing information of an automation system via augmented reality. The requirements given for the prototype system required user interface to be a browser interface, so it was decided to utilize WebGL library for implementing augmented reality. To implement the prototype system, a test field was available to use with previously installed automation system and IP cameras. Due the browser environment it was preferred to not calculate the camera orientation from the received video. Therefore it was decided to query cameras for their orientation and use the existing knowledge on their locations. Along the implementation, the related theoretical background related to the implementation is also introduced shortly in this text.

The implemented system was structured as client-server model web application, splitting into two pieces of server-side software and a client-side browser application. The server-side software was made to prepare the data from automation system and cameras in a format that could be passed to client. They was implemented mainly using C# and Python languages. Web application at the client side would make use of this data and generate augmented reality view over the video feed, utilizing Javascript language and WebGL library. Information would be passed from server to client in real-time, updating the augmented reality view.

The final implemented prototype provided the user with browser view where augmented reality was visible on top of camera feed. User could select vehicles and objects from the picture, generating visible geometry over these and providing real-time data about them. The implementation had some imperfections in fitting the augmented reality over the video feed. These imperfections were caused by inaccuracies in the queried data of camera orientations compared to way the cameras were installed. Even with these imperfections the implemented system fulfilled the requirements set for it, and it managed to visualize information about the vehicles and targets within the video. The implementation was left as prototype, and will not be utilized as such in further development. It however does provide a starting point for further development.

ALKUSANAT

Haluan osoittaa kiitokseni ohjauksesta ja kärsivällisyydestä Kari Syställe ja Antti Puhakalle. Lisäksi haluan osoittaa kiitokseni diplomityön aiheesta ja sen tekemisen mahdollisuudesta Ville-Pekka Kärjelle, Tomi Äijölle, Vesa Sarviluomalle, Tomi Mikkoselle, Katja Niemiselle ja Rita Fagerströmille. Kiitokset haluan myös osoittaa Markus Sertamolle ohjeistuksesta käytetyn automaatiojärjestelmän kanssa kommunikoimiseen.

Tampereella, 23.5.2017

Ilkka Määttä

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

API	eng. Application programming interface, ohjelmointirajapinta
ARB	Architectural review board
CGI	eng. Common gateway interface. Eräs asiakkaan ja palvelimen välisen tiedonsiirron toteuttamisen tekniikka.
CORS	eng. Cross Origin Resource Sharing
FOV	eng. Field of view, näkökenttä, näkökentän laajuus
GLSL	OpenGL shading language, ohjelmointikieli OpenGL sävyttimien kirjoittamista varten
GPS	eng. Global positioning system, maailmanlaajuinen paikallistamisjärjestelmä
HLS	eng. HTTP live streaming, HTTP suoratoisto. HTTP-protokollan kautta toimiva median suoratoiston protokolla.
IP-kamera	eng. IP-camera, Internet Protocol Camera.
IP-osoite	eng. IP address, Internet protocol address. Internetin protokollaosoite, jolla IP-verkkoon liitetyt laitteet voidaan yksilöidä.
MIT-lisenssi	MIT licence, Massachusetts Institute of Technology licence, eräs yleisesti käytössä oleva vapaa ohjelmistolisenssi
OpenGL ES	Open graphics library for embedded systems
OpenGL	Open graphics library
OWIN	Open Web Interface for .NET
RTSP	eng. Real-time streaming protocol. Protokolla median suoratoiston toteuttamiseksi IP-verkoissa.
SGI	Silicon Graphics, Inc.
URI	eng. Unique resource identifier. Merkkijono, jolla ilmaistaan verkkoresurssin sijaintia.
WebGL	Web graphics library

Käytettyjä käännöksiä

Edustapalvelin	eng. Reverse proxy. välipalvelin, joka hakee resursseja tai suorittaa toimintoja käyttäjän puolesta, piilottaen näiden lähteet käyttäjältä.
Koodinvaihto	eng. Transcoding. Videokuvan muuttaminen suoraan formaatista toiseen.
Sävytin	eng. Shader. käyttäjän määrittelemä osa OpenGL liukuhihnan toiminnallisuutta.
Sävytinohjelma	eng. Shader program. sävyttimen ohjelmakoodi

MATEMAATTISET MERKINNÄT

a	Kvaternion reaaliosa
b	Kvaternion imaginaariosa imaginaariyksikölle i
c	Kvaternion imaginaariosa imaginaariyksikölle j
$\boldsymbol{C}$	Koordinaatistomuunnoksen muunnosmatriisi
cos	Kosinifunktio
d	Kvaternion imaginaariosa imaginaariyksikölle k
$\overline{e_1}$, $\overline{e_2}$ ja $\overline{e_3}$	Koordinaatistomuunnoksen kohdekoordinaatiston virittävät vektorit
e_{ab}	Vektorin $\overline{e_a}$ alkio b
f	Etäisyys takaleikkaustasosta
i	Kvaternion enimmäisen imaginääriosan imaginaariyksikkö
j	Kvaternion toisen imaginääriosan imaginaariyksikkö
k	Kvaternion kolmannen imaginääriosan imaginaariyksikkö
$\boldsymbol{M}$	Muunnosmatriisi
n	Etäisyys etuleikkaustasosta
$\boldsymbol{P}$	Perspektiivimuunnoksen muunnosmatriisi
$\boldsymbol{q}$	Kvaternio
$\boldsymbol{q}^*$	Kvaternion konjugaatti
$\boldsymbol{q_r}$	Kiertoa kuvaava kvaternio
$\boldsymbol{r}$	Kvaternio
$\boldsymbol{R}$	Kiertomuunnoksen muunnosmatriisi
$\boldsymbol{R_x}$	Kiertomuunnoksen muunnosmatriisi X-akselin ympäri
$\boldsymbol{R_y}$	Kiertomuunnoksen muunnosmatriisi Y-akselin ympäri
$\boldsymbol{R_z}$	Kiertomuunnoksen muunnosmatriisi Z-akselin ympäri
$\boldsymbol{R_{q_r}}$	Kvaterniosta johdettu kiertomuunnoksen muunnosmatriisi
$\mathbb{R}^2$	Kaksiulotteinen reaaliavaruus
$\mathbb{R}^3$	Kolmiulotteinen reaaliavaruus
$\mathbb{R}^4$	Neliulotteinen reaaliavaruus
s_x	Skaalaus X-akselin suuntaisesti
s_y	Skaalaus Y-akselin suuntaisesti
s_z	Skaalaus Z-akselin suuntaisesti
$\boldsymbol{S}$	Skaalauksen muunnosmatriisi
$\boldsymbol{s}$	Kokonaismuunnoksen kvaternio
$\boldsymbol{s}^*$	Kokonaismuunnoksen kvaternion konjugaatti
sin	Sinifunktio
t_x	Siirtymä X-akselin suuntaisesti
t_y	Siirtymä Y-akselin suuntaisesti
t_z	Siirtymä Z-akselin suuntaisesti
$\boldsymbol{T}$	Siirtomuunnoksen muunnosmatriisi
$\boldsymbol{T_1}$	Siirtomuunnoksen muunnosmatriisi siirtymästä origokeskeiseksi
$\boldsymbol{T_2}$	Siirtomuunnoksen muunnosmatriisi siirtymästä takaisin alkuperäiseen paikkaansa
$\bar{u}$	Eräs yksikkövektori
u_x	Yksikkövektorin X-koordinaatti
u_y	Yksikkövektorin Y-koordinaatti
u_z	Yksikkövektorin Z-koordinaatti
$\bar{v}$	Vektori

$\boldsymbol{v}$	3D-avaruuden vektorin esitys kvaterniona
$\overline{v_m}$	Muunnettu vektori
$\boldsymbol{v_m}$	3D-avaruuden vektorin esitys kvaterniona muunnoksien jälkeen
$\overline{v_{2D}}$	Kaksiulotteinen vektori
$\overline{v_{h2D}}$	Kaksiulotteinen vektori homogeenisessä koordinaatistossa
$\overline{v_{3D}}$	Kolmiulotteinen vektori
$\overline{v_{h3D}}$	Kolmiulotteinen vektori homogeenisessä koordinaatistossa
w	W-koordinaatti
x	X-koordinaatti
y	Y-koordinaatti
z	Z-koordinaatti
$\cong$	Vastaavuus
θ	Kierron kulma

1. JOHDANTO

Virtuaalitodellisuus ja lisätty todellisuus ovet olleet paitsi tieteisfiktion unelmia, myös laajemman tutkimisen ja ohjelmistokehityksen kohteita. Viimeaikainen teknologian kehittyminen on edesauttanut näiden yleistymistä, ja viime vuosien saatossa näihin perustavia laitteita ja sovelluksia on ilmestynyt markkinoille. Vaikka suurin osa onkin viihdekäyttöön suunnattuja, myös esimerkiksi teollisuudessa ja lääketieteen alalla näiden sovellutuksia on tehty.

Tämän työn tavoitteena oli toteuttaa prototyyppijärjestelmä lisätystä todellisuudesta automaatiojärjestelmän tilan visualisointia varten. Vaatimuksena oli toteuttaa se selainympäristössä, jolloin päädyttiin hyödyntämään WebGL (Web graphics library) -ohjelmistokehystä lisätyn todellisuuden graafisen osuuden toteuttamisessa. Käytettävissä oli reaaliaikaista videokuvaa testausympäristöstä, johon automaatiojärjestelmää haluttiin visualisoida tuottamalla lisätyllä todellisuudella lisäinformaatiota videokuvaan. Prototyyppi rakennettiin toteuttamalla palvelinsovelluksia ja varsinainen selainsovellus. Palvelinsovellukset esikäsittelivät tarpeellisen datan, jonka perusteella selainsovelluksessa muodostettiin lisätyn todellisuuden näkymä.

Luvussa 2 esitellään tämän työn kannalta merkittävät motivaatiot ja lähtökohdat työn tekemiseksi, sekä annettu joitain tiedettyjä rajoitteita. Luvut 3 ja 4 esittelevät työhön liittyvää teoreettista taustaa lisätystä todellisuudesta ja 3D-grafiikkaan liittyvän matematiikan perusteista. Luku 5 esittelee lyhyesti WebGL-kirjaston perusrakenteen ja toiminnan, sekä lyhyesti esittelee tätä varten tehtyjä ohjelmistokehyksiä.

Luvussa 6 käydään läpi tämän työn ohessa tehty kokonaisjärjestelmä, tähän liittyvät yksittäiset sovellukset ja näiden yksityiskohtia. Luvussa 7 pohditaan lopputulosta mikä työssä saatiin aikaiseksi. Lopuksi luvussa 8 tiivistetään toteutukseen ja työhön liittyvä keskeinen tieto.

2. MOTIVAATIOT JA LÄHTÖKOHTA

Teollisuudessa käytetään yhä useammin automaatiojärjestelmiä, joissa erilaisilla laitteilla liikutellaan materiaaleja ja tarvikkeita. Tällainen voi olla esimerkiksi automaatiojärjestelmä, jossa nostureilla ja konttilukeilla siirretään kontteja paikasta toiseen tietokoneiden ohjaamana. Tavaran liikuttamiseen ja kokonaisjärjestelmien ylläpitoon ja seurantaan on olemassa laitespesifisiä järjestelmiä, joilla pyritään varmistamaan ja ylläpitämään toimintaa.

Koska laitteet toimivat reaalimaailmassa, automaatiojärjestelmissä voi muodostua erilaisia ongelmatilanteita, esimerkiksi laitteet voivat vikaantua. Vaikka järjestelmästä olisikin olemassa erillisiä visualisointityökaluja, voi ongelmatilanteiden havainnointi ja selvitys aiheuttaa ongelmia, kun joudutaan vertaamaan loki- ja tilatietoja, visualisaatiotyökalujen näkymiä ja reaalimaailman laitteita keskenään. Vaikka ongelmatilanne havaittaisiin visuaalisesti tai valvontakameran kuvasta, voi joissakin tilanteissa kyseisen ongelman lähteiden tunnistaminen järjestelmästä olla hankalaa.

Tämä visualisoinnin, loki- ja tilatietojen sekä visuaalisen havainnoinnin yhdistämisen ongelma muodostivat motivaation tutkia lisättyyn todellisuuteen pohjaavaa visualisointityökalua paikkaaman kyseistä ongelmaa. Tällä tavalla esimerkiksi vikatilanteista olisi mahdollista saada kameran videokuvan päälle lisättyä informaatiota, jonka avulla pystyttäisiin päättelemään mistä kyseisessä vikatilanteessa olisi kyse.

Lähtökohtana tässä työssä tehdylle projektille oli tuottaa ”proof-of-concept” prototyyppijärjestelmä, joka lisäisi lisätyn todellisuuden tapaisesti informaatiota laitteista ja kohteista videokuvan päälle. Kuvassa näkyvien koneiden ja materiaalien päälle muodostettaisiin geometriaa, joka noudattelisi reaaliajassa näiden sijaintia videokuvassa. Lisäksi kohteista tulisi myös nähdä lisäinformaatiota tarvittaessa.

Kuva 2.1. *3D-mallinnus geneerisestä IP-kamerasta, joka on saman tapainen kuin työssä käytössä olleet kamerat.*

Toteutuksen testausta ja kehittämistä varten oli käytettävissä testauskenttä. Tänne oli asennettu Kuvan 2.1 kaltaisia IP-kameroita (Internet protocol camera) ja automaatiojärjestelmä, mistä saatiin informaatiota siellä liikkuvista laitteista. IP-kameroita oli mahdollista ohjata etäohjauksella selaimen kautta, ja yksittäiseltä kameralta saatiin HTTP-kyselyillä tietoa sen tilatiedoista.

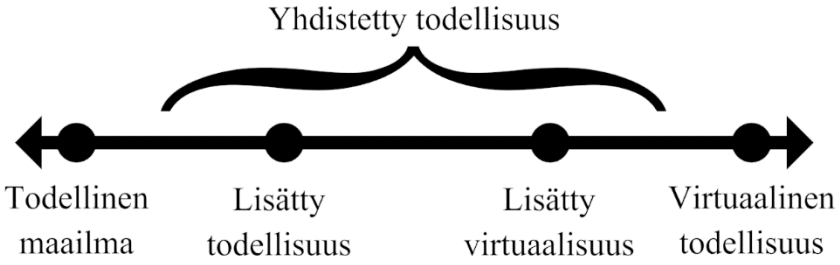
Testikentän kameroita ei oltu asennettu varta vasten tätä projektia varten, vaan asiakkaan omaa tuotekehitystä ja testaamista varten. Tämän vuoksi projektissa jouduttiin sopeutumaan kameroiden erilaisen käytön tuomiin ongelmiin, esimerkiksi tarkkojen sijaintitietojen puuttumiseen.

Prototyypiltä haluttiin, että sen käyttöliittymä olisi selainpohjainen, koska nykyiset työkalut ovat osittain selainpohjaisia. Selainpohjaisuuden avulla olisi myöhemmin mahdollista liittää toteutettu osuus osaksi nykyistä ympäristöä. Tämän vuoksi haluttiin välttää ylimääräinen raskas laskenta, ja siten kuvasta tunnistamisen sijasta haluttiin hyödyntää ennalta tunnettua tietoja ja arvioita ympäristöstä, sekä tukeutua kameroista saatavaan tietoon niiden ja orientaatiosta.

3. LISÄTTY TODELLISUUS

3.1 Määritelmä

Milgram et al. määrittelivät lisätyn todellisuuden esittelemällä niin kutsutun *todellisuus-virtuaalisuus jatkumon*. Tämän jatkumon ääripäinä ovat puhtaasti virtuaalinen todellisuus ja varsinainen todellisuus, ja näiden väliin kuuluvat *lisätty virtuaalisuus* (*augmented virtuality*) sekä itse lisätty todellisuus (*augmented reality*). Jatkumossa lisätty todellisuus nähtiin reaalimaailman täydentämisenä virtuaalisilla kappaleilla tai informaatiolla, ja lisätty virtuaalisuus vastaavasti virtuaalitodellisuutena, jota on laajennettu reaalimaailman kappaleilla. Koska eri järjestelmät toteuttavat eri tasolla todellisuuden ja virtuaalitodellisuuden yhdistämistä, Milgram et al. määrittelivät todellisuuden ja virtuaalitodellisuuden välisen alueen termillä *Yhdistetty todellisuus* (*Mixed reality*). Kuvassa 3.1 on yksinkertaistettu esitys Milgram et al. määrittelemästä jatkumosta. [1, p.2-4]



Kuva 3.1. *Yksinkertaistettu esitys todellisuus-virtuaalisuus jatkumosta. Pohjaa Milgram et al. tekstissä käytettyyn kuvaan. [1, p.3]*

Azuma määritteli lisätyn todellisuuden järjestelminä, jotka täyttävät seuraavat kolme kriteeriä: Niiden tulisi olla interaktiivisia järjestelmiä, jotka yhdistävät todellisuutta ja virtuaalisuutta ja ovat kolmiulotteisia järjestelmiä. Tällä määritelmällä haluttiin ilmaista se, että lisätty todellisuus ei ole sidottu mihinkään tietyn tyyppiseen näyttöön tai vaikkapa laseihin. Lisäksi määritelmällä pyrittiin erottamaan epäinteraktiiviset mediat lisätyn todellisuuden määritelmästä, esimerkkinä näistä tietokonegrafiikalla ehostetut elokuvat. [2, p. 2-3]

Milgram et al. ja Azuman määritelmät lisätystä todellisuudesta käyttävät esimerkkeinä järjestelmiä, joissa reaaliaikaiseen videokuvaan lisätään tietokoneella tuotettua grafiikkaa kuvastamaan virtuaalitodellisuuden kappaleita. Vaikka tämä on nykyisesti yleisin tulkintatapa lisätyn todellisuuden käytöstä, on myös ehdotettu lisätyn todellisuuden laajentuvan myös muiden aistien ehostamiseen. Esitettyjä tapoja ovat esimerkiksi tuntoaistillinen (haptinen) tai kuuloaistillinen lisätty todellisuus, jolloin virtuaalista todellisuutta voitaisiin havaita kosketuksella tai äänillä. [1, p. 6][2, p. 9]

3.2 Motivaatio ja käyttökohteet

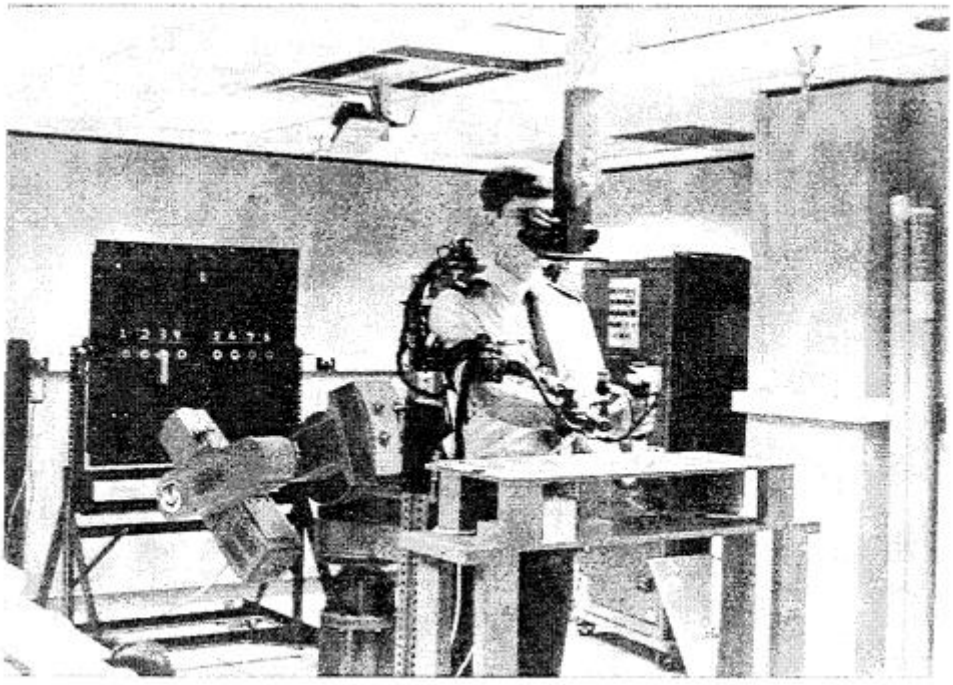
Lisätyn todellisuuden hyödyntämisellä pyritään tyypillisesti edistämään käyttäjän tietoisuutta ehostamalla ympäristöstä havaittavaa informaatiota. Tätä voidaan tehdä lisäämällä ja poistamalla kappaleita käyttäjän näköalueelta. Tällöin käyttäjä kykenee havaitsemaan ympäristöstään asioita, joita hän ei välttämättä voisi tavallisesti havaita. Toinen motivoiva piirre on lisätyn todellisuuden tuoma lisätarkkuus työntekijöille esimerkiksi teollisuuden alalla rakentamisessa. [3, p. 3-4]

Käyttökohteita lisätylle todellisuudelle on tunnistettu useita, ja teknologian kehittyessä lisätyn todellisuuden hyödyntämistä on tuotu eri käyttökohteisiin eri tavoin. Selkeitä käyttökohteita ovat lääketiede, teollisuus ja viihdeteollisuus. [3, p. 24-38][2, p. 3-9]

Lääketieteen alalla lisättyä todellisuutta hyödynnetään eri tavoin, tukena hoitamisessa tai ihmisten kouluttamisessa lääketieteen alalla. Hyödyntämisen eri tapoja on tutkittu ja kokeiltu, joista esimerkkeinä ovat laparoskopiset leikkaukset [4, p. 1] ja tietyn tyyppisten fobioiden altistushoidot [5, p. 1]. Lisäksi lääketieteen opetuksessa lisätyllä todellisuudella voidaan visualisoida anatomiaa ja yksityiskohtia [6, p. 1].

Lisättyä todellisuutta voidaan myös hyödyntää teollisuuden alalla työntekijöiden kouluttamisessa, mutta myös tuotantolinjoilla ja korjausoperaatioissa löytyy potentiaalisia hyödyntämiskohteita. Tuotantolinjoilla voidaan rakennettujen kappaleiden päälle sovittaa kappaleen 3D-malli, jolloin poikkeamat ja virheet voidaan havaita ja tarkistaa nopeasti [7, p. 1]. Azuma myös kuvailee mahdollisuutta helpottaa työntekijöiden opastusta, kun manuaalit ja dokumentaatio voidaan osin korvata visuaalisella tiedolla ja opastuksella [2, p. 5].

Louis Rosenberg rakennutti vuonna 1992 lisätyn todellisuuden järjestelmän, jossa etänä ohjattiin robottikäsiä suorittamaan yksinkertaista operointia. Tämän avulla Rosenberg kykeni havainnollistamaan, että lisätyn todellisuuden avustamana laitteen operaattorin suorituskyky parani huomattavasti. Kuvassa 3.2 on näkyvissä Rosenbergin rakennuttama järjestelmä. [8, p. 16-17, p. 39]



Kuva 3.2. *Kuva Louis Rosenbergin rakentamasta lisätyn todellisuuden järjestelmästä.[8, p. 12]*

Viihdeteollisuus on selkein hyödyntämisen kohde, erityisesti interaktiivisen median osalta. Viime vuosien aikana on tullut markkinoille useita lisättyä todellisuutta hyödyntäviä interaktiivisia pelejä ja sovelluksia [3, p. 30-32]. Lisäksi lisättyä todellisuutta hyödynnetään kaupallisesti, mm. erilaisilla virtuaalisilla pukukopeilla, joissa asiakkaat voivat sovittaa virtuaalisia versioita vaatteista päälleen [3, p. 25-28]. Huvipuistoissa lisättyä ja virtuaalista todellisuutta on hyödynnetty erilaisten elämyksien tai virtuaalisten vuoristoratojen muodossa [9, p. 7].

Lisättyä todellisuutta hyödynnetään myös taiteiden ja sisustussuunnittelun osalta. Taiteilijat voivat suunnitella esimerkiksi maalaamisen tai veistämisen seuraavia askelia lisätyn todellisuuden avulla, ja suunnittelijat voivat koestaa huonekaluja ja sisustusta ennen näiden varsinaista hankkimista. Vastaavasti voidaan suunnitella rakennuksien ulkonäköä ja sopimista maisemaan hyödyntämällä lisättyä todellisuutta. [9, p. 8][10, p. 23-24]

3.3 Historiasta

Ensimmäisiä suoria lisätyn todellisuuden konsepteja löytyy vuodelta 1901, L. Frank Baumin novellista *”The Master Key”*. Tässä novellissa kuvaillaan silmälasit, jotka projisoivat henkilöiden otsalle kirjaimen kuvastamaan tämän persoonallisuutta. Vuosina

1955-1962 Morton Heilig suunnitteli ja rakennutti prototyypin järjestelmästä, jolla yksittäinen katsoja voisi kokea elokuvia eri aistiensa kautta. Tämän järjestelmän nimeksi hän antoi Sensorama. Kuvassa 3.3 on kuva kyseisestä järjestelmästä. [3, p. 4] [11]

Kuva 3.3. *Heiligin suunnittelema Sensorama [12]*

Vuonna 1968 Ivan Sutherland kokosi ensimmäisen lisätyn todellisuuden järjestelmän, käyttämällä kypäränäyttöä. Yleensä ensimmäiseksi käyttökerraksi termille ”augmented reality” katsotaan olevan Tom Claudell ja David Mizzel vuodelta 1990. [3, p. 4-5]

Louis Rosenberg rakensi ensimmäisen varsinaisen lisätyn todellisuuden järjestelmän vuonna 1992. Tällä järjestelmällä operaattori pystyi etänä suorittamaan robottikädellä yksinkertaisia operaatioita, ja lisätyllä todellisuudella käyttäjä sai avustavaa informaatiota ympäristöstä. Järjestelmällä pyrittiin tutkimaan lisätyn todellisuuden tuomia hyötyjä mm. suorituskyvyn ja tarkkuuden tehostamiseen hyödyntämällä lisätyn todellisuuden tietoja. [3, p. 4-5][8, p. 18]

Samoihin aikoihin Rosenbergin järjestelmän kanssa Feiner et al. suunnittelivat ja esittelivät järjestelmän nimeltä KARMA (Knowledge-based Augmented reality for Maintenance Assistance), jota pidetään yhtenä ensimmäisistä merkittävimmistä julkaisuista lisätyn todellisuuden osalta. KARMA-järjestelmällä pyrittiin tutkimaan mahdollisuuksia visualisoida laitteistojen korjaamisessa ja ylläpidossa tarvittavia tietoja, kuten visualisoida laitteiston sisäosia tai korostaa tarvittavia toimenpiteitä. [13, p. 54-57][3, p. 4]

Milgram et al. määrittelivät todellisuus-virtuaalisuus jatkumon vuonna 1994, ja 1997 Azuma laajensi ja yksityiskohtaisti lisätyn todellisuuden määritelmää. Azuman määritelmää pidetään nykyäänkin määritelmänä lisätylle todellisuudelle. [3, p. 4-5]

3.4 Toteutustapoja ja yleisiä tekniikoita

Lisätyn todellisuuden järjestelmiä on erilaisia eri käyttötarkoituksia varten. Azuma lajittelee tällaisten järjestelmien lähtökohdiksi kaksi vaihtoehtoa: optisen ja videoteknologisen lähestymistavan. [2, p. 10]

Optisessa lähestymistavassa lisätty todellisuus heijastetaan läpikuultavan pinnan kautta katsojan silmiin, jolloin pinnan lävitse nähtävä todellisuus ja heijastettu virtuaalinen todellisuus yhdistyvät. Tyypillisesti optista lähestymistapaa on toteutettu lentäjien kypäränäytöissä tai älylaseissa. Eräitä viimeaikaisia esimerkkejä optisesta lähestymistavasta ovat Microsoftin Hololens-älylasit ja Google Glass-älylasit. [2, p. 10]

Videoteknisessä ratkaisussa videokuvaa käsitellään reaaliaikaisesti, tuottamalla sitä vasten tietokonegrafiikkaa. Lopulta käsitelty videokuva toistetaan käyttäjälle näytön kautta. Tällaisissa järjestelmissä päähän puettavissa versioissa näytön tausta ei ole läpinäkyvä, toisin kuin vastaavissa optisen lähestymistavan laitteissa. Toisaalta videotekninen lähestymistapa sallii myös, että videokuva näkyy vaikkapa tietokoneen näytön kautta, eikä siten ole sidottu suoraan käyttäjän katseen tai pään orientaatioon, vaan vaikkapa etäkameran orientaatioon. [2, p. 11-13]

Molemmissa lähestymistavoissa on hyötyjä ja haittoja toisiinsa nähden, ja ne sopivat eri tilanteisiin eri tavoin. Optisen järjestelmän etuja ovat mm. yksinkertaisuus prosessoinnin tarpeessa, koska vain virtuaalitodellisuus tarvitsee tuottaa katsojalle. Vastaavasti videoteknisissä järjestelmissä etuja ovat mm. monimuotoisemmat vaihtoehdot videon käsittelyssä, kun alkuperäistä kuvaa käsitellään suoraan. Videoteknisempi vaihtoehto tarjoaa myös potentiaalista (mutta hankalasti toteuttavaa) ajatusta tiedon karsimisesta ja reaalimaailman kappaleiden ”poistamisesta”. [2, p.13-15]

Tekniset lähtökohdat näytön osalta ovat vain yksi aspekti lisätyn todellisuuden järjestelmää, ja näiden lisäksi tarvitaan myös tapa kohdistaa reaalimaailma ja virtuaalitodellisuus toisiinsa. Näitä varten on useita tapoja, joita ovat mm. sensoripohjainen kohdistus ja konenäköön pohjaava kohdistus. [10, p. 92]

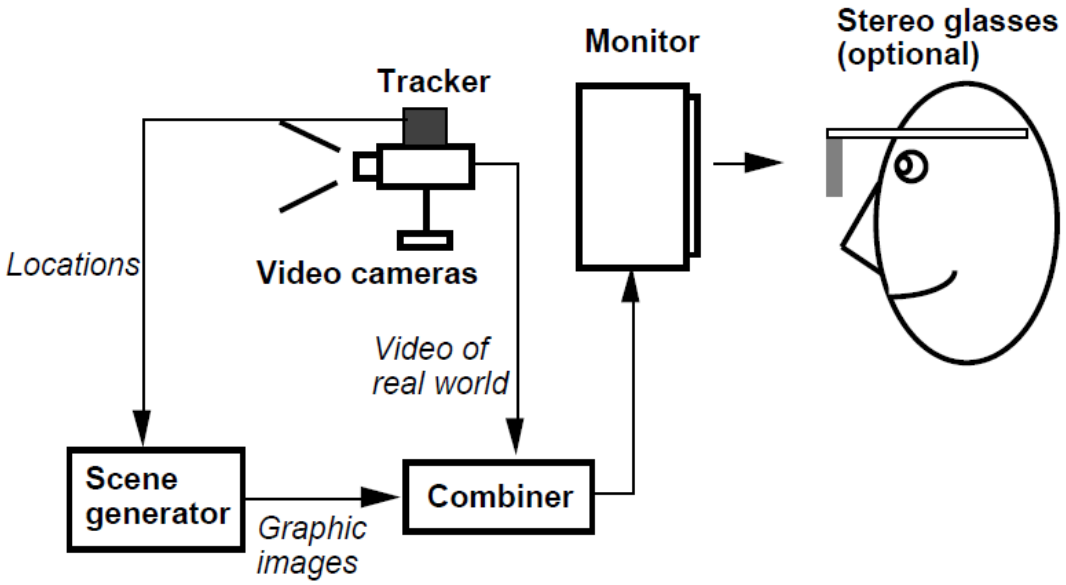
Sensoripohjaisessa kohdistamisessa hyödynnetään laitteiston sensoreita paikannuksen ja orientaation tunnistamiseen, ja näistä tiedoista pyritään päättelemään virtuaalisen tiedon sijainti katsojan näkökulmasta. Tätä lähestymistapaa käytetään paljon mobiililaitteilla, joissa nykyään käytetään esimerkiksi kiihtyvyysantureita ja GPS-paikannusta (Glopal positioning system). Näillä tiedoilla voidaan yrittää päätellä kameran orientaatio ja sijainti, jolloin 3D-data voidaan asemoida kuvaan oikeellisesti. [9, p. 12] [10, p. 92]

Konenäön periaatteita hyödyntämällä voidaan havainnoida ennalta määriteltyjä kappaleita, ja niiden kameran havainnoiman muodon ja sijainnin perusteella päätellä kameran sijaintia ja orientaatiota. Ennalta määritellyt kappaleet voivat olla objekteja kuten esimerkiksi kirjat kirjahyllyssä, tai vaihtoehtoisesti ns. markkereita, joita on piirretty ympäristöön tai vaikkapa paperille. Vaihtoehtoisesti voidaan määritellä ympäristöstä maamerkkejä, joita pyritään havaitsemaan kuvasta ja päättelemään niiden pohjalta orientaatiota. [10, p. 38-39, p. 92-93]

Vaikka nämä ja muut kohdistustavat yksittäisinä voivat tarjota toimivan kohdistamisen, hyvin yleisesti yhdistellään eri kohdistustapoja. Esimerkkinä voisi olla sensoreiden informaation hyödyntäminen yhdistämistä markkeripohjaisen kohdistuksen kanssa. Yhdistelemisen motivaationa on tyypillisesti paikata eri kohdistustapojen omia heikkouksia tai täydentää tietoisuutta ympäristöstä, jolloin saadaan tarkempi lopputulos kohdistuksessa. [10, p. 92]

3.5 Lisätty todellisuus tämän työn puitteissa

Tässä tekstissä kuvatussa työssä lisätty todellisuus noudattelee aliluvussa 3.4 esiteltyä videoteknistä ratkaisua, joka noudattelee etäisesti Azuman esittämää diagrammia monitoripohjaisesta lisätystä todellisuudesta [2, p. 13], jota on esitetty kuvassa 3.4. Vaikka kuvassa esitetäänkin vaihtoehtoisena osana järjestelmää erikoislasit, tässä työssä näitä ei hyödynnetty.



Kuva 3.4. *Konseptidiagrammi monitoripohjaisesta lisätystä todellisuudesta [2, p.13]*

Käytössä olleista kameroista saadaan sensoritietoina yksittäisen kameran asento, ja kameran sijainnin ollessa staattinen näistä tiedoista muodostetaan virtuaalisen todellisuuden orientaatiot. Tästä tiedosta siten yhdistellään videokuvan päälle virtuaaliset kappaleet, ja muodostettu kuva välitetään näytöllä käyttäjälle.

Tehty järjestelmä hyödyntää enimmäkseen sensoritietoja kohdistamisessa, joskin kameran manuaalisessa kalibroinnissa hyödynnettiin tunnettua visuaalista tietoa ympäristöstä. Valittu ympäristö asetti omia rajoitteitansa, joiden valossa päädyttiin olla hyödyntämättä konenäköä kohdistamisessa.

4. MATEMAATTISTA TAUSTAA

Tietokonegrafiikan luomisessa keskeisessä asemassa ovat matemaattiset operaatiot, joiden avulla geometriasta rakennetaan lopullinen kuva. Tässä esitellään lyhyesti tietokonegrafiikkaan liittyvää matemaattista taustaa ja keskeisimpiä asioita, joita tässäkin työssä hyödynnettiin. Näitä ovat geometriset muunnokset ja niiden yhdisteleminen, kvaterniot ja perspektiiviprojektiot sekä näiden käyttöä työssä lyhyesti.

4.1 Homogeeniset koordinaatit

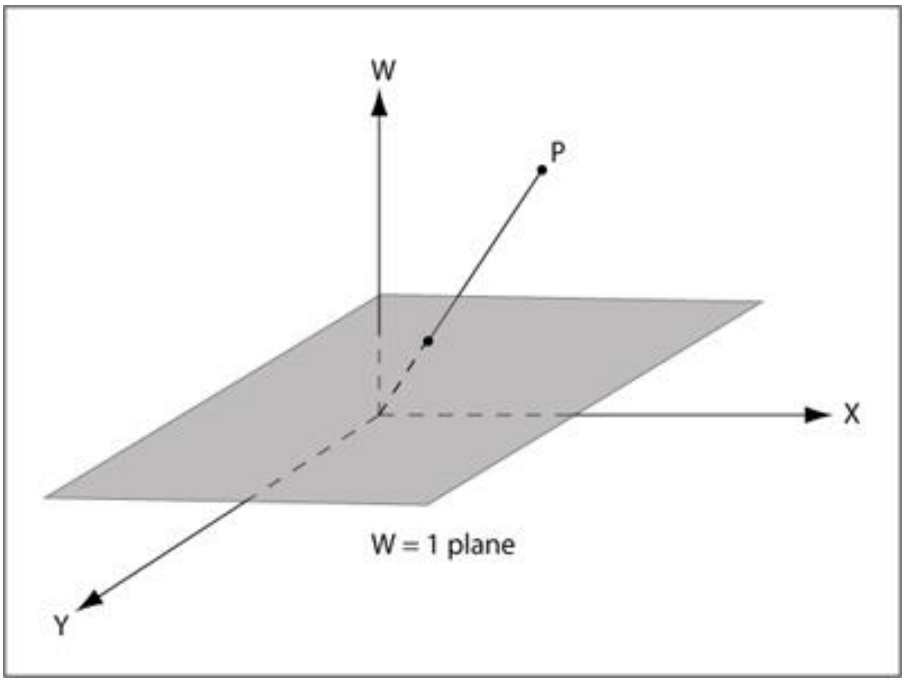
Keskeisimpiä käytettäviä matemaattisia käsitteitä grafiikan tuottamisessa ovat pisteet avaruudessa ja näistä muodostettavat geometriset muodot. Avaruuden pisteet tyypillisesti määritellään kaksi- tai kolmiulotteisessa koordinaatistossa, jossa on määritelty origo. Näin ollen niitä voidaan myös esittää vektoreina ja siksi yleisesti näitä pisteitä kutsutaankin kontekstista riippuen sanoilla vektori tai *verteksi* (kts. Luku 5). [14, s. 33]

Geometriset muunnokset vektoreille esitetään tyypillisesti matriiseina. Matriisien käyttäminen muunnoksien esittämiseen on yleistä, sillä se sallii muunnoksien yhdistämisen pelkällä matriisien kertolaskulla. Tällöin saman kokonaismuunnoksen tekeminen usealle vektorille onnistuu vähemmällä laskennalla. [14, s.107]

Vaikka tyypillisesti vektoreita kuvaillaan koordinaatteina avaruudessa $\mathbb{R}^2$ (2D-grafiikassa) tai $\mathbb{R}^3$ (3D-grafiikassa), kaikki geometriset muunnokset näissä avaruuksissa eivät kuitenkaan ole esitettävissä matriiseina. Tätä ongelmaa voidaan kiertää käsittelemällä vektoreita niin kutsuttuina homogeenisina koordinaatteina. Homogeenisissa koordinaateissa otetaan käyttöön yksi lisäulottuvuus, jolloin alkuperäisen avaruuden tarvittavat muunnokset voidaan esittää matriiseina korkeammassa ulottuvuudessa. Lisäulottuuvuden kautta vektorit homogeenisessä koordinaatistossa ovat kaavan (1) esitysmuotoa, jossa *x*, *y* ja *z* kuvastavat alkuperäisten ulottuvuuksien koordinaatteja, ja *w* on lisätty ulottuvuus. [14, s. 96-100]

$$\overrightarrow{v_{2D}} = \begin{bmatrix} x \\ y \\ w \end{bmatrix}, \ \overrightarrow{v_{3D}} = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \tag{1}$$

Homogeenisia koordinaatteja voidaan tulkita siten, että kertomalla vektori skalaarilla saadaan lopputuloksena saman pisteen esitys. Kahdessa ulottuvuudessa tämä voidaan tulkita siten, että kaikki avaruuden pisteet ovat kolmannen ulottuvuuden suoria, ja kaikki saman w-koordinaatin pisteet sijaitsisivat samalla tasolla. Tätä on havainnollistettu Kuvassa 4.1. [14, s. 96-100]



Kuva 4.1. *Kaksiulotteisen pisteen havainnollistus homogeenisen koordinaatiston suorana.[15]*

Koska 2D-grafiikassa tyypillisesti geometriaa käsitellään samalla tasolla, homogeenisen koordinaatin muodostaminen voidaan toteuttaa määrittelemällä w-koordinaatiksi lukuarvon 1. Vastaavasti homogeenisen koordinaatiston vektorit voidaan projisoida samalle tasolle jakamalla se skalaarilla, joka on yhtä suuri kuin *w*-koordinaatti. Kaavat (2) ja (3) havainnollistavat tätä yhteyttä. [14, s. 97-98]

$$\overline{v_{2D}} = \begin{bmatrix} x \\ y \end{bmatrix} \cong \overline{v_{h2D}} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2}$$

$$\overline{v_{h2D}} = \begin{bmatrix} x \\ y \\ w \end{bmatrix} \cong \begin{bmatrix} x/w \\ y/w \\ 1 \end{bmatrix} \cong \overline{v_{2D}} = \begin{bmatrix} x/w \\ y/w \end{bmatrix} \tag{3}$$

Homogeenisten koordinaattien tulkintaa voidaan laajentaa myös korkeammille ulottuvuuksille, eli kolmannessa ulottuvuudessa vastaavasti kaikki vektorit projisoituvat samalle neljännen ulottuvuuden hypertasolle. [14, s. 99-100]

4.2 Kolmiulotteiset perusmuunnokset

Keskeisiä osia kolmiulotteisen grafiikan tuottamisessa ovat vektorien geometriset muunnokset sekä näiden yhdisteleminen. Kolmannen ulottuvuuden muunnoksia on olemassa useita, mutta tämän tekstin puitteissa käydään kierto, siirto, skaalaus, koordinaatistomuunnos ja perspektiivimuunnos. Edellä mainittujen lisäksi on olemassa mm. vääntömuunnos. [14, s. 104]

Kuten aliluvussa 4.1 on todettu, homogeenisen koordinaatiston kautta geometriset muunnokset voidaan esittää matriiseina. Koska 3D-grafiikan tilanteessa homogeenisessä koordinaatistossa käytetään vektoreita avaruudessa $\mathbb{R}^4$, myös muunnosten esitykset matriiseina ovat tyypillisesti 4x4-matriiseja. Geometrinen muunnos vektorille tapahtuu kertomalla vektori muunnosmatriisilla. Esimerkiksi vektorin $\bar{v}$ muunnos matriisilla $\boldsymbol{M}$ on muotoa

$$\overline{v_m} = \boldsymbol{M}\bar{v}\,. \tag{4}$$

Siirtomuunnos 3D-avaruudessa tapahtuu summaamalla koordinaatteihin siirtymät kyseisen koordinaatiston pääakselien suuntaisesti, mutta pitämällä homogeeninen parametri entisellään. Siirtomuunnoksen matriisi $\boldsymbol{T}$ ja vektorin $\overline{v_{h3D}}$ muunnos matriisikertolaskuna on esitelty kaavoissa (5) ja (6), jossa t_x, t_y ja t_z ovat siirtymiä akselin suuntaisesti.[14, s. 103]

$$\boldsymbol{T} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

$$\boldsymbol{T}\,\overline{v_{h3D}} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} x + t_x w \\ y + t_y w \\ z + t_z w \\ w \end{bmatrix} \cong \begin{bmatrix} x/w + t_x \\ y/w + t_y \\ z/w + t_z \\ 1 \end{bmatrix} \tag{6}$$

Vastaavasti skaalaaminen tapahtuu kertomalla jokainen koordinaatti skalaarilla. Määrittely on esitelty kaavassa (7), jossa s_x, s_y ja s_z ovat skaalauskertoimet akselin suuntaisesti. [14, s. 103]

$$\boldsymbol{S} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7}$$

Koordinaatistomuunnoksessa vektori muunnetaan eri koordinaatiston esitysmuotoon. Tätä varten määritellään kolme ortonormaalia yksikkövektoria $\overline{e_1}$, $\overline{e_2}$ ja $\overline{e_3}$ jotka virittävät avaruuden $\mathbb{R}^3$ ja siten määrittelevät koordinaatiston. Pisteen sijainti muunnetaan vastaavaan sijaintiin kohdekoordinaatistossa. Tämän määrittely on esitetty kaavassa (8). Koordinaatistomuunnoksen käyttökohteita ovat esimerkiksi geometrian siirtäminen katsojan koordinaatiston esitysmuotoon. [14, s. 94, s. 104]

$$\overline{e_1} = \begin{bmatrix} e_{11} \\ e_{12} \\ e_{13} \end{bmatrix}, \overline{e_2} = \begin{bmatrix} e_{21} \\ e_{22} \\ e_{23} \end{bmatrix}, \overline{e_3} = \begin{bmatrix} e_{31} \\ e_{32} \\ e_{33} \end{bmatrix}, \; \boldsymbol{C} = \begin{bmatrix} e_{11} & e_{12} & e_{13} & 0 \\ e_{21} & e_{22} & e_{23} & 0 \\ e_{31} & e_{32} & e_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8}$$

Kierron muunnokset ovat määritelty kiertona 3D-avaruuden pääakselien ympäri. Kierrot x-, y- ja z-akselien ympäri ovat määritelty kaavan (9) matriiseilla. Näissä muunnoksissa kierto tapahtuu aina koordinaatiston pääakselin ympäri θ astetta. [14, s. 103-104]

$$\boldsymbol{R_x} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \boldsymbol{R_y} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$\boldsymbol{R_z} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{9}$$

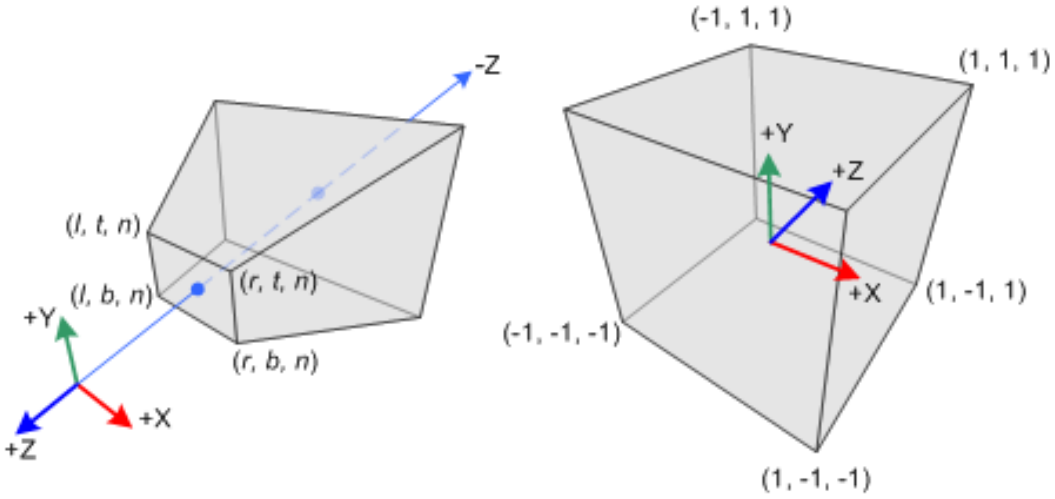
Jos kierrettävän geometrian keskipiste ei leikkaa kiertoakselia, kierto aiheuttaa myös siirtymää sijainnissa. Näin ollen tyypillisesti halutaan yhdistellä muunnoksia siten, että geometria keskitetään origoon, jonka jälkeen suoritetaan kiertomuunnos ja lopuksi geometria siirretään takaisin alkuperäiseen sijaintiin. [14, s. 102-103]

Usean muunnokset tekeminen tapahtuu kertomalla vektori muunnosmatriiseilla oikeassa järjestyksessä, jolloin muunnosmatriiseja voidaan yhdistää yhdeksi muunnosmatriisiksi. Kaava (10) esittää edellä mainittua kierto-operaatiota. Tässä $\boldsymbol{T_1}$ ja $\boldsymbol{T_2}$ ovat siirtomatriiseja vektorin siirrosta origokeskeiseksi ja siirrosta takaisin alkuperäiseen paikkaansa. Matriisi $\boldsymbol{R}$ on halutun kierron matriisi. [14, s. 102-103]

$$\boldsymbol{T_2 R T_1} = \boldsymbol{M} \Rightarrow \boldsymbol{T_2 R T_1}\bar{v} = \boldsymbol{M}\bar{v} \tag{10}$$

Kiertomatriiseja voidaan myös yhdistää samalla tavalla, muodostaen kokonaiskierron matriisi. Mikäli geometriaa halutaan kiertää muun kuin pääakselien ympäri, täytyy suorittaa siirto sellaiseen koordinaatistoon, missä tämä kiertoakseli on pääakselina. Kiertomuunnoksen jälkeen tarvitaan vielä muunnos takaisin alkuperäiseen koordinaatistoon. [14, s. 104]

Perspektiivimuunnosta käytetään kuvanmuodostuksen yhteydessä (kts. luku 4.4) perspektiiviprojektion muodostamisessa. Kuvaa muodostaessa erotetaan avaruudesta katkaistun pyramidin, frustumin, muotoinen alue, jonka sisältö muodostaa lopullisen kuvan. Projisoinnin ja syvyystiedon säilyttämisen kannalta tyypillisesti frustumin sisälle jäävälle osuudelle suoritetaan perspektiivimuunnos, joka geometrisesti havainnollistuu frustumin litistämisenä suorakulmaiseksi särmiöksi. Kuva 4.2 havainnollistaa muunnoksen ajatusta. [14, s. 187-189]



Kuva 4.2. *Havainnollistus perspektiivimuunnoksesta. Kuvassa on muunnoksen lisäksi myös suoritettu normalisointi, jolloin lopputuloksena on kuutio.[16]*

Perspektiivimuunnoksen matriisi on määritelty kaavassa (11), jossa *n* on katsojan etäisyys *etuleikkaustasosta*, ja *f* on katsojan etäisyys *takaleikkaustasosta*. Nämä tasot ovat tyypillisesti frustumin kärki ja pohja (etuleikkaustaso lähempänä alkuperäisen pyramidin kärkeä). [14, s. 189-190]

$$\boldsymbol{P} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{f+n}{n} & f \\ 0 & 0 & -\frac{1}{n} & 0 \end{bmatrix} \tag{11}$$

4.3 Kvaterniot ja kiertomuunnos

Kierto haluttujen akselien ympärillä edellä mainituilla matriiseilla vaatii monivaiheista laskentaa, kun jokaisen kierron suhteen tarvitaan potentiaalisesti siirto uuteen koordinaatistoon. Tätä varten on olemassa kehittyneempiä tapoja, kuten kiertoakselin ja kierron avulla määritellyt kiertomatriisit. Kiertojen esitysmuodot näissäkin tapauksissa ovat tyypillisesti 4x4 matriiseja, mutta monimutkaisemmilla kaavoilla määriteltynä.

Toinen vaihtoehtoinen tapa esittää kiertoja kompaktimmin on esittää nämä *kvaternioina*. Kvaternio on matemaattinen numerojärjestelmä, jonka kehittäjä on Hamilton vuodelta 1843 [14, s. 107]. Kvaterniot ovat laajennos kompleksiluvuista, jossa kahden komponentin sijasta on käytössä 4 komponenttia: reaaliosa ja 3 imaginaarikomponenttia [17, p. 75].

Kvaternioille toimivat kompleksilukujen tapaan yhteen- ja vähennyslaskut, ja näille on määritelty kerto- ja jakolaskuoperaatiot. Toisin kuin reaali- ja kompleksiluvuilla, kertolasku kvaternioilla on ei-kommutatiivinen, jolloin matriisien tapaan kvaternioiden kertolaskussa järjestyksellä on väliä. [17, p. 75-76]

Tietokonegrafiikassa yleinen käyttökohde kvaternioille on se, että niiden avulla voidaan määritellä kiertoja valittujen akselien ympärille [14, s. 111-112]. Paitsi että kierrot kvaternioina voidaan esittää tiiviimmässä muodossa kuin vastaavat kiertomatriisit, myös kiertojen välinen interpolointi voidaan toteuttaa yksinkertaisemmin kvaternioiden avulla [18, p. 245-246].

Kvaternioiden yleisimmät esitysmuotot on esitetty kaavassa (12). Kvaternioita voidaan myös esittää parina, jossa on reaaliluku ja vektorina esitetty imaginaariosat.

$$\boldsymbol{q} = a + bi + cj + dk = (a, b, c, d) \tag{12}$$

Jokaisella imaginaarikomponentilla on oma imaginaariyksikkönsä: *i*, *j*, ja *k*. Näiden keskinäiset suhteet ja käyttäytyminen on määritelty kaavassa (13). Jokaisen yksikön toinen potenssi on -1. Kertolaskussa yksiköiden järjestyksellä on merkitys, ja tästä johtuen kvaternioiden kertolaskut eivät ole kommutoivia. [14, s.109-110][17, p. 75-76]

$$\begin{aligned} i^2 = j^2 = k^2 = -1 = ijk \\ ij = -ji = k \\ ki = -ik = j \\ jk = -kj = i \end{aligned} \tag{13}$$

Kuten imaginaariluvuille, voidaan kvaternioille määritellä konjugaatti kaava (14), ja sitä kautta myös kvaternion normin kaava (15). [14, s. 109-110]

$$\boldsymbol{q}^* = a - bi - cj - dk \tag{14}$$

$$\|\boldsymbol{q}\| = \sqrt{\boldsymbol{q}\boldsymbol{q}^*} = \ldots = \sqrt{a^2 + b^2 + c^2 + d^2} \tag{15}$$

Geometrinen kierto 3D-avaruudessa voidaan muodostaa kvaternioilla halutun akselin ympärillä. Tämän muodostamiseen tarvitaan akselin suuntainen ja kaavan (16) mukainen yksikkövektori.

$$\bar{u} = \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix}, \ \|\bar{u}\|=1 \ . \tag{16}$$

Kun haluttu kierto on vastapäivään θ astetta, kierto yksikkövektorin $\bar{u}$ ympäri voidaan esittää kaavan (17) mukaisella yksikkökvaterniolla. [14, p. 111]

$$\boldsymbol{q_r} = \cos\frac{\theta}{2} + (u_x i + u_y j + u_z k)\sin\frac{\theta}{2}, \ \|\boldsymbol{q_r}\| = 1 \tag{17}$$

Kiertomuunnosta tehdessä käsiteltävä vektori voidaan samaistaa kvaternioon

$$\boldsymbol{v} = xi + yj + zk, \tag{18}$$

missä x, y ja z ovat vektorin koordinaatit 3D-avaruudessa. Reaaliosa näissä on tyypillisesti 0, kun puhutaan kierroista 3D-avaruuden pisteille. Kvaternioita, joiden reaaliosa on 0, kutsutaan myös ”puhtaiksi kvaternioiksi”. [14, s. 110-111]

Kiertomuunnos kvaternioilla tapahtuu siten kertomalla kierrettävä vektori (puhdas kvaternio) kierron kvaterniolla ja sen konjugaatiolla kaavan (19) mukaisesti [14, p. 111].

$$\boldsymbol{q_r}\, \boldsymbol{v}\, \boldsymbol{q_r}^* = \boldsymbol{v_m}\,. \tag{19}$$

Aivan kuten muunnosmatriisien kertolaskulla voidaan muodostaa kokonaismuunnosta esittävä matriisi, voidaan myös sama tehdä kvaternioilla. Kuten edellä onkin jo mainittu, kvaternioiden kertolaskuissa järjestyksellä on väliä. Kvaternioiden kertolaskun lopputuloksena on kokonaiskiertoa kuvaava kvaternio, ja tällä kerrotaan käyttäen kuten kaavassa (20) on esitetty. [14, p. 112]

$$(\boldsymbol{q}\, \boldsymbol{r})\boldsymbol{v}\, (\boldsymbol{q}\, \boldsymbol{r})^* = \boldsymbol{s}\, \boldsymbol{v}\, \boldsymbol{s}^* = \boldsymbol{v_m}\,. \tag{20}$$

Kvaterniokierrot voidaan myöskin muuntaa tavalliseksi kiertomuunnosta kuvaavaksi matriisiesitykseksi eri tavoin. Eräs tapa saada kvaterniolla esitetyn kierron kiertomuunnoksen matriisi, on esitetty kaavassa (21). Tämä kaava olettaa käytetyn kvaternion olevan yksikkökvaternio. [19]

$$\boldsymbol{q_r} = a + bi + cj + dk$$

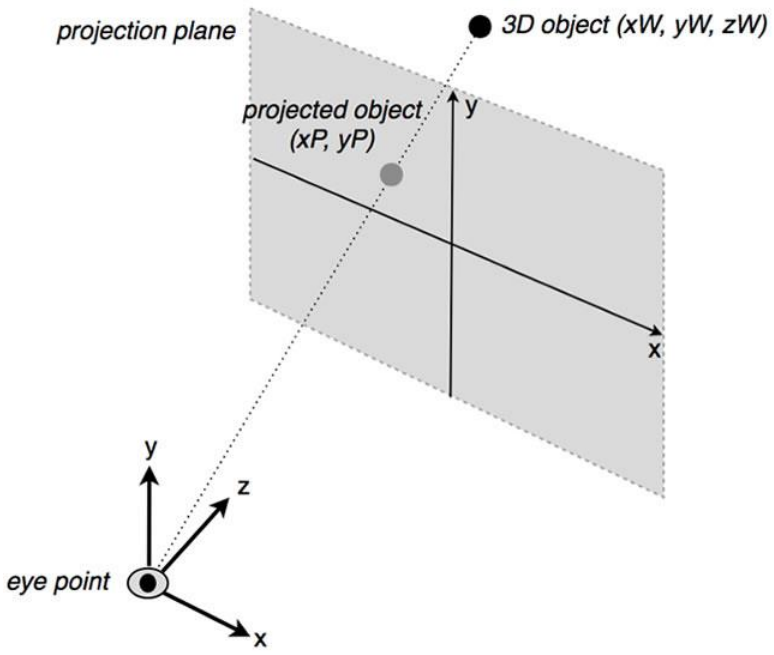
$$\boldsymbol{R_{q_r}} = \begin{bmatrix} 1-2c^2-2d^2 & 2bc-2ad & 2bd+2ac & 0 \\ 2bc+2ad & 1-2b^2-2d^2 & 2cd+2ab & 0 \\ 2bd-2ac & 2cd-2ab & 1-2b^2-2c^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{21}$$

Kvaterniosta johdettu kiertomatriisi edelleen vaatii, että geometria on keskitetty origon ympärille, jolloin tällaistakin kiertoa tyypillisesti käytetään kaavan (10) tavoin.

4.4 Kuvanmuodostus lyhyesti

Tietokonegrafiikassa kolmiulotteisesta avaruudesta muodostetaan yleisesti kaksiulotteista grafiikkaa, jota voidaan siten toistaa yleisimmillä näytöillä ja kuvaruuduilla. Tämä vaatii, että avaruuden kappaleiden geometriat käsitellään ja projisoidaan kaksiulotteiseksi.

Tätä varten geometrialle suoritetaan projektio kuvaustasolle. Kuvaustaso on taso, jonka pisteistä muodostetaan lopullinen kaksiulotteinen kuva. Projisoitaessa tyypillisesti määritellään kuvaustason lisäksi niin kutsutut *etu-* ja *takaleikkaustasot*, joilla rajataan näkyvän geometrian osuus. [14, s. 185, s. 187]



Kuva 4.3. *Havainnollistus perspektiivi projektiosta.[20]*

Perspektiiviprojektiossa avaruuden geometria projisoidaan muodostamalla projisointisuoria katsojan ja geometrian välillä, ja hakemalla tämän suoran leikkauspistettä kuvaustason kanssa. Katsoja on määritelty tyypillisesti pisteenä avaruudessa, jolloin kaikki projisoinnissa muodostetut suorat leikkaavat keskenään kyseisessä pisteessä. Ajatusta on havainnollistettu kuvassa 4.3. [14, s. 184-185]

Tätä kautta ajateltuna katsojan pisteestä lähtevät suorat, jotka kulkevat kuvaustason kulmien kautta, rajaavat kolmiulotteisesta avaruudesta pyramidin muotoisen kappaleen (joka jatkuu äärettömyyteen). Etu- ja takaleikkaustaso leikkaavat tästä frustumin (katkaistun pyramidin), jota kutsutaan näköfrustumiksi. Näköfrustumi määrittää alueen, jonka sisällä olevasta geometriasta muodostuu kuva. [14, s. 187-188]

Perspektiivimuunnoksessa näköfrustumin sisältö projisoidaan siten, että näköfrustumi muuntuu suorakulmaiseksi särmiöksi. Näin kaikki projisointisuorat muuttuvat rinnakkaisiksi, ja säilytetään tieto geometrian keskinäisistä syvyyseroista kuvaustason suhteen. [14, s. 188-189]

Perspektiivimuunnoksen jälkeen avaruuden geometria tyypillisesti normalisoidaan ja näköalueen ulkopuolelle jäävät geometriat leikataan pois. Normalisoinnissa näköfrustumin sisälle jäävän geometrian vektorit muunnetaan siten, että näiden koordinaattien arvot ovat

sovitettu $[-1,1]$ arvoalueelle. Geometrian normalisoinnin ja leikkaamisen jälkeen geometria projisoidaan kuvaustasolle ja lopuksi kuvaustason pisteet skaalataan kuvan koordinaatteihin. Samoin kuin perspektiivimuunnos ja aiemmin esitellyt muut perusmuunnokset, näkövolyymin normalisointi voidaan myös määritellä ja toteuttaa muunnosmatriisina. [14, s. 192-197]

Katsojan pisteestä alkava nelisivuinen pyramidi muodostaa katsojan näkökentän, ja tämän pyramidin vastakkaisien sivujen väliset kulmat määrittelevät näkökentän laajuuden. Näitä kulmia kutsutaan yleensä näkökentän leveydeksi ja korkeudeksi. Näkökentän leveydestä tai korkeudesta käytetään myös englanninkielistä termiä *field of view*(FOV), jota käytettäessä puhutaan myös vaakasuorasta (*horizontal*) ja pystysuorasta (*vertical*) näkökentän laajuudesta. Näkökentän leveys (tai korkeus) on keskeinen konsepti tietokonegrafiikassa, ja jotkin ohjelmistokehykset käyttävätkin näkökentän leveyttä tai korkeutta perspektiiviprojektion määrittelemiseen [21].

5. WEBGL

WebGL on OpenGL ES -spesifikaatioon (Open graphics library for embedded systems) pohjautuva Javascript API (Application programming interface, ohjelmointirajapinta) laitteistokiihdytetyn tietokonegrafiikan tuottamiseen selainympäristössä. Se hyödyntää HTML5-ympäristön canvas-elementtejä OpenGL (Openg graphics library) kontekstina, ja tarjoaa OpenGL ES -spesifikaatiota mukailevan rajapinnan grafiikan tuottamiseen näille. [22]

5.1 Historiasta

Silicon Graphics, Inc. (SGI) oli suunnitellut omia IRIX-pohjaisia tietokoneitansa varten IRIS GL -nimisen kirjaston tietokonegrafiikan tuottamiseen. Tämä kirjasto oli kaupallinen ja suljetun lähdekoodin grafiikkakirjasto, josta SGI päätti julkistaa avoimen, siistitymmän ja laitteistoriippuvuuksista irrotetun version nimellä OpenGL. OpenGL 1.0 julkistettiin tammikuussa 1992. [17, p. 6-7]

Vuonna 2008 OpenGL spesifikaatiosta päädyttiin tekemään kaksi ”profiilia”: *compability* ja *core*. Näistä ensimmäisellä haluttiin taata yhteensopivuus aina OpenGL 1.0 versioon asti, kun taas jälkimmäisellä katkaistiin yhteensopivuus taaksepäin. Tämän tekstin kirjoittamisen aikaan uusin versio on OpenGL 4.5, julkaistu elokuussa 2014. Nykyään Khronos Group konsortiumia osana oleva Architectural Review Board (ARB) suunnittelee, hallinnoi ja tuottaa OpenGL-spesifikaatioita. [17, p. 8-9][23]

Sulautettujen järjestelmien sekä mobiililaitteiston kehittyessä kyvykkäämmäksi tuottamaan grafiikkaa, kiinnostus 3D-grafiikan tuottamiseen näillä alustoilla alkoi kasvaa. Vuonna 2003 julkaistiin tätä varten OpenGL for Embedded Systems eli lyhennettynä OpenGL ES. OpenGL ES 1.0 rakennettiin OpenGL version 1.3 pohjalta, karsimalla siitä harvemmin käytettyjä ominaisuuksia pois. Vuonna 2007 julkaistiin OpenGL ES 2.0, ja tämän tekstin kirjoittamisen aikaan viimeisin julkaistu versio OpenGL ES:stä on 3.2, joka julkaistiin syyskuussa 2015. [17, p. 706-708][24]

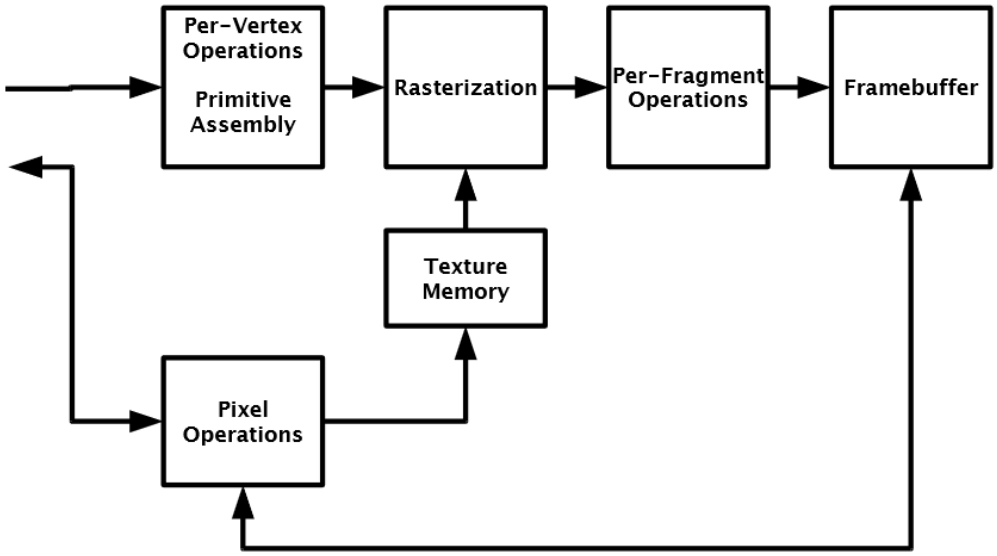
Maaliskuussa 2011 Khronos Group julkaisi OpenGL ES 2.0 -standardin pohjalle rakennetun grafiikkakirjaston selaimille, nimellä WebGL 1.0 [25]. Helmikuussa 2017 kirjastosta julkaistiin WebGL 2.0, joka puolestaan on rakennettu OpenGL ES 3.0 pohjalle [26].

5.2 OpenGL ES 2.0 liukuhihna

OpenGL toimii kuten liukuhihna, lukien sille annettuja syötteitä ja käsitellen näitä liukuhihnan eri vaiheissa. Liukuhihnamainen rakenne mahdollistaa käskyjen tehokkaamman

käsittelemisen, koska liukuhihna käsittelee useita käskyjä samanaikaisesti sen eri osissa. [17, p. 4]

OpenGL versiossa 2.0 otettiin käyttöön liukuhihnarakenne, joka sallii käyttäjien tuottaa ja hyödyntää omia *sävyttimiä* osana liukuhihnaa (sävyttimistä lisää aliluvussa 5.3). Vastaava liukuhihnan rakenteen muutos tuli OpenGL ES 2.0 -versioon, ja sitä kautta se on myös käytössä WebGL-kirjastossa. Koska nämä käyttävät samaa rakennetta, tässä aliluvussa tehdään lyhyt katsaus OpenGL ES 2.0 version liukuhihnan rakenteeseen. Kuvassa 5.1 on korkean tason kuvaus liukuhihnan rakenteesta, joka on käytössä OpenGL ES 2.0 -versiossa. [17, p. 707-708][27, p. 12-14]



Kuva 5.1. *Yleiskuvaus OpenGL ES 2.0 liukuhihnan rakenteesta. [27, p. 13]*

Ensimmäinen vaihe liukuhihnalla on *verteksien* käsitteleminen. Tässä vaiheessa järjestelmälle annetut 3D-avaruuden pisteet, joita tyypillisesti kutsutaan sanalla verteksi, käsitellään ja piirtokäskyissä määritetyllä tavalla näistä muodostetaan *primitiivejä*, joita ovat mm. kolmio tai kolmioviuhka. Liukuhihnalle välitetyt verteksit käsitellään yksitellen, ja näistä muodostetut primitiivit lopuksi muunnetaan ja leikataan näköfrustumin sisälle. [17, p. 10][27, p. 11, p. 15-18]

Rasteroinnissa muodostetaan primitiiveistä ikkunakoordinaattia vastaavia väri ja syvyystietoja, joita kutsutaan termillä *fragmentti*. Nämä sisältävät rasterointituloksen kyseiselle pisteelle, ja nämä välitetään seuraavalle vaiheelle. Fragmentit käsitellään siten, että jo-

kaista kuvapistettä varten saadaan sen lopullinen väri, ennen kuin ne kirjoitetaan kehyspuskurille. Kehyspuskurin lopputulos välitetään lopulta käyttäjän näkyville. [27, p.11, p. 48]

Kuten kuvasta 5.1 näkee, kehyspuskurille kirjoitettua kuvadataa voidaan lukea takaisin liukuhihnalla, käyttää tulevissa operaatioissa. [27, p. 11]

Vaikka liukuhihna on määritelty vaiheittaiseksi ja se on yleensä kuvattu järjestyksessä, tämä ei ole tiukka sääntö vaan ohjenuora ja tapa organisoida sitä. Se miten liukuhihna toimii, riippuu toteutuksesta, tyypillisesti näytönohjaimien ajureista tai laitteistosta. [27, p. 11]

5.3 Sävyttimet

Kuten edellä lyhyesti esiteltiin, sävyttimet ovat käyttäjän pieniä ohjelmia, joita suoritetaan osana OpenGL liukuhihnaa. Näiden tehtävä on käsitellä liukuhihnalta tulevia syötteitä ja muodostaa näistä tietynmuotoisia ulostuloja seuraaville liukuhihnan vaiheille. Sävyttimillä yleensä tehdään verteksien ja tuotettujen fragmenttien käsittelemistä. [17, p. 16-17][27, p. 26, p. 86]

Sävyttimet tyypillisesti käännetään *sävytinohjelmaksi* ajonaikaisesti, ja nämä ohjelmat välitetään OpenGL toteutukselle suoritettavaksi. Esikäännettyjä sävytiohjelmia voidaan myös ladata ja välittää suoraan toteutukselle. [27, p. 26]

OpenGL ES 2.0 -versiossa on käytössä kaksi sävytintä: *verteksisävytin* ja *fragmenttisävytin*. Nämä sisältyvät edellisessä aliluvussa esitellyssä liukuhihnamallissa vastaavasti ennen primitiivien muodostamista ja tuotettujen fragmenttien käsittelyyn. Myöhemmissä OpenGL ES -versioissa on tullut liukuhihnamalliin lisäksi muun tyyppisiä sävyttimiä. [17, p. 17][27 p. 11]

Oikeellisen toiminnan saavuttamiseksi OpenGL ES 2.0 -versiosta eteenpäin täytyy asettaa käyttöön vähintään verteksi- ja fragmenttisävyttimet. Mikäli näitä sävytinohjelmia ei ole asetettu käyttöön, liukuhihnassa nämä vaiheet toimivat ennalta määrittelemättömällä tavalla riippuen toteutuksesta. [27, p. 26, p. 86]

Verteksisävytin suoritetaan liukuhihnalta vastaanotetuille vertekseille ennen niiden muuntamista primitiiveiksi. Se ottaa parametreinaan puskuroituja verteksejä ja muuta sille annettua dataa. Näitä hyödyntäen sävytinohjelma muodostaa lopullisen verteksin sekä haluttaessa lisädataa, jotka välitetään liukuhihnalla eteenpäin. Lisädata on tyypillisesti tarkoitettu lopullisten fragmenttien käsittelemistä varten fragmenttisävyttimessä, ja se onkin yleensä sidottu verteksien pohjalta luotujen primitiivien kulmapisteiden tietoihin. Verteksisävyttimiltä vaaditaan, että jokaista luettua yksittäistä verteksiä kohden tuotetaan sävyttimen ulostulona aina täsmälleen yksi verteksi. [17, p. 29-30, p. 224, p. 310]

Yleinen käyttökohde verteksisävyttimille on geometristen muunnoksien tekeminen yksittäisille vertekseille tai pohjatyö muodostetun primitiivin jatkokäsittelyä varten fragmenttisävyttimessä. Esimerkiksi valaistuslaskentoja varten voidaan jokaista verteksiä kohden muodostaa kappaleen pinnan normaalin vektori lisädatana, ja primitiivin rasteroinnissa muodostettujen fragmenttien käsittelyssä laskea fragmentin valaistusta ottaen nämä pinnan normaalivektorit huomioon.

Fragmenttisävytin suoritetaan rasteroinnin tuottamille fragmenteille. Parametrina fragmenttisävytin saa tuotetun fragmentin, ja kyseistä fragmenttia koskevat muut tiedot, mm. edellä mainittua lisädataa jos sellaista on. Lisädata voi olla joko verteksisävyttimeltä sellaisenaan vastaanotettu tietue, tai se voi olla interpoloitu fragmenttiin liittyvien lisädatojen väliin. Näiden avulla fragmenttisävytin manipuloi fragmenttiin liittyvää väritietoa, esimerkiksi muodostamalla valaistuksen vaikutusta kyseiseen fragmenttiin. Vaadittu ulostulo on fragmentin väri. [17, p. 42, p. 324][27, p. 88-89]

Fragmenttisävyttimellä yleisesti luodaan tietokonegrafiikassa efektejä. Kuvatekstuurin lukeminen tuotetaan yleensä tässä vaiheessa, etsimällä fragmenttia vastaava piste kuvatekstuurista ja määrittelemällä tämän värin pohjalta tuotetun fragmentin väriä. Tekstuurien lisäksi valaistukseen liittyvät efektit voidaan toteuttaa täällä.

Verteksi- ja fragmenttisävyttimet määritellään omalla GLSL-ohjelmointikielellään. Kielen nimi tulee sanoista *OpenGL Shading Language*. Kielenä GLSL on C-kielen sukuinen ja sisältää joitain C- ja C++-kielestä tuttuja perusrakenteita, mutta sisältää poikkeamia esimerkiksi matriisien ja vektoreiden ollessa sisäänrakennettuna itse kieleen. Jokaista OpenGL-spesifikaation versiota vasten on jokin versio GLSL-kielestä, joka myös määritellään sävytinohjelman lähdekoodin alussa kääntäjää varten. [17, p. 17, p. 188]

Sävytinohjelmat käsitellään ohjelmakoodissa joko esikäännettyinä binääreinä, tai lähdekoodina, joka käännetään eksplisiittisesti. OpenGL ES 2.0 toteutukselta on vaadittu vähintään, että se tukee jompaakumpaa näistä tavoista. Kun sävytinohjelman binääri on haettu tai tuotettu, se välitetään käytettäväksi eksplisiittisillä komennoilla. [27, p. 26-31]

Kuten edellä todettiin, OpenGL ES 2.0 sisältää ainoastaan verteksisävyttimen ja fragmenttisävyttimen. OpenGL ES-spesifikaation uudemmissa versioissa sekä itse OpenGL-spesifikaatiossa on nykyään olemassa myös tesselaatio-, geometria- ja laskentasävyttimiä. Tesselaatiosävyttimellä verteksien pohjalta muodostetaan interpolaatiolla dynaamisesti lisää verteksejä ja geometriasävyttimellä manipuloidaan jo muodostettuja primitiivejä ennen niiden rasterointia. Laskentasävytin on tarjolla muun kuin graafisen laskennan toteuttamisessa, ja ei varsinaisesti ole osana itse grafiikkaliukuhihnaa. [17, p. 17, p.32, p. 36, p. 47]

5.4 Erot OpenGL-kirjastoon nähden

WebGL on suunniteltu ja rakennettu selainympäristöjä varten, ja toimii siten HTML5-ympäristössä. HTML5-standardissa esiteltiin canvas-elementti, joka tarjoaa Javascript API:n grafiikan piirtämiseen. WebGL hyödyntää tätä elementtiä 3D-grafiikan piirtämiseen selaimen näkymään. [22]

Selainympäristö johtaa myös siihen, että WebGL:n resurssien käsittely poikkeaa osittain OpenGL-ympäristön vastaavista. Tyypillisesti resurssit kuten tekstuurit ja geometria, haetaan ja ladataan verkkoyhteyden välityksellä. Hyvin yleisesti WebGL-sovelluksille haetut resurssit tulevat samasta lähteestä kuin kyseisen verkkosivun sisältö. Tilanteissa, joissa osa datasta ei ole lähtöisin samasta alkulähteestä, WebGL-spesifikaatio kieltää tällaisten resurssien käytön sellaisinaan. [28]

Kun tulee tarve käsitellä resursseja eri lähteestä, täytyy hankkia alkuperäisestä lähteestä lupa kyseisien resurssien käyttämistä varten. Tämä voidaan hoitaa käyttämällä Cross Origin Resource Sharing (CORS) -mekanismia. Tällä mekanismilla voidaan pyytää ja vastaanottaa resursseja ulkopuolisista lähteistä, välittämällä pyynnöissä informaatiota kyselyn alkuperästä ja tarvittaessa tunnistetietoja. Vaihtoehtoisesti palvelin voi merkitä CORS-merkinnän suoraan vastauksiin esimerkiksi siinä tilanteessa, että pyydetty resurssi on julkista. CORS-mekanismi toimii HTTP-viestinnässä välitettyjen viestien otsikkotiedoissa. [28][29]

Teknisempiä eroja WebGL ja OpenGL välillä ovat mm. liukulukujen rajaus 32-bittisiksi, tekstuurikokojen rajaukset, 3D-tekstuurien puute ja jotkin semanttiset poikkeamat. [30]

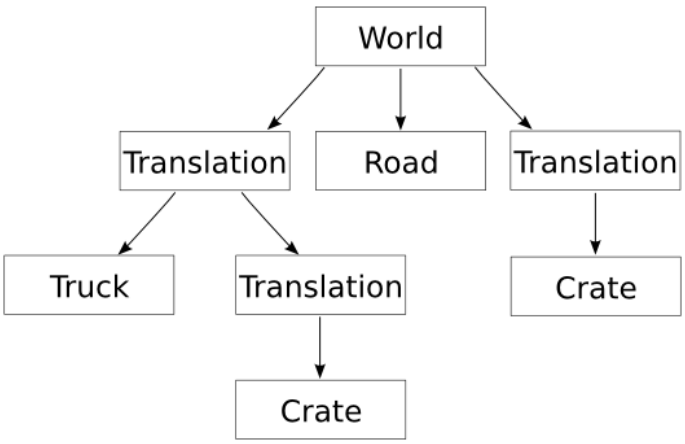
5.5 WebGL-ohjelmistokehyksistä

Vaikka WebGL on Javascript API, sen käyttäminen vaatii paljon yksityiskohtaista datan käsittelyä, kuten esimerkiksi verteksidatan puskurointia ja välitystä WebGL-toteutuksen liukuhihnalle. Tästä voi muodostua kompleksista koodia, jota sellaisenaan ei ole helppo laajentaa ja ylläpitää. Tätä ongelmaa helpottamaan on luotu useita ohjelmistokehyksiä tukemaan WebGL-sisällön tuottamista. Monet kehykset tuovat mukanaan WebGL:n lisäksi myös muita lisäominaisuuksia, kuten työkaluja äänitiedostojen toistamiseen ja tapoja lukea käyttäjän syötteitä. Tässä aliluvussa käydään lyhyesti muutamia WebGL-ohjelmistokehyksiä.

Three.js on Ricardo Cabellon vuonna 2010 julkaisema Javascript-kehys grafiikan tuottamiseen selaimessa. Tuki WebGL:lle tuli myöhemmin, kun WebGL itse julkaistiin. Three.js on julkaistu MIT-lisenssillä. Three.js tarjoaa työkalut 3D-sisällön tuottamiseen selainympäristössä. [31]

Keskeinen perusrakenne sovelluksilla, jotka ovat tehty hyödyntäen Three.js-kehystä, on *näkymägraafi* (*scenegraph*), kamera, ja renderöijä (*renderer*). Kamera määritellään omana objektina erillään näkymägraafista, ja renderöijä muodostaa kameran ja näkymägraafin pohjalta kuvan. [32]

Näkymägraafi on tietorakenne, johon ohjelmakoodissa kootaan virtuaalisen maailman sisältämät objektit. Kuvassa 5.2 on esitelty geneerinen tapa muodostaa näkymägraafi. Näkymägraafiin sijoitetut objektit tyypillisesti sisältävät kyseisen objektin geometrian ja muut tarvittavat resurssit. Esimerkiksi Kuvan 5.2 rekka-objekti (kuvassa nimellä Truck) voisi sisältää siihen liittyvän geometrian, tekstuurit sekä rekan sijaintiin ja liikkumiseen liittyvää tietoa. [33]



Kuva 5.2. *Esimerkki näkymägraafista, johon on koottu autotie, rekka ja pari laatikkoa. Koska rekka ja laatikot sijaitsevat 3D-avaruudessa eri kohdissa, näkymägraafissa näitä ennen on määritelty siirtmät.[33]*

Toinen yleisesti käytetty WebGL-kehys on Babylon.js. Se on Javascript- ja Typescript-kehys pelien tuottamiseen hyödyntämällä WebGL-rajapintaa, ja on julkaistu Apache-lisenssillä. Babylon.js noudattelee yllä kuvattua rakennetta, joskin kamera on osa näkymägraafia, ja renderöijää ei ole määritelty suoraan erikseen. Babylon.js sisältää lisäominaisuuksia joita ei Three.js sisällä, mm. fysiikkamoottorin. [34][35]

Copperlich on Ambieran kehittämä WebGL-kehys. Se julkaistiin vuonna 2010, ja on ilmainen käyttää. Poikkeamana edellä mainittuihin ohjelmistokehyksiin on näkymägraafin muodostaminen ja rakentaminen erilliseen tiedostoon, joka ladataan javascript-koodissa käyttöön. Näkymägraafin luomista ja muokkaamista varten Ambiera tarjoaa kaupallista sovellusta nimeltä CopperCube, joskin on myös mahdollista määritellä näkymägraafeja käsin kuten edellä mainituissa kehyksissä. [36][37]

Unity on Unity Technologiesin kehittämä monialustainen pelimoottori. Se julkaistiin alunperin vuonna 2005 OS X -käyttöjärjestelmälle, mutta tätä on sen jälkeen laajennettu

eri pelikonsoleille ja käyttöjärjestelmille. Unity on kaupallisesti lisensoitu. Unity 5 versiosta lähtien se tukee myös WebGL ympäristöä, tukien versioita WebGL 1.0 ja 2.0. [38][39]

5.6 WebGL tehdyn työn puitteissa

Tässä tekstissä kuvatun työn tekemisen aikana WebGL versiota 2.0 ei vielä oltu julkaistu, jolloin käytössä oli ainoastaan WebGL versio 1.0 ja sitä hyödyntävät ohjelmistokehykset. Ohjelmistokehyksistä Three.js ja Babylon.js olivat ne kehykset, jotka olivat harkinnassa ottaa käyttöön. Näiden kahden väliltä lopulta päädyttiin käyttämään toteutuksessa Three.js-kehystä.

Tämän työn puitteissa käytetyt ominaisuudet Three.js-kehyksestä löytyvät myös muista kehyksissä, kuten edellä mainitusta Babylon.js-kehyksestä. Tämän valossa vastaavan toiminnallisuuden olisi voinut toteuttaa muullakin ohjelmistokehyksellä, ja siten käytetyt ominaisuudet eivät suoraan olleet syy valinnan tekemiselle. Valinnan perusteena olivat valmiit ohjeistukset HTML5 video-elementillä olevan videokuvan käyttämiseen WebGL-tekstuurina kyseisen kehyksen avulla.

WebGL-rajapinnan suoralta käyttämiseltä vältyttiin enimmäkseen hyödyntämällä Three.js:n ominaisuuksia, ja pitämällä geometrian kompleksisuus yksinkertaisena. Three.js-kehyksen sisäänrakennetusta kuutiomallista venytettyjä suorakulmioita hyödynnettiin lisätyn todellisuuden objekteina, jolloin käytetty 3D-geometria pysyi hyvin yksinkertaisena ja helppona hallinnoida ohjelmakoodissa. Yksinkertainen geometria myös nopeutti prosessointia, mikä oli tärkeää koska kyseessä oli reaaliaikaiseen videokuvaan sidottua virtuaalisen maailman dataa.

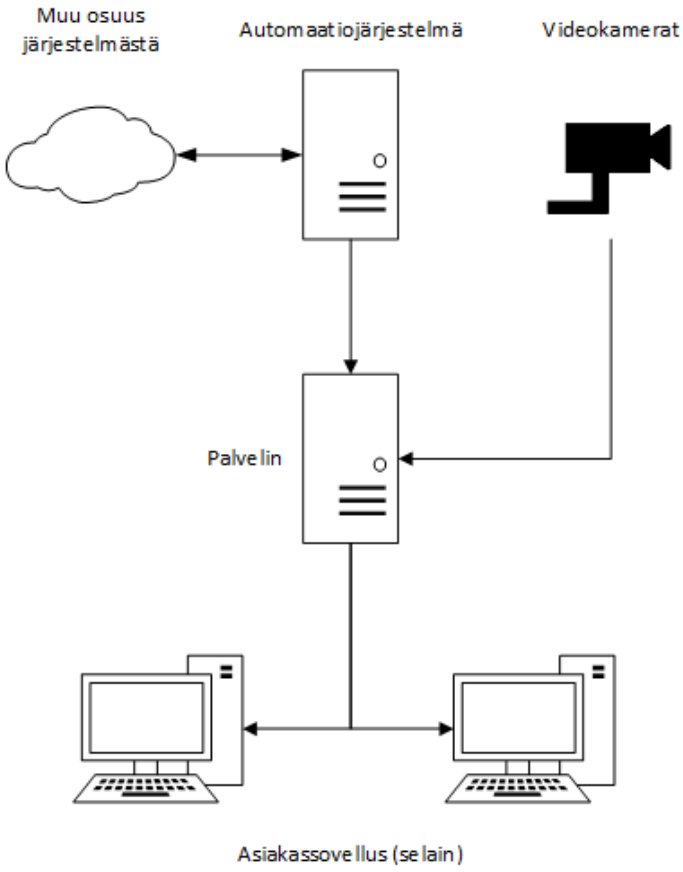
Sävyttimien käyttöä yritettiin, mutta se osoittautui tehtyjen valintojen valossa hankalaksi hyödyntää, ja siten päädyttiin käyttämään sävyttimienkin osalta ohjelmistokehyksen sisäänrakennettuja sävyttimiä. Näin ollen lopullisessa prototyypissä ei tarvittu käsintehtyjä GLSL-kielisiä sävytinohjelmia, eikä niiden kääntämistä ja käyttöön ottamista.

Luvussa 4.3 esiteltyjä kvaternioita hyödynnettiin, ja näille löytyi suoraan tuki Three.js-kehyksestä. Three.js dokumentoi käyttävänsä kvaternioiden muuntamisessa kierron matriisiksi luvussa 4.3 esitettyä kaavaa (21). Muita geometrisia työkaluja, joita tässä työssä hyödynnettiin, oli säteenjäljitys.

6. TOTEUTETTU JÄRJESTELMÄ

6.1 Yleiskuvaus toiminnasta

Toteutettu järjestelmä suunniteltiin asiakas-palvelin mallisena, ja sen yksinkertaistettu rakenne on esitelty kuvassa 6.1. Asiakas ottaa selaimella yhteyden palvelimeen, jolta selain hakee verkkosivun ja sitä kautta suorittaa Javascript-kielellä toteutettua WebGL-sovellusta. Tämä sovellus vastaavasti keskustelee palvelimen kanssa, ja hakee tältä videokuvaa ja informaatiota, ja muodostaa näistä reaaliaikaisesti lisätyn todellisuuden näkymän käyttäjälle.



***Kuva 6.1** – Yleisluonnos toteutuksen järjestelmän rakenteellisesta ajatuksesta*

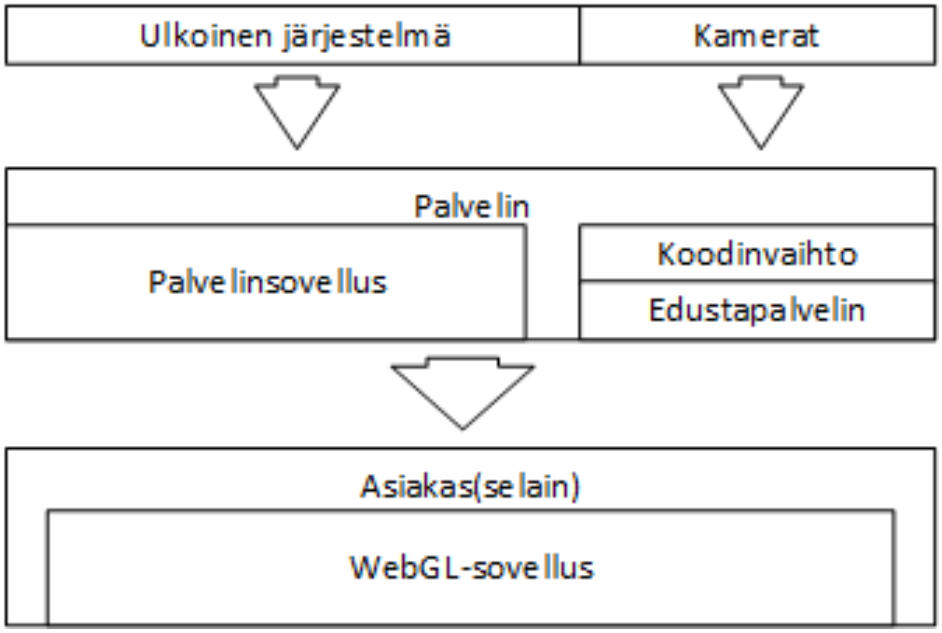
Palvelin vastaavasti hakee käytetystä järjestelmästä tietoa sen tilasta, jäsentää näistä asiakassovelluksen ymmärtämää tietoa ja välittää nämä tälle. Samaan aikaan se myös vastaanottaa valvontakameran kuvaa ympäristöstä, käsittelee tämän ja välittää tätä asiakassovelluksen ymmärtämässä muodossa.

Kuten kuva 6.1 havainnollistaa, tehty järjestelmä tukee useita asiakkaita yhtäaikaisesti, julkaisten informaatiota kaikille asiakkaille jotka ovat palvelimeen yhteydessä. Vaikka

toteutettu järjestelmä oli suunniteltu käyttäen tätä rakennetta, aikarajoitteiden puitteissa järjestelmää suunniteltiin ja testattiin ajamalla palvelinosuutta samalla koneella kuin itse selainta.

6.2 Yksityiskohtaisempi yleiskuvaus toteutuksesta

Kuvassa 6.2 on esitelty hienojakoisempi yleiskuvaus toteutuksen keskeisistä osista ja tiedon liikkumisesta. Palvelinosuus on jaettu kaksiosaiseksi: videokuvan muuntaminen formaatista toiseen, eli *koodinvaihdon* (*transcoding)* tekeminen, sekä erillisenä tästä varsinaisen tiedonvälityspalvelun.



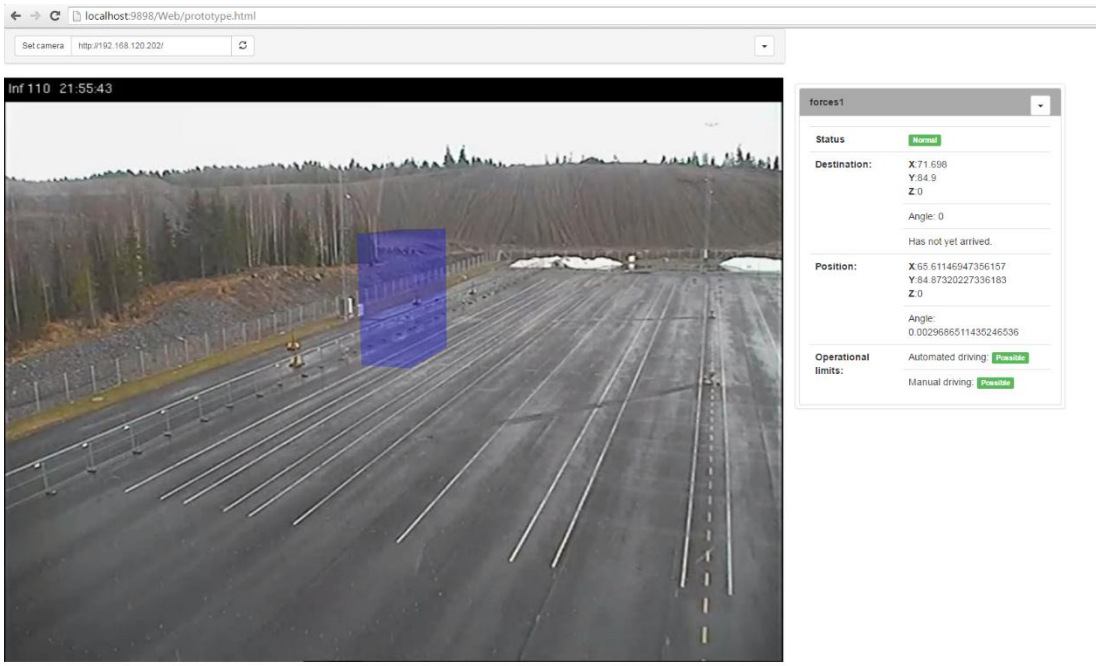
***Kuva 6.2** – Yksinkertaistettu vuokaavio informaation liikkumisesta ja keskeisistä järjestelmän osista*

Palvelinsovellus kuuntelee kohdejärjestelmää, ja vastaanottaa tältä informaatiota tilasta- ja ympäristöstä. Se esikäsittelee nämä ja välittää näitä asiakkaille. Lisäksi se hakee kohdejärjestelmän ja kameroiden nykyisen tilan aina asiakkaan tätä pyytäessä. Tämän pyynnön asiakas tekee alustaessaan itsensä alkutilaan. Yhteyden muodostuksen jälkeen asiakkaan ja palvelinosuuden välillä on jatkuva yhteys, jonka kautta palvelin välittää informaatiota muutoksista asiakkaille.

Projektissa käytetty kohdejärjestelmä julkaisee reaaliajassa informaatiota. Palvelin kuuntelee näitä muutoksia, ja muodostaa näistä pienempiä paketteja, joita se välittää kaikille asiakkaillensa. Koska käytetyt kamerat eivät julkaisseet automaattisesti mitään informaatiota vastaavalla tavalla, pyydetään uutta tilaa näiltä tasaisin aikavälein, ja vastaavalla tavalla muutokset välitetään asiakkaille.

Palvelinsovelluksen lisäksi toteutettiin tästä erillisinä sovelluksina koodinvaihtosovellus ja tätä varten oma edustapalvelin. Koodinvaihtosovellus lukee kameroiden tuottamaa videon suoratoistoa, suorittaa tälle koodinvaihdon, ja tuottaa tästä uutta suoratoistoa lokaalissa ympäristössä. Asiakaspuolen verkkosivu vastaavasti pääsee lukemaan tätä suoratoistoa käyttämällä edustapalvelinta.

Asiakasosuus koostuu ensisijaisesti HTML5-verkkosivusta ja siihen liittyvästä suoritettavasta Javascript-ohjelmakoodista, joka hoitaa paitsi videokuvan yhdistämisen lisätyn todellisuuden kanssa, mutta myös itse virtuaalisen maailman tiedon ylläpitämisen ja säilytyksen.



***Kuva 6.3** – Kuvankaappaus toteutetun järjestelmän selainnäkymästä. Kuvassa simuloidun automaatiojärjestelmän dataa on visualisoitu kameran kuvaan.*

Selainsovellus avaa käyttäjälle näkymän, joka on kaksiosainen: kamerasäädin ja varsinainen videokuva. Toteutetun selainosuuden ulkoasua on havainnollistettu kuvassa 6.3.

Sivun yläreunassa oleva palkki sisältää tiedot valitusta kamerasta ja piilotettuna kontrolleja lisätyn todellisuuden virtuaalisen kameran hienosäätöön. Oletuksena näkyvissä on kenttä, jolla on mahdollisuus vaihtaa käytössä olevan kameran IP-osoitetta. Piilotetut kentät sisältävät kameralta saadut arvot, ja hienosäätökenttiä joilla voidaan korjata poikkeamia tai virhettä kameran asennoissa. Poikkeamien ja tietojen manuaalinen muokkaus vaikuttaa suoraan lisätyn todellisuuden geometriaan ja asennoitumiseen videokuvassa. Mikäli kamera fyysisesti liikkuu tai päivittää tietojansa, palvelimelta vastaanotettu muutosviesti täyttää automaattisesti siihen liittyvät kentät välittömästi.

Videokuva-osuus sisältää videokuvan, ja siihen päälle piirretyn lisätyn todellisuuden. Oletuksena mitään geometriaa ei piirretä näkyviin. Kun käyttäjä vie hiiren jonkin seuratun kohdejärjestelmän objektin päälle, ilmestyy hiiren viereen infolaatikko, mistä selviää kyseisen objektin id tai nimi. Käyttäjä voi valita objektin painamalla sitä hiirellä, jolloin kyseinen objekti tulee näkyviin. Samalla aukeaa kuvassakin näkyvä ponnahdusikkuna, missä on reaaliaikaisesti päivittyvää informaatiota kohteesta.

Piirretty objekti liikkuu ja kääntyy reaaliaikaisesti noudattaen kohdejärjestelmän antamia tietoja oikeasta objektista. Mikäli käytetty kamera muuttaa orientaatiotaan, päivittyvät nämä tiedot pienellä viiveellä näkymään. Mikäli hiiren painalluskohdassa kuvassa sijaitsee syvyystasolla useampi objekti, kaikki nämä piirretään, ja informaatioikkuna sisältää eriteltynä kaikkien näiden tiedot.

Kuvassa 6.3 ei ole oikea automaatiojärjestelmä käynnissä, vaan simulaatiossa virtuaalinen objekti on asetettu kuvassa näkyvään sijaintiin.

6.3 Palvelimen sovellukset

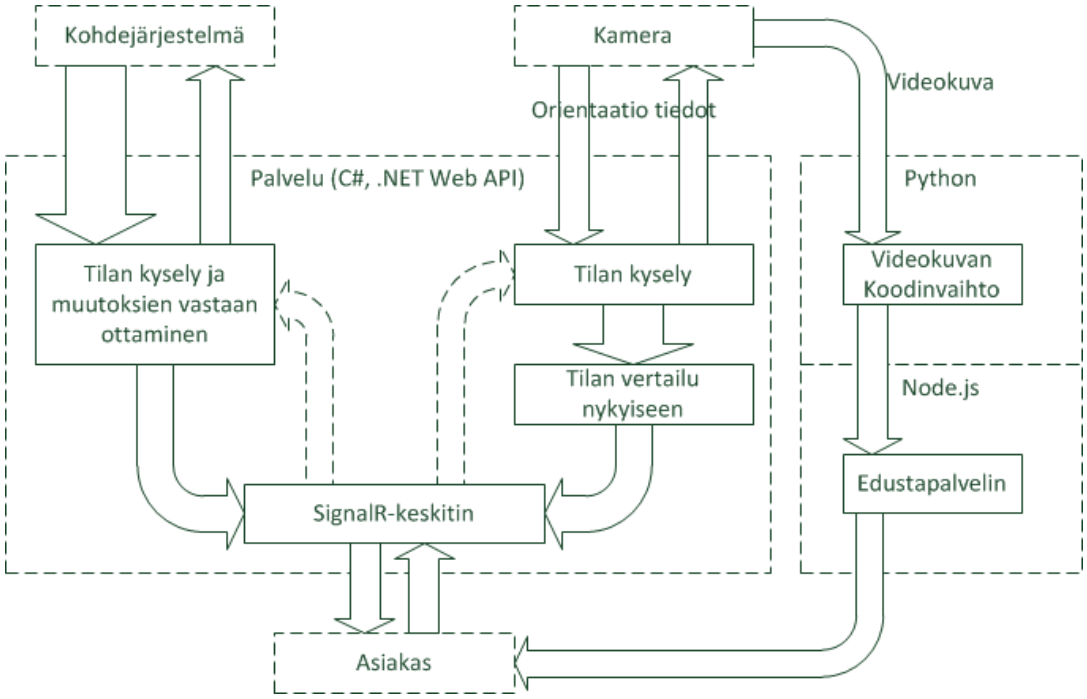
Kuvassa 6.4 havainnollistetaan palvelimen puolen sovelluksien rakenteita. Kaiken kaikkiaan toteutettuja sovelluksia oli kolme: palvelinsovellus tiedonvälitykseen, videokuvan käsittely ja suoratoisto sekä edustapalvelin videokuvaa varten. Asiakas luki videokuvan edustapalvelimen kautta HLS-suoratoistona (HTTP live streaming), ja kuunteli SignalR-kirjaston avulla palvelinsovellukselta tietoja.

Palvelinsovellus toteutettiin ensisijaisesti C#-kielellä, hyödyntämällä ASP.NET Web API-kehystä ja SignalR-kirjastoja. ASP.NET Web API on ohjelmistokehys .NET-ympäristöön, jonka avulla voidaan luoda HTTP-viestintään pohjaavia verkkopalveluja. ASP.NET Web API:n kanssa höydynnettiin OWIN-kirjastoja (Open Web Interface for .NET), jonka avulla toteutettua verkkopalvelua pystyttiin isännöimään omalla palvelinkoneella ja omassa kehitysympäristössä. [40][41]

SignalR-kirjasto on reaaliaikaista tiedonsiirtoa varten toteutettu kirjasto. Se hyödyntää ympäristöstä riippuen eri tekniikoita yhteyksien muodostamiseen, ensisijaisesti käyttäen HTML5 WebSocketteja ja näiden puuttuessa muita vanhempia tekniikoita. [42]

Näiden lisäksi käytössä olivat myös tarpeelliset kirjastot kohdejärjestelmän kanssa keskustelemiseen ja sen kuuntelemiseen. C# ja .NET-ympäristö valittiin ensisijaisesti sen takia, että käytetyn kohdejärjestelmän kanssa kommunikointiin tarvittavat kirjastot olivat toteutettu vastaavaan ympäristöön.

Palvelinsovellus keskustelee kameroiden kanssa käyttäen HTTP-viestintää. Käytetyt kamerat tarjosivat CGI-rajapinnan (Common gateway interface), jolloin kameran orientaatiotiedot saatiin pyytämällä niitä tietyn muotoisia URI-osoitetta (Unique resource identifier) käyttäen.



Kuva 6.4. *Yksityiskohtainen esitys palvelimen puolen sovelluksista ja niiden rakenteista ja rooleista*

Jokaista kameraa varten palvelun puolella ylläpidetään viimeisintä tunnettua tilaa näistä, ja palvelu kysyy tietyn ajan välein kameroiden nykyisiä tiloja muutoksien varalta. Saatua tilaa verrattaan edeltävään tilaan, ja jos havaitaan muutoksia, päivitetään uudet tiedot sisäiseen tilaan ja muodostetaan lyhyt päivitysviesti lähetettäväksi kaikille asiakkaille. Kameroiden tilat koostuvat kameran orientaatiosta, näkökentän laajuuden arvosta ja ennalta konfiguroiduista virheiden ja poikkeamien arvioinneista. Näistä orientaatio ja näkökentän laajuus muodostetaan heti kyselyn tuloksista.

Kohdejärjestelmään palvelu keskustelee käyttämällä tämän omia kirjastoja. Yhteyden kautta pystytään lähettämään pyyntöjä kohdejärjestelmän nykyisestä tilasta. Tästä vastauksena saadusta informaatiosta muodostetaan isompi paketti, joka välitetään yksittäiselle asiakkaalle. Lisäksi kohdejärjestelmä julkaisee aika ajoin muutoksista mitä järjestelmässä tapahtuu, ja näitä päivitysviestejä kuunneltiin palvelinpuolen sovelluksesta. Näistä informaatiosta muodostettiin pienempiä muutoksia, jotka välitettiin eteenpäin kaikille asiakkaille.

Kun asiakassovellus on alustamassa itseään, se tekee palvelulle pyynnön järjestelmän nykyisestä tilasta. Tällöin palvelu vastaavasti tekee pyynnön kohdejärjestelmän koko nykyisestä tilasta, joka muotoillaan ja palautetaan kyselyn tehneelle asiakkaalle. Vastuu tämän tilan ylläpidosta jätettiin asiakkaalle, jolloin kaikki yksittäiset muutokset suoriutuvat

pienikokoisilla muutosviesteillä. Tämä lähestymistapa tehtiin sen pohjalta, että uutta tietoa tuli kohdejärjestelmältä reaaliajassa, ja tämä haluttiin saada päivitettyä kaikille asiakkaille mahdollisimman nopeasti.

SignalR-kirjastolla toteutettiin palvelulle ja asiakkaalle yksinkertaiset rajapinnat, joilla nämä keskustelivat keskenään. Palvelinpuolella rajapinta kokonaisuudessaan on vain pyyntö koko tilasta, kun taas asiakaspuolen rajapinta määriteltiin tarkemmaksi. SignalR-kirjaston toiminnallisuuteen kuuluvat keskitysrakenteet tarjosivat palvelinsovelluksessa tiedot kaikista auki olevista yhteyksistä asiakkaisiin. Tätä ominaisuutta hyödyntämällä tehtiin palvelinsovellukseen viestinvälittäjä, joka vastaanottaa kameroiden ja järjestelmän tilamuutoksia, ja välittää näitä asiakkaille samaa tahtia kuin muutoksia muodostuu. Tällä tavalla tietojen ylläpitäminen saatiin prosessina eristettyä varsinaisesta rajapinnasta, eikä asiakkaan tarvitse jatkuvasti tehdä kyselyjä palvelimelle uuden tiedon toivossa. Keskitysrakenteen käyttäminen tarjosi suoraan tuen usealle yhtäaikaiselle asiakkaalle ilman lisämuutoksia.

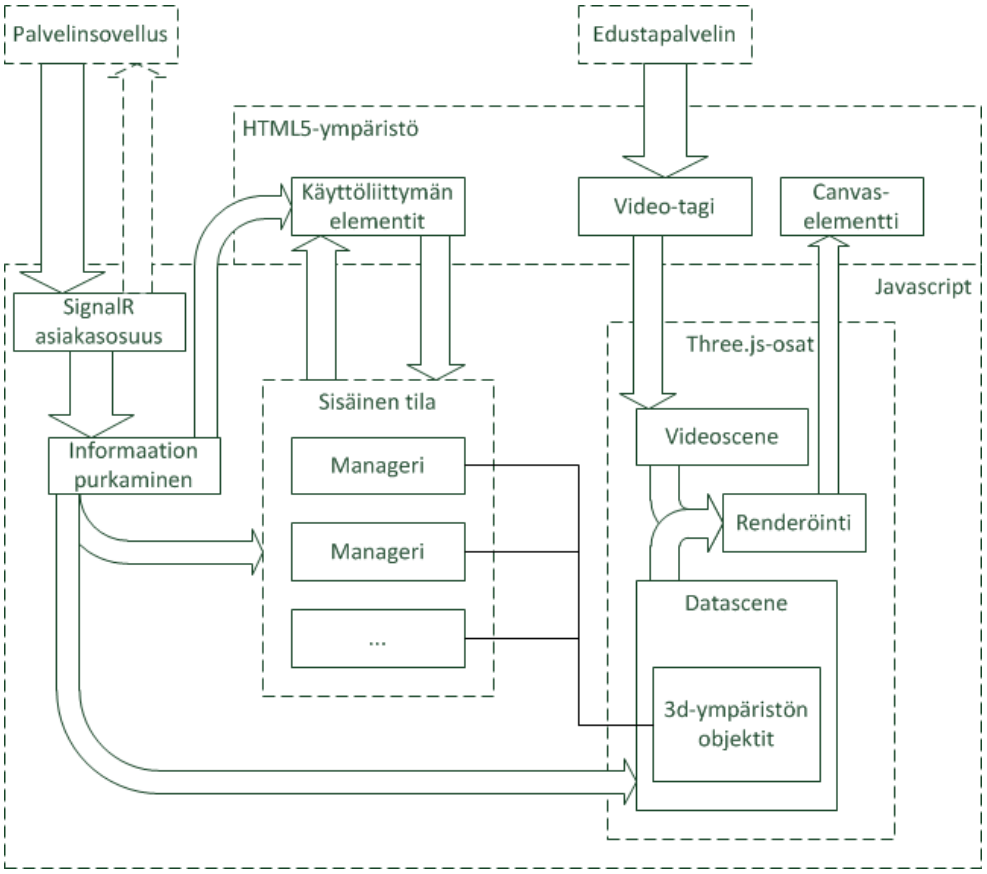
Kuten kuvassa 6.4 on pyritty havainnollistamaan, C# palvelinsovelluksesta on toteutettu erillisenä koodinvaihtosovellus ja edustapalvelin tätä varten. Koodinvaihtosovellus toteutettiin Python-kielellä hyödyntämällä LibVLC-kirjaston python-toteutusta. LibVLC-kirjasto huolehti videokuvan vastaanottamisesta, koodinvaihtamisesta ja suoratoistamisesta uudelleen lokaalisti. Lokaalin suoratoiston lukemista varten rakennettu edustapalvelin toteutettiin Node.js-sovelluksena, hyödyntäen cors-anywhere -kirjastoa. Käytetyt kamerat lähettivät H.264-formaatin RTSP-suoratoistoa, jolle suoritettiin koodinvaihto Theora-formaatin HLS-suoratoistoksi. Käytetty cors-anywhere-kirjasto myös lisäsi tarpeelliset CORS-otsikot HTTP-viesteihin, jotka kulkevat edustapalvelimen kautta.

Koodinvaihtaminen todettiin tarpeelliseksi, sillä tämän projektin toteuttamisen aikaan useimmat selaimista ilman selainlaajennuksia eivät tukeneet RTSP-suoratoiston lukemista HTML5-videona [43]. Vastaavasti lopulliseen formaattiin päädyttiin sillä perusteella, että työn toteutuksen aikana Theora-formaatin videokuvaa voitiin toistaa sekä Firefox että Chrome selaimilla suoraan ilman lisäosia [44]. Itse edustapalvelimen käyttämiseen päädyttiin esiin nousseesta tarpeesta liittää HLS-suoratoistopaketteihin CORS-kentät.

6.4 Asiakaspuolen selainsovelluksen rakenne

Kuvassa 6.5 on havainnollistettu asiakaspään selainsovelluksen rakennetta. Palvelinsovellukselta kuunnellaan päivityksiä, joiden pohjalta päivitetään sisäistä tilaa sekä käyttöliittymän komponentteja. Samaan aikaan edustapalvelimen kautta luetaan videokuvaa kameroista, josta välitetään kuva WebGL-tekstuurina. Videokuvan ja sisäisen tilan pohjalta muodostetaan lisätyn todellisuuden näkymä, joka välitetään Canvas-elementillä käyttäjän näkyville.

Asiakaspuolen verkkosivussa hyödynnettiin paitsi SignalR-kirjaston Javascript-toteutusta palvelinsovelluksen kanssa keskustelemiseen, myös Three.js-kirjastoa WebGL:n käyttämiseen sekä bootstrap-kirjastoa verkkosivun visuaalista ulkonäköä varten.



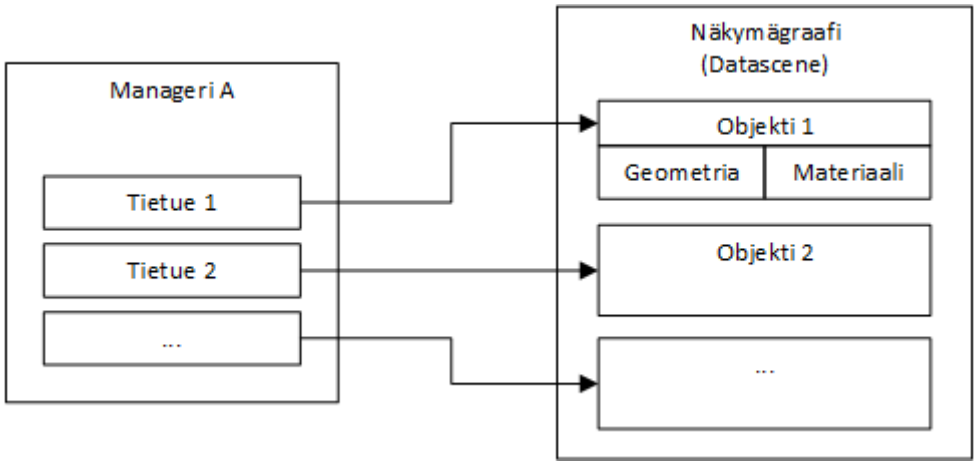
Kuva 6.5 *Yksityiskohtaisempi kuvaus asiakaspään selainsovelluksen toiminnan yksityiskohdista.*

Videokuvan vastaanottamista varten Javascript-koodissa luodaan HTML5-standardin video-tagi. Tämän avulla muodostetaan yhteys edustapalvelimeen, ja sitä kautta aloitetaan Theora-formaatin HLS-suoratoiston vastaanottaminen. Tätä vastaanotettua videota ei näytetä suoraan selaimessa, vaan se annetaan Three.js käytettäväksi WebGL-tekstuurina. WebGL-grafiikkaa varten on määritelty verkkosivun lähdekoodissa canvas-objekti, jolle varsinainen 3D-grafiikan piirtäminen tapahtuu.

Vastaavasti itse C# palvelinsovellukseen muodostetaan yhteys SignalR-rajapinnan avulla, ja kun verkkosivu on latautunut, välitetään tätä kautta palvelinsovellukselle pyyntö kohdejärjestelmän koko sen hetkisestä tilasta. Kun tämä tilatieto on vastaanotettu, muodostetaan siitä lokaalisti ympäristö luomalla jokaista käsiteltävää objektia kohden tietueita muistiin, ja muodostamalla tarvittavat näkymägraafit Three.js-kehystä varten. Käytetyt näkymägraafit ovat kuvassa 6.5 nimetty nimillä Videoscene ja Datascene.

Sisäisen tilan, ja siten lisätyn todellisuuden objektien, ylläpitämistä varten päädyttiin muodostamaan tietueita, joihin vastaanotettu ei-visuaalinen tieto talletettiin. Näillä tietueilla oli aina jokin ID tai nimi, jolla nämä pystyttiin erottelemaan toisistansa. Tämän pohjalta päädyttiin tekemään yksinkertaisia ”manageri”-olioita, joissa saatuja tunnisteita sidottiin talletettuihin arvoihin. Jokaista virtuaalisen maailman objektityyppiä kohden rakennettiin oma manageri, ja tätä kautta käsiteltiin kyseisiä objekteja virtuaalisessa maailmassa.

Managereihin talletetut objektit sisältävät palvelulta saadut tiedot, mm. sijainnin ja mitat, sekä niitä varten luodut kuvanmuodostamisen kannalta tärkeät tiedot, kuten varsinaiset geometriat ja niiden visuaalisen tiedon.



Kuva 6.6. *Havainnollistus ”managerin” ja näkymägraafin rakennetusta yhteydestä. Manageri on hakurakenne, jossa tietue viittaa näkymägraafista löytyvään objektiin.*

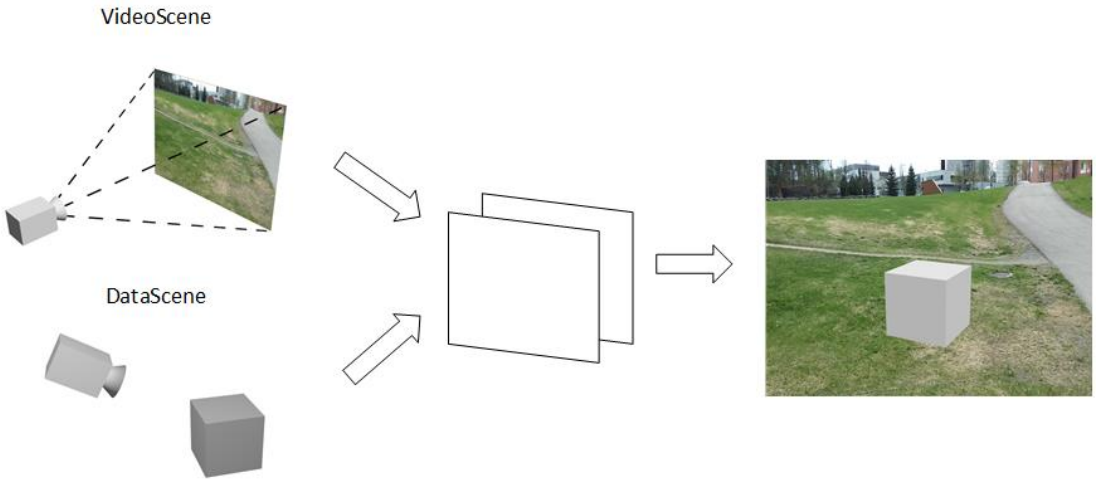
Kuvassa 6.6 on pyritty havainnoimaan managerien yhteyttä näkymägraafiin. Managerin sisältämät informaatiot ovat viitteitä käytetyn näkymägraafin sisältämiin tietoihin, jolloin piirtämisessä ei tarvitse eksplisiittisesti käyttää ja selata managerien sisältöä.

Kun järjestelmältä saapuu muutosviesti, haetaan tätä vastaava objekti siihen liittyvältä managerilta. Viestistä luetaan tiedot, jotka asetetaan haetun objektille uusiksi arvoiksi, ja päivitetään geometrioita sekä muuta informaatiota tarpeen vaatiessa. Muutokset välittyvät suoraan käyttöliittymään.

Kuvan muodostuksessa on käytössä kaksi näkymägraafia: videokuva ja lisätyn todellisuuden virtuaalinen maailma. Videokuvan näkymägraafi (nimetty kuvassa 6.5 Videoscene) sisältää kameran lisäksi vain neliö-geometrian, joka peittää kameran näkökentän täysin. Tämä ei sisällä mitään valaistuksia, ja nelikulmiolle määritellään käyttöön aikaisemmin mainittu video-tagi tekstuuriksi. Tätä näkymägraafia piirtämällä muodostuu käytetylle canvakselle videokuva ilman muutoksia.

Lisätyn todellisuuden näkymägraafiin (Kuvissa 6.5 nimetty nimellä Datascene) on rakennettu geometriaa kohdejärjestelmän tilasta ja ympäristöstä. Käytetty 3D-maailma päätettiin määritellä käyttämään järjestelmän raportoimien koordinaattien nollakohtaa käytetyn lokaalin 3D-avaruuden origona. Järjestelmä raportoi sijainnit reaalimaailman sijainteina käyttäen millimetritarkkuuksia, ja nämä muuntuvat käytettyyn 3D-maailman koordinaateiksi muuttamalla nämä arvot metreiksi. Jokaista objektia kohden on näkymägraafissa määritelty geometriat ja resurssit, joiden avulla nämä piirretään näkyville.

Kuvan luomiseksi piirretään molemmat näkymägraafit järjestyksessä, eli videokuvan päälle piirrettiin 3D-geometriat. Kuva 6.7 havainnollistaa tätä prosessia. Näin muodostettiin efekti, että virtuaalisen maailman objektit näkyisivät videokuvassa päällekkäin. Tuotettu 3D-geometria määriteltiin läpikuultavaksi, jolloin videokuva pääsi näkymään tämän läpi.



Kuva 6.7. *Havainnollistus toteutetusta tavasta yhdistää videokuva ja virtuaalimaailma.*

Palvelun konfiguraatioihin oli kameroiden osalta määritelty sijaintitiedot ja orientaatiopoikkeamat, joiden avulla sijoitettiin kamera 3D-maailmaan. Kuten Luvussa 2 mainittiin, ei haluttu tehdä kuvasta tunnistamista, vaan haluttiin tukeutua ympäristöstä saatavaan informaatioon. Näin ollen päädyttiin käyttämään kameran ilmoittamia arvoja kierrosta, kallistumasta ja suurennuksesta virtuaalisen maailman kameran orientoimiseen.

Koska kameran ilmoittamista arvoista kierto ja kallistuma koskevat kiertoa jonkin akselin ympärille, päädyttiin käyttämään kvaternioita näiden kiertojen kuvaamiseen. Jokaista akselia vasten muodostettiin sitä vastaava kvaternio, ja näistä muodostettiin lopullinen kameran orientaatio. Muodostetut kvaterniot kerrottiin keskenään, muodostaen siten kokonaiskierron kvaternion. Tämä muunnettiin lopuksi kiertomatriisiksi, jota käytettiin kameraobjektin orientoimiseen 3D-maailmassa.

Three.js-kirjasto tarjoaa myös sisäänrakennetusti tuen säteenjäljitykselle yksittäisen näkymägraafin avaruudessa, käyttäen kameraobjektia lähtöpisteenä ja suuntaa kuvan koordinaatteina. Tätä hyödynnettiin etsimään hiiren osoittimen alla oleva objekti 3D-maailmasta, sekä valinnan muodostamisessa. Kun hiiren osoitin on kuvassa jonkin objektin päällä, muodostetaan ensimmäisestä säteelle osuneesta objektista käyttäjälle näkyviin pienenä HTML-ikkunaelementtinä vihjeteksti, jossa on kyseisen objektin nimi ja id. Tämä elementti myös seuraa hiiren osoitinta.

Käyttäjä voi valita objekteja kuvasta klikkaamalla hiiren painikkeilla. Valinnan muodostamisessa jokaista säteenjäljityksen säteelle osunutta objektia kohden haettiin manageireista vastaavat objektit, joille asetettiin materiaalin väri näkyväksi. Tämän seurauksena valitut objektit ilmestyivät kuvaan. Valinnasta pidettiin yllä edellisen valinnan objektiviitteet. Kun valinta muuttuu, edellisestä valinnasta jokaista objektia päivitettiin vastaavasti muuttamalla materiaalin väri läpinäkyväksi, piilottaen nämä. Kun objekteja oli valittuna, muodostetaan kelluva HTML-elementti, johon on koostettu valinnan objekteista informaatiota. Tätä kelluvaa elementtiä päivitettiin reaaliajassa aina kun uutta informaatiota saatiin palvelimelta.

Edellä mainittuja käyttöliittymäelementtejä varten määriteltiin omat Javascript-objektit, jotka määrittelevät miten ne muodostetaan HTML-elementteinä näkyville. Kun käyttäjä liikuttaa hiirtä, säteenjäljityksen lisäksi päivitetään kumpaakin luokkaa, jotta vihjetekstin ikkuna ja informaatioikkuna päivittyvät. Informaatioikkunalle on myös määritelty toiminnallisuutta, jotta käyttäjä voi siirrellä tätä, joskaan tämä ikkunan siirtely ei toiminut kaikissa selaimissa työn toteutushetkellä.

6.5 Kameroiden sovittaminen

Jotta virtuaalitodellisuuden geometriat täsmäisivät videokuvan maailmaan, täytyi tehdä virtuaalitodellisuuden kameran sovittamista reaalimaailman kameroihin. Tällä tavalla saadaan virtuaalisen kameran ja reaalisen kameran katselukulma ja perspektiivi vastaamaan toisiansa, jolloin Kuvan 6.7 tapaisesti lisätyt virtuaaliset objektit sopisivat paremmin reaalimaailmaan. On eri tapoja toteuttaa kameroiden sovittaminen, ja osaa näistä on jo esitelty luvussa 3.4.

Tässä työssä käytössä olleiden kameroiden sijainnin ja tilan mallinnus sisältää useita vapausasteita, joiden läpikotaiseen tutkimiseen ei käytetty enempää aikaa kuin prototyyppitason mielessä nähtiin tarpeelliseksi. Näitä vapausasteita olivat sijainnin, suurennuksen ja kierron epätarkkuudet sekä kameroiden epätarkka asennus. Käytössä olleiden kameroiden sijainnista oli tarjolla vain rajallisesti informaatiota, ja kameroiden ilmoittamat suurennuksen arvot käyttivät mielivaltaista asteikkoa. Kameroiden asentamisessa oli poikkeamia ja virheitä, minkä lisäksi kameroiden asennussuunta ei ollut tunnettu ja siten kameroiden kierron nollakohtaa ei tunnettu.

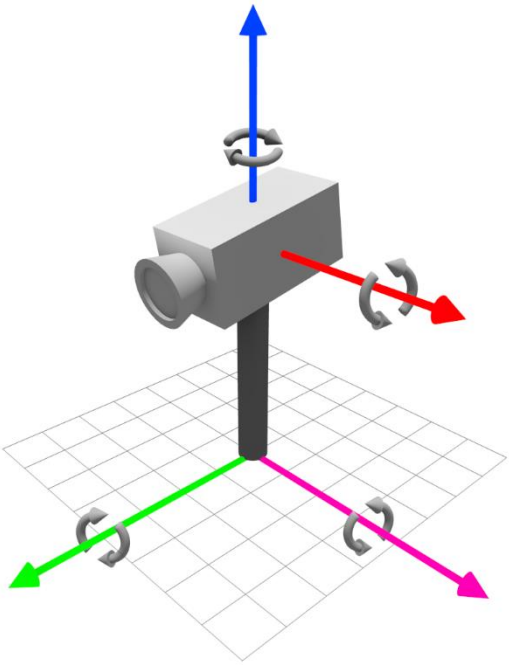
Koska kohdejärjestelmä sijaitsi tasaisella alueella, jossa oli tunnettuja maamerkkejä sekä tunnettuja suoria viivoja maastossa, arvioitiin vapausasteita sovittamalla niitä tunnettuun tietoon ja videokuvaan. Koska 3D-maailman tarkoitus oli sopia tähän maailmaan, arvioinnissa ja hienosäädössä hyödynnettiin lisättyä todellisuutta käyttämällä yksinkertaista rautalankamallia tasosta ja maamerkeistä, jonka reunoja ja sijaintia sovitettiin näkyvään maastoon.

Kamerat noudattivat valmistajan käyttämää asteikkoa suurennuksen arvojen raportoimiseen, joka koostui numeerisesta alueesta 1-9999. Laitteiden datalehdissä oli määritelty käytössä olevat optisen suurennuksen näkökentän leveyden kulmat asteina, mutta ei tämän alueen asettautumista raportoituihin arvoihin. Näkökentän leveyden lisäksi kameroiden suurennuksen kertoimet asteikon alueella oli ilmoitettu. Datalehdissä oli ainoastaan näkökentän leveyden kulma. Tämän seurauksena kameran ilmoittamasta suurennuksen arvosta laskettu näkökentän leveys täytyi vielä muuntaa näkökentän korkeuden kulmaksi, jotta sitä voitaisiin käyttää Three.js-ympäristössä.

Koska suurennuksen asteikko oli osin tuntematon, arvioitiin aluksi muita arvoja siten, että FOV-arvo oli sidottu maksimiin, toisin sanoen näkökentän leveys oli sidottuna maksimiin. Tämä asettautui käytetyllä asteikolla sen pienimpään arvoon. Kun muita vapausasteita oli saatu arvioitua, lähdettiin arvioimaan suurennuksen arvojen ja FOV-arvojen välistä muunnosta sovittamalla ja muodostamalla arvoista pareja.

Käsin arvioiduista arvoista pyrittiin aluksi muodostamaan arvioivaa funktiota, mutta virheiden puitteissa mitään suoraan sopeutuvaa funktiota ei saatu muodostettua. Virhettä muodostui erityisesti suurilla suurennuksen arvoilla, jolloin pienikin näkökentän leveyden muutos näkyi isoina harppauksina asteikossa. Lopulta ajan puitteissa päädyttiin hyödyntämään jo ennalta sovitettuja arvoja, lineaarisesti interpoloiden näiden arvojen välillä. Tällä päästiin tyydyttävään tulokseen.

Kuten aiemmin on mainittu, 3D-maailman kameraobjektin orientointi suoritettiin muodostamalla kvaternio kierrosta. Kvaternio muodostettiin yhdistämällä eri akselien suhteen tehdyt pyöritykset oikeassa järjestyksessä, kuten Kuvassa 6.8 on havainnollistettu. Kameran kierrosta määriteltiin kameran pystysuoran akselin suuntainen kierto, ja kallistumasta kameran katsomissuuntaan ja kiertosuuntaan kohtisuorassa akselissa. Käytetyissä kameroissa ilmeni, että näiden kierron nollakohta vaihteli riippuen kamerasta, jolloin käytettyä kierron arvoa täytyi lisätä tai vähentää riippuen käytetystä kamerasta.



Kuva 6.8. *Kameran orientaatioon vaikuttaneiden kiertojen havainnollistus. Kameran sisällä on kierto ja kallistus, ja alla oleva taso kuvastaa asennustasoa.*

Koska käytössä olleet kamerat eivät olleet ideaalisia, ilmeni niiden asennossa asennuksen epätarkkuudesta johtuvaa virhettä. Tätä virhettä arvioitiin määrittelemällä kaksi kameran kiertoakselin kanssa kohtisuoraa ja keskenään ortogonaalia vektoria, joiden ympäri tehtiin kiertoa. Peruste tälle oli se, että näillä määriteltäisiin vino taso, mikä mukailisi oikeaa asennustasoa kameralle. Tätä asennustason virhettä on havainnollistettu kuvassa 6.8 käyttämällä tasoa ja tälle määriteltyjä kiertoakseleita. Lopulta muodostettiin siis neljä kvaternioa, joista yhdistettiin lopullinen kokonaiskierto.

6.6 Muita havaintoja

Kokemattomuus, käytössä ollut aika ja suuri määrä vapausasteita kameroiden informaatiossa tuotti hankaluuksia lisätyn todellisuuden sovittamisessa videokuvaan. Mitään yleisiä menetelmiä tätä varten ei varsinaisesti hyödynnetty, vaan käytettiin käsin sovittamista ja kameran tiedottamia arvoja. Lopputulos jäi joiltain osin epätarkaksi, ja virhettä ilmeni, kun kameraa liikuteltiin tietyin tavoin. Näkökentän leveydelle tehdyssä arvioinnissa ilmeni myös epätarkkuuksia.

Kameroiden kuvassa ilmeni myös potentiaalista radiaalista vääristymää. Tämä näkyi siten, että kuvan reunoilla 3D-maailma ei välttämättä ollut yhtenäinen videokuvan kanssa. Tämän vääristymän korjaamiseen on olemassa erilaisia tekniikoita ja tapoja, mutta tämä ongelma ilmeni loppuvaiheilla projektia, eikä sitä ei ehditty korjaamaan.

Alkuvaiheessa, kun kameran videokuvaa yritettiin saada käyttöön asti, ilmeni WebGL:n tiukahko piirre CORS-datan suhteen: mikäli käytettyä resurssia ei ole merkitty oikeellisesti CORS-kentillä, WebGL ei tätä suostu käsittelemään vaan aiheuttaa virheen. Tämän kiertämiseen olisi voinut olla parempiakin ratkaisuja kuin edustapalvelin ja LibVLC, mutta käytettävissä olleen ajan puitteissa päädyttiin tähän ratkaisuun.

Koodinvaihdossa havaittiin lisäksi harmillinen piirre epäsynkronoitumisen muodossa: pidemmällä ajalla videokuvan ja reaalimaailman välinen ero alkoi ilmetä lisätyssä todellisuudessa. Koska lisättyä todellisuutta päivitettiin reaaliajassa samaa tahtia mitä informaatiota saatiin palvelulta, lisätty todellisuus oli aina videokuvaa edellä. Tätä heikensi vielä se, että käytetty koodinvaihto ja uudelleenlähetys vaikuttivat hitaasti kasvattavan tätä väliä datan ja videokuvan välillä. Käytetyn ajan puitteissa tähän ei löydetty ratkaisua muutoin, kuin tarjoamalla käyttäjälle painike videokuvan uudelleen lataamiseen synkronoiden sitä samalla.

7. LOPPUTULOKSESTA

Lopputulos toteutuksessa oli järjestelmä, jolla kameran videokuvassa näkyvien laitteiden ja objektien ympärille muodostettiin geometriaa, joka tuli käyttäjän näkyville kyseisen objektin valitsemisen yhteydessä. Lisäksi valituista objekteista muodostui käyttöliittymään informaatioikkuna, jossa näistä haetut tiedot koostettiin ja päivitettiin reaaliajassa. Vaikka järjestelmään jäikin epätäydellisyyksiä, kokonaisuutena lopputulos kykeni asemoimaan virtuaalisia objekteja videokuvaan. Nämä objektit noudattelivat videokuvassa näkyvää perspektiiviä, ja lisätyt objektit vastasivat yleisesti kuvassa näkyvien fyysisten objektien sijaintia.

Koska toteutettu järjestelmä tehtiin prototyyppi ja ”proof-of-concept” henkisesti, lopputulos jäi suurimmilta osin yksinkertaiseksi. Mitään yleisiä menetelmiä lisätyn todellisuuden toteuttamiseen ei käytetty suoraan, koska niiden hyödyntäminen ei ollut välttämättä edes mahdollista ympäristön vapausasteiden määrän ja tietojen puutteen varjossa. Käyttöliittymän ja lisätyn todellisuuden visualisoinnin toteutus jäi hyvin yksinkertaiseksi. Lopputulos kykeni silti täyttämään siltä vaaditut ominaisuudet ja toiminnallisuuden.

Prototyyppihenkisestä lähestymistavasta johtuen järjestelmästä löytyy paljolti paranneltavaa, mikäli tästä lähdettäisiin kehittämään oikeaa tuotetta. Isoimpia jäljelle jääneitä parannuksen kohteita olivat epätarkkuuksien ja virheiden arvioiminen. Virheiden arviointiin ja niiden huomioimiseen itse kameran sovittamisessa täytyisi käyttää enemmän aikaa. Prototyyppiä kehitettäessä vapausasteita ja informaation puutteita oli paljon, joka hankaloitti projektin toteuttamista. Jatkokehityksessä tämä saattaisi helpottua, mikäli otetaan käyttöön varta vasten asennettua laitteistoa ja tunnetaan laitteistojen sekä ympäristön informaatiota tarkemmin.

Selaimen sovellus ei ottanut mitään kantaa videokuvan ja lisätyn todellisuuden keskinäisiin oikeellisuuksiin. Sovellus oletti, että konfiguraatiotiedostoihin asetetut informaatiot, poikkeamat ja virheet täsmäisivät siten, että lisätty todellisuus saataisiin luotua oikeellisesti. Lopputuloksessa virtuaalisten objektien sijainti videokuvassa poikkesi osittain odotetuista. Jääneiden virheiden korjaamiseen ei ehditty käyttämään valtavasti aikaa toteutuksen ollessa ”proof-of-concept” tyyppinen.

Videokuvaa hyödynnettiin tässä toteutuksessa manuaalisesti, mutta tällöin arviot jäivät silmämääräisiksi ja siten osaltansa epätarkoiksi. Automaattista sovitusta varten pystyisi tekemään toteutetun koodinvaihtosovelluksen tapaisesti erillisen sovelluksen, joka tekisi kuvan pohjalta arvioita sovituksen arvoista. Tätä voisivat helpottaa kuviot kentällä tai tarkasti määritellyt ja kuvasta tunnistettavat maamerkit.

Vaikka tässä prototyypissä päädyttiinkin olla pitämättä sisäistä tilaa itse palvelinpuolella, voisi tätä vaihtoehtoa harkita informaation prosessoinnin kannalta. Reaaliaikaisesti saadusta informaatiosta voisi myös koostaa informaatiota selainkäyttöliittymälle välitettäväksi, esimerkiksi listaus kaikista liikkuvista laitteista tai jatkuvasti päivittyvä listaus tapahtumista kentällä. Tällä hetkellä käytettiin järjestelmän kertomasta informaatiosta vain osa, ja niistäkin vain nykyhetkiset tiedot.

Toteutuksessa käytetty SignalR-keskitysrakenne mahdollisti usean yhtäaikaisen käyttäjän palvelemisen palvelinsovelluksessa ainakin teoriassa. Tätä ei kuitenkaan aikarajoitteiden puitteissa koskaan testattu. Usean asiakkaan tukeminen ei vaatisi suuria muutoksia itse sovelluksien osalta, mutta ongelmaksi voi tällöin voi muodostua videokuvan suoratoiston vaatima verkkoyhteyden nopeus. Toteutettu järjestelmä koestettiin yhdellä tietokoneella, jolloin ei videokuvan suoratoistossa tapahtunut muita viiveitä kuin kamerayhteydestä ja koodinvaihdosta aiheutuneet viiveet. Näiden viiveiden ja verkkoyhteyden vaatimukset ovat myös asia, joka täytyisi ottaa jatkokehityksessä huomioon.

Kameroiden käyttämä videoformaatti rajoitti niiden käyttämistä ilman koodinvaihtosovellusta välissä. Koodinvaihdon toteutus jäi jokseenkin irralliseksi osaksi palvelinpuolta, ja tätä voisi jatkossa yhdistää jollain tapaa enemmän itse palvelinsovellukseen. LibVLC:stä on olemassa myös C#-kirjastototeutus, jolla sitä saisi ehkä toteutettua osana itse palvelua. Tämän projektin puitteissa tarve saatiin jo nykyisellä ratkaisulla täytettyä, eikä tähän käytetty sen enempää aikaa. Tulevaisuudessa, jos kamerat olisivat varta vasten projektia varten valittuja, voisi videoformaatissa olla enemmän valinnanvaraa tai voisi koodinvaihdon ehkä jopa toteuttaa itse kameroissa.

WebGL:n hyödyntäminen jäi prototyypissä yleisesti ottaen yksinkertaiseksi. Valittu tapa piirtää videokuva ja sen päälle piirretty lisätty todellisuus hankaloitti osittain havaitun radiaalisen vääristymän korjaamista. Jatkossa voisi tämän toteuttamista jatkaa ja korjata vääristymää potentiaalisesti ennen piirtämistä esimerkiksi verteksisävyttimen tai fragmenttisävyttimen avulla. Tämän tekstin kirjoittamisen aikana julkaistiin WebGL 2.0 -spesifikaatio, joka vastaa OpenGL ES 3.0 API-version spesifikaatiota. Tämän työn kannalta tästä ei välttämättä olisi ollut hyötyä, muuten kuin toteutuksen potentiaalisen yksinkertaistamisen kautta.

Aikarajoituksista johtuen joitain ajatuksia jäi toteuttamatta, ja prototyypin kehittämisenkin jälkeen on noussut visioita, miten toteutusta olisi voinut vielä laajentaa. Edellä mainitun sisäisen tilan säilytyksen ja lisäinformaation prosessoinnin kautta olisi voinut mahdollisesti tarjota esimerkiksi laitteiden kulkureittejä kentällä, tai voinut visualisoida tarkemmin koneiden tiloja. Lisätyn todellisuuden lisäksi toteutuksesta voisi johtaa virtuaalitodellisuuden sovelluksen, jossa käyttäjä voisi liikkua ja tarkastella virtuaalisessa maailmassa järjestelmän tilaa.

Tällä hetkellä prototyypin tulevaisuudesta ei ole muuta tietoa, kuin että siitä ei tulla sellaisenaan tekemään mitään varsinaista lopputuotetta. Mikäli toteutusta lähdettäisiin jalostamaan eteenpäin, ympäristö ja käyttötapa muuttuvat suuresti verraten toteutetussa prototyypissä nyt käytettyihin. Toteutettu prototyyppijärjestelmä kuitenkin toi esiin potentiaalisen hyödyn vastaavan järjestelmän toteuttamisesta.

8. YHTEENVETO

Tässä työssä toteutettiin kokonaisuus, jossa asiakas voi selaimen kautta nähdä videokameran kuvaa ehostettuna lisätyn todellisuuden virtuaalisilla objekteilla. Lisätyn todellisuuden virtuaalisten objektien tiedot pyydettiin ja vastaanotettiin kohdejärjestelmästä, ja näin muodostetut objektit sijoitettiin kameroiden videokuvaan hyödyntämällä tietoa kohdealueelle asennetuista kameroista. Käyttäjä pystyi valitsemaan videossa esiintyviä objekteja, ja tarkastelemaan sitä kautta niiden tietoja, joita päivitettiin reaaliajassa.

Toteutettua kokonaisuutta tehtiin proof-of-concept –prototyyppinä, joka jäi hyvin yksinkertaiseksi ominaisuuksiltaan. Toteutus kokonaisuudessaan koostui asiakassovelluksesta ja palvelinsovelluksesta, joista palvelinsovellus välitti kaikille asiakkaille informaatiota reaaliajassa. Lisäksi videokuvan saattaminen asiakkaalle asti vaati lisäsovelluksia palvelimen tasolle.

Palvelinpuolen sovelluksissa hyödynnettiin useita kieliä ja sovelluskehyksiä. Palvelinsovellus oli C#-sovellus ja videokuvaa varten käytettiin Node.js- ja Python-sovelluksia. Asiakkaan selainpään sovelluksessa käytettiin Javascript-kieltä, ja hyödynnettiin SignalR-kirjastoa palvelun kanssa keskustelemiseen sekä Three.js-ohjelmistokehystä WebGL-grafiikan tuottamiseen.

Ympäristön vapausasteiden ja asetettujen rajoitusten puitteissa mitään suoria yleisesti käytettyjä menetelmiä lisätyn todellisuuden toteuttamiseksi sovittamiseksi kuvaan ei käytetty, vaan näiden sijaan hyödynnettiin ennalta tiedettyjä lukuarvoja ja tietoja esim. kamerojen sijainneista. Koska nämä tiedot eivät olleet täydellisesti tiedossa, tehtiin manuaalista hienosäätöä ja arviointia puuttuvista vapausasteista. Hienosäätäminen ja arviointien virheiden minimoiminen jäivät osittain vajaaksi.

Epätarkkuuksista ja esille nousseista ongelmista huolimatta toteutettu järjestelmä täytti siltä vaaditut toiminnallisuudet. Tehdystä prototyypistä ei todennäköisesti tulla jatkamaan mitään lopputuotetta sellaisenaan, mutta tämä voi tarjota pohjan eri tyyppiselle jatkoprojektille.

LÄHTEET

[1] Milgram P., Kishino F., A taxonomy of mixed reality visual displays, IEICE Transactions on Information Systems, Vol E77-D, No.12 December 1994, 15 p. Saatavissa: https://cs.gmu.edu/~zduric/cs499/Readings/r76JBo-Milgram_IEICE_1994.pdf

[2] Azuma R., A survey of augmented reality, MIT Press, Presence: Teleoperators and Virtual Environments, Volume 6 Issue 4, August 1997, pp. 355-385, Saatavissa: http://ronaldazuma.com/papers/ARpresence.pdf

[3] Carmigniani J., Furht B., Augmented Reality: An Overview, Springer, Handbook of Augmented Reality, 2011, pp. 1-46, Saatavissa: http://link.springer.com/book/10.1007/978-1-4614-0064-6

[4] Mountney P., Fallert J., Nicolau S., Soler L., Mewes P.W., An Augmented Reality Framework for Soft Tissue Surgery, Springer, Cham, Medical Image Computing and Computer-Assisted Intervention – MICCAI 2014. Lecture Notes in Computer Science, vol 8673. 2014, pp. 423-431. Saatavissa: http://link.springer.com/chapter/10.1007/978-3-319-10404-1_53

[5] Botella C., Bretón-López J., Quero S., Baños R., García-Palacios A., Treating Cockroach Phobia With Augmented Reality, Elsevier ltd, Behavior Therapy 41, 2010, pp. 401-413, Saatavissa: http://www.sciencedirect.com/science/article/pii/S0005789410000389

[6] Kancherla A., J. Rolland, D. Wright, G. Burdea, A Novel Virtual Reality Tool for Teaching Dynamic 3D Anatomy, Springer, Berling, Heidelberg, Computer Vision, Virtual Reality and Robotics in Medicine, 1995, pp. 163-169, Saatavissa: http://link.springer.com/chapter/10.1007/978-3-540-49197-2_18

[7] Nolle S., Klinker G., Augmented reality as a comparison tool in automotive industry, IEEE, 2006 IEEE/ACM International Symposium on Mixed and Augmented Reality, 2006, pp. 249-250 Saatavissa: http://ieeexplore.ieee.org/document/4079291/

[8] Rosenberg L., The Use of Virtual Fixtures as Perceptual Overlays to Enhance Operator Performance in Remote Environments, STANFORD UNIV CA CENTER FOR DESIGN RESEARCH, 1992, p. 52, Saatavissa: http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA292450

[9] Sood R., Pro Android Augmented Reality, Apress, 2012, 329 p. Saatavissa: http://link.springer.com/book/10.1007/978-1-4302-3946-8

[10] Siltanen S., Theory and applications of marker-based augmented reality, VTT, 2012, 241 p., Saatavissa: http://www.vtt.fi/inf/pdf/science/2012/S3.pdf

[11] Augmented reality history, background and philosophy, Macquarie University, Saatavissa (viitattu 23.5.2017): https://wiki.mq.edu.au/display/ar/Augmented+reality+history,+background+and+philosophy

[12] Inventor in the field of virtual reality, Saatavissa (viitattu 23.5.2017): http://www.mortonheilig.com/InventorVR.html

[13] Feiner S., Macintyre B., Seligmann D., Knowledge-based augmented reality, ACM, Communications of the ACM - Special issue on computer augmented environments: back to the real world, Volume 36 Issue 7, July 1993, pp. 53-62, Saatavissa: http://dl.acm.org/citation.cfm?id=159587

[14] Puhakka A., 3D-grafiikka, Talentum Media, 2008, 453 s. ISBN: 9778-952-14-1192-2

[15] 1.5.2. Homogeneous Coordinates, Cartography for Swiss Higher Education, 2012, Saatavissa (viitattu: 23.5.2017): http://www.e-cartouche.ch/content_reg/cartouche/graphics/en/html/Transform_learningObject2.html

[16] Song H.A., OpenGL Projection Matrix, 2012, Saatavissa (viitattu: 23.5.2017): http://www.songho.ca/opengl/gl_projectionmatrix.html

[17] Sellers G., Wright R., Haemel N., OpenGL SuperBible, Pearson, 2013, 848 p. ISBN: 9780321902948

[18] Shoemake K., Animating rotation with quaternion curves, ACM, ACM SIGGRAPH Computer Graphics, Volume 19 Issue 3, July 1985, pp. 245-254, Saatavissa: http://dl.acm.org/citation.cfm?id=325242

[19] Using Quaterniont to Perform 3D rotations, Saatavilla (viitattu: 23.5.2017): http://www.cprogramming.com/tutorial/3d/quaternions.html

[20] Scabia M., Working with Stage3D and perspective projection, Adobe Systems Incorporated, 15.11.2011, Saatavissa (viitattu:23.5.2017): http://www.adobe.com/devnet/flashplayer/articles/perspective-projection.html

[21] PerspectiveCamera, Saatavissa (viitattu: 23.5.2017): https://threejs.org/docs/index.html#api/cameras/PerspectiveCamera

[22] WebGL - OpenGL ES for the Web, Khronos Group, Saatavissa (viitattu 23.5.2017): https://www.khronos.org/webgl/

[23] Hirshon J., Khronos Group Announces Key Advances in OpenGL Ecosystem, Khronos Group, Saatavissa (viitattu 23.5.2017): https://www.khronos.org/news/press/khronos-group-announces-key-advances-in-opengl-ecosystem

[24] Trevett N., Khronos Expands Scope of 3D Open Standard Ecosystem, Khronos Group, 10.8.2015, Saatavissa (viitattu: 23.5.2017): https://www.khronos.org/news/press/khronos-expands-scope-of-3d-open-standard-ecosystem

[25] Crabb A., Khronos Releases Final WebGL 1.0 Specification to Bring Accelerated 3D Graphics to the Web without Plug-ins, Khronos Group, 3.3.2017, Saatavissa (viitattu: 22.52017): https://www.khronos.org/news/press/khronos-releases-final-webgl-1.0-specification

[26] Crabb A., Khronos Reveals API Updates & New Working Groups at GDC, Khronos Group, 27.2.2017, Saatavissa (viitattu: 22.52017): https://www.khronos.org/news/press/khronos-reveals-api-updates-new-workgroups-at-gdc

[27] Munshi A., Leech J., OpenGL ES Common Profile Specification Version 2.0.25 (Full Specification) (November 2, 2010), Khronos Group, 2010, Saatavissa (viitattu 23.5.2017): https://www.khronos.org/registry/OpenGL/specs/es/2.0/es_full_spec_2.0.pdf

[28] Using CORS to load WebGL textures from cross-domain images, Mozilla, 2011, Saatavissa (viitattu: 23.5.2017): https://hacks.mozilla.org/2011/11/using-cors-to-load-webgl-textures-from-cross-domain-images/

[29] Cross-Origin Resource Sharing, W3C, 2014, Saatavissa (viitattu: 23.5.2017): https://www.w3.org/TR/2014/REC-cors-20140116/

[30] WebGL and OpenGL Differences, Saatavissa (viitattu: 23.5.2017): https://www.khronos.org/webgl/wiki/WebGL_and_OpenGL_Differences

[31] Profenza G., About three.js, Saatavissa (viitattu: 23.5.2017): https://stackoverflow.com/tags/three.js/info

[32] Creating a scene, Saatavissa (viitattu: 23.5.2017): https://threejs.org/docs/index.html#manual/introduction/Creating-a-scene

[33] Barros L., Open Scene Graph: The Basics, 2010, Saatavissa (viitattu: 23.5.2017): https://stackedboxes.org/2010/05/05/osg-part-1-the-basics/

[34] Babylon.js, Saatavissa (viitattu: 23.5.2017): https://www.babylonjs.com/

[35] Creating a Basic Scene, Babylon.js, Saatavissa (viitattu: 23.5.2017): http://doc.babylonjs.com/tutorials/creating_a_basic_scene

[36] N. Gebhardt, CopperLicht Tutorial: Hello World, Ambiera, 2011, Saatavissa (viitattu: 23.5.2017): http://www.ambiera.com/copperlicht/documentation/tutorials/tutorial-01.html

[37] CopperLicht Frequently Asked Questions, Ambiera, Saatavissa (viitattu: 23.5.2017): http://www.ambiera.com/copperlicht/faq.html

[38] Getting started with WebGL development, Unity Technologies, Saatavissa (viitattu: 23.5.2017): https://docs.unity3d.com/Manual/webgl-gettingstarted.html

[39] Haas J., A History of the Unity Game Engine, An Interactive Qualifying Project, Worchester Polytechnic Institute, 2014, Saatavissa (viitattu: 23.5.2017): https://web.wpi.edu/Pubs/E-project/Available/E-project-030614-143124/unrestricted/Haas_IQP_Final.pdf

[40] Learn About ASP.NET Web API, Microsoft, Saatavissa (viitattu: 23.5.2017): https://www.asp.net/web-api

[41] Microsoft ASP.NET Web API 2.2 OWIN Self Host, Nuget.org, Saatavissa (viitattu: 23.5.2017): https://www.nuget.org/packages/Microsoft.AspNet.WebApi.OwinSelfHost/

[42] ASP.NET SignalR, Saatavissa (viitattu: 23.5.2017): http://signalr.net/

[43] Mozilla Developer Network and individual contributors, Live streaming web audio and video, Saatavissa (viitattu: 23.5.2017): https://developer.mozilla.org/en-US/Apps/Fundamentals/Audio_and_video_delivery/Live_streaming_web_audio_and_video

[44] Mozilla Developer Network and individual contributors, Media formats supported by the HTML audio and video elements, Saatavissa (viitattu: 23.5.2017): https://developer.mozilla.org/en-US/docs/Web/HTML/Supported_media_formats