TAMPEREEN TEKNILLINEN YLIOPISTO

TOMI LAMMINSAARI

3D-MAAILMAN KAMERAN OHJAAMINEN KASVOJEN PAIKANNUKSEN AVULLA

Diplomityö

Tarkastaja: Tommi Mikkonen
Aihe, tarkastaja ja kieli hyväksytty
Tieto- ja sähkötekniikan tiedekunnan
tiedekuntaneuvoston kokouksessa
9.5.2012

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
LAMMINSAARI, TOMI: 3D-maailman kameran ohjaaminen kasvojen paikannuksen avulla
Diplomityö, 47 sivua
Joulukuu 2012
Pääaine: Ohjelmistotuotanto
Tarkastaja: professori Tommi Mikkonen
Avainsanat: 3D, OpenCV, HAAR

Eleiden hyödyntäminen sovellusten käyttöliittymissä on yleistynyt viime vuosina kosketusnäyttöjen ansiosta. Pelikonsoleille on jo usean vuoden ajan ollut tarjolla peliohjaimia, joita ohjataan painikkeiden lisäksi peliohjainta heiluttamalla ja kallistelemalla. Myös käyttäjän vartalon liikkeiden ja asennon tunnistamista käytetään pelikäytössä paljon. Kolmiulotteista grafiikkaa hyödynnetään tiedon visualisoinnissa ja käyttöliittymissä aiempaa enemmän ja tämä lisää tarvetta uusille tavoille tunnistaa käyttäjän tekemiä eleitä.

Tässä diplomityössä tutkittiin menetelmiä käyttäjän pään sijainnin paikantamiseen web-kameran avulla ja kuinka tämän sijaintitiedon avulla näytöllä esitettävää 3D-näkymän kuvakulmaa voidaan muuttaa vastaamaan käyttäjän pään liikkeitä. Tavoitteena oli toteuttaa aliohjelmakirjasto, jonka avulla pään paikannus on helposti hyödynnettävissä 3D-sovelluksissa.

Kamerakontrollerin toteutuksessa käytettiin avoimen lähdekoodin OpenCV-kirjastoa, joka tarjoaa kasvojen etsimiseen soveltuvan hahmontunnistusalgoritmin ja suuren määrän toimintoja ihonväristen aluiden rajaamiseksi web-kameran kuvasta. Tässä diplomityössä ajetaan suorituskykytestit sekä hahmontunnistusalgoritmin että värirajauksen suorituskyvylle ja näiden testien perusteella kamerakontrollerin toteutustavaksi valitaan menetelmä, jossa hyödynnetään molempien hyviä puolia. Halutessa paikantaa kasvojen etäisyyttä kamerasta oletetaan, että käyttäjän päähän on kiinnitetty helposti jäljitettävä merkki, joka antaa tarvittavan tiedon etäisyyden laskemiseksi.

Toteutettu kamerakontrolleri yltää hyvässä valaistuksessa halvan kameran ja halvan tietokoneen kanssa noin 15:een kasvojen sijainnin paikannukseen sekunnissa. Heikossa valaistuksessa suorituskyky laskee noin puoleen. Epätasaisen tai muuttuvan valaistuksen kanssa kamerakontrollerin suorituskyky ja paikannustarkkuus heikkenevät huomattavasti ja ajoittain se jopa hukkaa käyttäjän pään sijainnin. Kontrolloiduissa olosuhteissa kamerakontrollerin toimintatarkkuus on tyydyttävä.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Degree programme in Information technology
LAMMINSAARI, TOMI: Controlling the camera of 3D world by using real time face tracking
Master's thesis, 47 pages
December 2012
Major: Software engineering
Examiner: professor Tommi Mikkonen
Keywords: 3D, OpenCV, HAAR

Gestures have become very common elements of modern user interfaces as the touch screens have gained popularity in consumer gadgets. At the same time the importance of physical buttons have diminished. For example the latest gaming consoles have controllers that can detect how players move and hold the controllers and this motion is significant part of the gameplay. Another technology that is coming more and more common is 3D graphics. Many displays provide true 3D view already and these will increase the demand for new ways to detect gestures.

This master's thesis studies different methods to track user's head position with web camera. Located head position works as an input that changes the viewing angle of the current 3D scene. The target was to implement a class library that can be used to introduce head tracking features to existing 3D applications.

The primary tool for this study was an open source software library called OpenCV. It provides effective object detection algorithms and image color space filtering functionalities that were used in face detection. This thesis studies both the object detection and the skin color based face tracking methods. Performance tests were executed for both methods. Based on those results a hybrid solution was created for camera controller's actual implementation.

Implemented camera controller can provide approximately 15 head positions per second with low-end camera and low-end computer equipment but this requires good and stable lighting conditions. In average lighting conditions the performance of camera controller drops by half. In uneven or fluctuating lighting conditions the camera controller might fail to track user's head. Under controlled conditions, the camera controller provides quite accurate positioning for user's head.

ALKUSANAT

Tämän diplomityön kirjoittaminen tuntui hyvältä alusta loppuun asti. Työn aihe oli toki itse valittuna kiinnostava, mutta kaikkein eniten sen kirjoittamisessa tyydytti se, että tiesin sen olevan viimeinen tarvittava opintosuoritus ennen valmistumista osittain liiankin pitkäksi venähtäneen teekkariajan päätteeksi. Vaimoni ja lapseni ansaitsevat kiitokset tuestaan tämän työn valmistumiseksi. Erittäin usein he keksivät viikonloppuiltapäivinä sellaista tekemistä, että minä sain rauhassa keskittyä diplomityön tekemiseen.

Myös työn tarkastajana olleelle professori Tommi Mikkoselle haluan sanoa kiitokset, koska hänen ansiostaan työn rakenne ja kirjoitusasu parantuivat huomattavasti.

Tampereella 11.11.2012
Tomi Lamminsaari

SISÄLLYS

1 JOHDANTO

Tietokoneiden ohjaaminen erilaisten eleiden avulla on yleistä. Sovellusten käyttöliittymät eivät enää koostu vain painikkeista ja listoista joita käyttäjät painelevat näppäimistön ja hiiren avulla. Kosketusnäyttöjen yleistyminen puhelimissa, taulutietokoneissa ja jopa sulautetuissa tietokonelaitteissa on esitellyt uudenlaisten pyyhkäisyeleiden käyttämisen osana laitteiden käyttöliittymiä. Pelkän näytön koskettamisen lisäksi käyttäjä voi tehdä erilaisia toimintoja liu'uttamalla yhtä, kahta tai jopa useampaa sormea näytön pinnalla. Esimerkiksi näytöllä olevaa kuvaa voi zoomata tai pyörittää koskemalla näyttöä kahdella sormella yhtä aikaa ja pyörittämällä niitä toistensa ympäri ja vaihtelemalla niiden etäisyyttä toisistaan.

Myös pelaamiseen erikoistuneiden pelikonsolien puolella eleiden hyödyntäminen on yleistynyt. Nintendon kehittämässä Wii-pelikonsolissa pelejä ohjataan painikkeiden lisäksi itse peliohjainta ravistamalla, kallistelemalla tai sillä lyömällä. Pelimaailman hahmo tekee vastaavia liikkeitä peliohjaimen kulloisenkin asennon mukaisesti. Microsoft on tuonut markkinoille Xbox 360-pelikonsolille Kinect-nimisen peliohjaimen, joka koostuu lähinnä kameroista ja etäisyyssensoreista. Tällöin pelaaja ei tarvitse minkäänlaista peliohjainta, vaan Kinect tulkitsee pelaajan vartalon asennot ja liikkeet pelimaailman ohjauskomennoiksi. Uudet digitaalikamerat osaavat tarvittaessa viivyttää esimerkiksi ryhmäkuvan ottamista siten, että yhdenkään kuvattavan henkilön kasvot eivät ole vain puoliksi näkyvissä ja että kaikki hymyilevät. Nämä kaikki toiminnot edellyttävät ihmisten tekemien luonnollisten eleiden tunnistamista.

3D-teknologian käyttäminen on lisääntynyt sekä viihdekäytössä että työelämässä. Monista elokuvista tulee elokuvateattereihin 3D-versiot, joissa elokuvakokemusta saadaan vahvistettua lisäämällä syvyyselementti kuvaan. Katsojalle syntyy suurempi läsnäolon vaikutelma elokuvan tapahtumiin. Kotitalouksiin tarkoitettuihin televisioihin on nykyään saatavilla vastaavia 3D-ominaisuuksia.

Tässä diplomityössä tutkitaan mahdollisuutta käyttää tietokoneen web-kameraa käyttäjän kasvojen paikantamiseen suhteessa tietokoneen näyttöön. Tämä sijainti muutetaan 3D-maailman kameran sijainniksi, jolloin tietokoneen 3D-grafiikkaa esittävä sovellus voi piirtää ruudulle maiseman siitä kuvakulmasta, josta käyttäjä katsoo näyttöä. Tämän tarkoituksena on vahvistaa käyttäjälle syntyvää vaikutelmaa siitä, että tietokoneen näyttö todellakin olisi ikkuna 3D-maailmaan.

Työn rakenne on seuraava. Luvussa 2 esitellään työssä tarvittavat koordinaatistot ja transformaatiot koordinaatistojen välillä. Lisäksi luvussa esitellään oleellisimmat värimallit. Luvussa 3 esitellään 3D-moottorin toimintaa yleisellä tasolla. Luku 4 keskittyy työn toteutuksessa hyödynnettyyn OpenCV-kirjastoon ja sen tarjoamien toiminnallisuuksien esittelyyn. Luvussa 5 määritellään kamerakontrollerin vaatimukset, arvioidaan menetelmien suorituskykyä ja lopuksi esitellään kamerakontrollerin arkkitehtuuri ja toteutus. Luku 6 esittelee valmiille kamerakontrollerille tehtyjen suorituskykyä mitanneiden testien tulokset. Lisäksi luvussa arvioidaan myös kamerakontrollerin toteutusta ja esitetään mahdollisia jatkokehitysajatuksia. Lopuksi luvussa 7 esitetään yhteenveto tehdystä tutkimuksesta.

2 KOORDINAATISTOT JA TRANSFORMAATIOT

3D-grafiikassa on ennen kaikkea kyse kolmiulotteisten (x,y,z) pisteiden liikuttelusta ja pyörittelyistä eri vektoreiden ympäri ja näiden pisteiden välisten alueiden täyttämisestä halutuilla väreillä. Tällaisia pisteitä, jotka rajaavat kappaleen reunoja, kutsutaan vertekseiksi. Lopullinen grafiikan ulkoasu riippuu siitä, miten verteksit yhdistetään toisiinsa ja mistä suunnasta niitä katsotaan. Tästä syystä erilaiset koordinaatistot ja transformaatiot, joilla pisteitä siirretään koordinaatistosta toiseen, muodostavat ehkä tärkeimmän työkalun 3D grafiikan ohjelmointiin.

Kohdassa 2.1 esitellään tärkeimmät 3D-ohjelmoinnissa käytettävät koordinaatistot, jonka jälkeen kohdassa 2.2 esitetään, kuinka pisteitä kuvataan koordinaatistosta toiseen. Kohta 2.3 käsittelee perspektiiviprojektiota ja kohta 2.4 käänteistä perspektiiviprojektiota. Kohdassa 2.5 esitetään, kuinka etäisyyttä voidaan arvioida kahden kiintopisteen avulla. Kohdassa 2.6 käydään läpi työssä tarvittavat värimallit ja lopuksi kohdassa 2.7 näytetään, kuinka väriarvoja kuvataan värimallista toiseen.

2.1 Koordinaatistoja

Karteesinen eli suorakulmainen koordinaatisto on koordinaatisto, jossa kunkin ulottuvuuden suuntainen akseli on kohtisuorassa muiden ulottuvuuksien akseleita kohtaan. Esimerkiksi kolmiulotteisessa karteesisessa koordinaatistossa on kolme toisiaan vastaan kohtasuorassa olevaa akselia. Kaikki merkittävimmät 3D-grafiikan ohjelmointirajapinnat perustuvat kolmiulotteiseen karteesiseen koordinaatistoon.

Objektin koordinaatisto (*object space*) on yhden objektin geometriaan liittyvä koordinaatisto, jonka origo sijaitsee objektin keskipisteessä. Objektin geometrian määrittävien verteksien sijainnit ovat aina suhteessa objektin koordinaatiston origoon. Kun objekti sijoitetaan varsinaisessa maailmassa johonkin paikkaan, se tarkoittaa, että objektin keskipiste sijoitetaan kyseiseen paikkaan. Vaikka objektia liikuteltaisiin 3D-maailmassa miten tahansa, verteksien sijainteja ei tarvitse muuttaa, vaan niiden sijainnit säilyvät objektin koordinaatistossa määriteltyinä. [30]

Maailmakoordinaatisto (*world space*) on 3D-maailman laajin koordinaatisto. Kun objekteja sijoitetaan maailmaan, määrittelemme sen keskipisteelle uuden sijainnin ja tarvittaessa kiertokulmat, mikäli haluamme objektien olevat useissa eri aseinnoissa. Tällä tavalla voimme säästää muistia, kun saman geometrian sisältäviä objekteja voi olla useita eri maailmakoordinaatiston paikoissa, mutta ne kaikki hyödyntävät samaa objektin koordinaatistossa määriteltyä geometriaa. Kun haluamme tarkastella esimerkiksi objektien välisiä törmäyksiä, meidän tulee laskea transformaatio, jolla verteksien sijainnit saadaan kuvattua objektin koordinaatistosta maailmakoordinaatistoon. Käytännössä transformaatio tapahtuu muodostamalla sopiva tranformaatiomatriisi, jolla jokainen objektin koordinaatistossa määritelty verteksin koordinaatti kerrotaan. [30]

Katselukoordinaatisto (*view space*) kertoo pisteiden sijainnit suhteessa katsojan sijaintiin ja katselusuuntaan. Tämän koordinaatiston piste (0,0,0) sijaitsee tismalleen katsojan silmän kohdalla, ja piste (0,10,0) sijaitsee 10 yksikköä katsojan silmän yläpuolella. [30]

Katselukoordinaatistolle on tarvetta tilanteissa, joissa haluamme tietää mitkä objektit sijaitsevat esimerkiksi katsojan takana ja ovat siten näkökentän ulkopuolella. Vaikka 3D-rajapinnat osaavatkin käsitellä tilanteet, joissa objektit ovat kokonaan näkökentän ulkopuolella, on näkymättömissä olevat pisteet ja tasot syytä karsia pois ennen kuin ne siirretään 3D-rajapinnan piirrettäviksi. 3D-rajapinnan valmistelu seuraavan geometrian piirtämiseen edellyttää piirron alustamisia, joiden aikana 3D-grafiikan piirrosta vastaava laitteisto ei kykene piirtämään täydellä teholla. Jos seuraavaksi valmisteltu geometria piirtyisikin ruudun ulkopuolelle, on valmisteluun käytetty suoritusaika tietyssä mielessä hukkaan heitettyä suoritinaikaa, jota olisi voitu käyttää tehokkaamminkin.

Kuvaruutukoordinaatisto on kaksiulotteinen suorakulmainen koordinaatisto, jonka x-akseli kulkee kuvaruudulla vasemmalta oikealle ja y-akseli ylhäältä alas. Tämän koordinaatiston piste (0,0) sijaitsee kuvaruudun vasemmassa yläkulmassa. Kuvaruudun oikean alakulman koordinaatin arvo riippuu näyttölaitteen resoluutiosta jolloin resoluutioon 1680 $\times$ 1050 kykenevän laitteen oikean alakulman koordinaatti on (1679,1049).

Normalisoitu kuvaruutukoordinaatisto on kaksiulotteinen suorakulmainen koordinaatisto, joka kuvaa pisteen sijaintia kuvaruudulla. Koordinaatiston lukualue sekä x- että y-akselien suunnassa suljettu väli [-1,1], jossa -1 viittaa näyttölaitteen vasempaan tai yläreunaan ja 1 vastaavasti oikeaan tai alareunaan. Normalisoitu koordinaatisto on riippumaton näyttölaitteessa käytettävästä resoluutiosta. Oli resoluutio mikä tahansa, x-koordinaatin arvo x=0 sijaitsee aina näyttölaitteen keskellä. Näytön resoluution ollessa w$\times$h, konversio kuvaruudun pisteestä (X_{pix}, Y_{pix}) normalisoituun kuvaruutukoordinaatistoon onnistuu yhtälöiden 2.1 avulla.

$$X_{norm} = \frac{2X_{pix}}{w} - 1, Y_{norm} = \frac{2Y_{pix}}{h} - 1 \tag{2.1}$$

3D-grafiikan tapauksessa normalisoitu kuvaruutukoordinaatisto on viimeinen koordinaatisto, johon pisteet kuvataan näkymää piirrettäessä. Menetelmä, jolla kolmiulotteisista pisteistä saadaan kaksiulotteisen kuvaruutukoordinaatiston pisteitä kutsutaan perspektiiviprojisoinniksi, jota käsitellään tarkemmin alakohdassa 2.3.

2.2 Transformaatiot

Kun 3D-maailma halutaan pirtää ruudulle, selvitetään, miltä se näyttää kun sitä katsotaan jostain tietystä pisteestä tiettyyn suuntaan. Tässä yhteydessä objektien vertekseille tehdään useita transformaatioita koordinaatistosta toiseen. Käytännössä objektin koordinaatistossa määritellyt pisteet tulee ensin transformoida maailmakoordinaatistoon, jotta saadaan selville, missä verteksit sijaitsevat 3D-maailman sisällä. Tämän jälkeen verteksit transformoidaan maailmakoordinaatistosta katselukoordinaatistoon, jotta saadaan selville niiden sijainnin suhteessa katselupisteeseen. Lopuksi verteksit transformoidaan katselukoordinaatistosta normalisoituun kuvaruutukoordinaatistoon. Tämä viimeinen transformaatio kulkee nimellä perspektiiviprojektio.

3D-grafiikan ytimenä oleva matematiikka perustuu kolmiulotteisen koordinaatiston sisällä tehtäviin lineaarimuunnoksiin. Kolmiulotteiselle pisteelle tehtävä lineaarimuunnos tarkoittaa, että sille on voimassa lausekkeen 2.2 mukaiset yhtälöt [13, s. 72].

$$\begin{aligned} x'(x,y,z) &= U_1x + V_1y + W_1z + T_1 \\ y'(x,y,z) &= U_2x + V_2y + W_2z + T_2 \\ z'(x,y,z) &= U_3x + V_3y + W_3z + T_3 \end{aligned} \tag{2.2}$$

Koska 3D-grafiikan laskentaa tehdään yleisesti matriiseilla, tämä sama voidaan esittää lausekkeen 2.3 mukaisessa muodossa [13, s. 72].

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} U_1 & V_1 & W_1 \\ U_2 & V_2 & W_2 \\ U_3 & V_3 & W_3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} T_1 \\ T_2 \\ T_3 \end{bmatrix} \tag{2.3}$$

Lausekkeesta 2.3 nähdään, että transformaatio koostuu kahdesta erillisestä komponentista. Ensimmäinen on U:n, V:n ja W:n kertoimista rakentuva 3x3-matriisi **M**, joka toimii transformoitavan pisteen kertoimena. Toinen komponentti muodostaa siirtovektorin **T**. Matriisin **M** avulla on mahdollista tehdä monenlaisia transformaatioita, mutta 3D-grafiikan käsittelyssä sitä käytetään kuvaamaan pisteen pyöritystä origon ympäri. Pyöritys voi tapahtua joko x-, y- tai z-akselin ympäri käyttämällä lausekkeen 2.4 mukaisia pyöritysmatriiseja [13, s. 78].

$$\mathbf{M}_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\theta & -sin\theta \\ 0 & sin\theta & cos\theta \end{bmatrix}, \mathbf{M}_y = \begin{bmatrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ -sin\theta & 0 & cos\theta \end{bmatrix}, \mathbf{M}_z = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.4)$$

Nämä peruspyöritykset muodostavat pyöritysmatriisin, kun pistettä pyöritetään vain yhden akselin ympäri. Mikäli pyörityksen tulee tapahtua usean akselin ympäri, muodostetaan kutakin akselia vastaava pyöritysmatriisi. Lopullinen pyöritysmatriisi saadaan aikaiseksi kertomalla akselikohtaiset pyöritysmatriisit keskenään siinä järjestyksessä, missä pyöritysten halutaan tapahtuvan. [4, s. 84]

Valitaan objektista yksi verteksi ja oletetaan sen sijainniksi paikkavektorin **P** osoittama sijainti. Vektori **T** kuvaa objektin keskipisteen sijaintia maailmakoordinaatistossa, ja 3x3-matriisi **M** kuvaa objektin kiertoa keskipisteensä ympäri. Verteksin sijainti maailmakoordinaatistossa saadaan laskettua lausekkeen 2.5 avulla. [13, s. 81]

$$\mathbf{P'} = \mathbf{MP} + \mathbf{T} \quad (2.5)$$

Jos tehdään useampia peräkkäisiä transformaatioita, eli ensin transformoidaan matriisin $\mathbf{M}_1$ ja sijainnin $\mathbf{T}_1$ suhteen ja sen jälkeen matriisin $\mathbf{M}_2$ ja sijainnin $\mathbf{T}_2$ suhteen, laskutoimitukset tulee ketjuttaa lausekkeen 2.6 mukaisesti. [13, s. 81]

$$\mathbf{P'} = \mathbf{M}_2(\mathbf{M}_1\mathbf{P} + \mathbf{T}_1) + \mathbf{T}_2 \quad (2.6)$$

Tästä nähdään, että mitä useampia transformaatioita tähdään, sitä useampia kerto- ja yhteenlaskuja per verteksi vaaditaan. Tämä on suorituskyvyn kannalta huono asia, koska yksittäinen objekti saattaa sisältää kymmeniä tuhansia verteksejä, ja jokainen kertolasku enemmän vaikuttaa erittäin raskaasti suorituskykyyn. Tähän ongelmaan löytyy ratkaisu, kun siirrytään käyttämään 4-ulotteisia verteksien paikkavektoreita ja 4x4-kokoisia transformaatiomatriiseja. Verteksien sijainnit itsessään ovat 3-ulotteisia vektoreita, mutta transformaation ajaksi ne laajennetaan 4-ulotteisiksi asettamalla w-komponentti 1:ksi. Eli verteksin **P** sijainniksi tulee vektori ($\mathbf{P}_x$, $\mathbf{P}_y$, $\mathbf{P}_z$, 1).

4x4-kokoiseksi transformaatiomatriisiksi **F** saadaan lausekkeen 2.7 mukainen matriisi [13, s. 82].

$$\mathbf{F} = \begin{bmatrix} M_{11} & M_{12} & M_{13} & T_x \\ M_{21} & M_{22} & M_{23} & T_y \\ M_{31} & M_{32} & M_{33} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.7)$$

Transformaatiomatriisissa **F** on yhdistettynä rotaation määrittävä 3x3-matriisi **M** ja sijainnin määrittävä vektori **T**. Tämän matriisimuodon paras ominaisuus on se, että kun kaksi tällaista transformaatiomatriisia kerrotaan keskenään, tuloksena on yhdistetty trans-

formaatiomatriisi [13, s. 82]. Kun verteksin 4-ulotteinen paikkavektori kerrotaan yhdistetyllä transformaatiomatriisilla, tulos on sama kuin lausekkeen 2.6 määrittämä kahden transformaation antama tulos. Neliulotteisia vektoreita ja matriiseja käytettäessä vaadittavien kertolaskujen määrä jää pienemmäksi kuin kolmiulotteisilla matriiseilla operoitaessa, koska peräkkäiset transformaatiot voidaan ensin yhdistää matriisien kertolaskuna yhteen. Vasta tämän jälkeen tulokseksi saadulla matriisilla kerrotaan objektin tuhansien verteksien paikkavektorit.

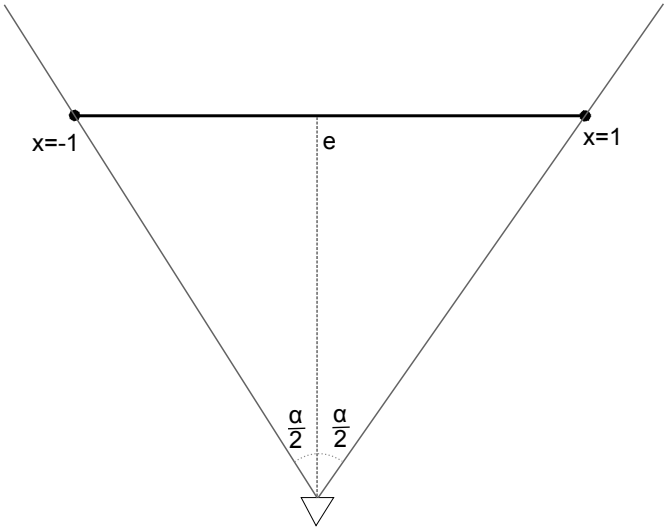
2.3 Perspektiiviprojektio

Kun 3D-maailmassa sijaitseva objekti halutaan esittää kaksiulotteisella pinnalla, kuten valokuvassa, se pitää projisoida kolmiulotteisesta koordinaatistosta kaksiulotteiseen koordinaatistoon. Tämä projisointi voidaan tehdä usealla tavalla, joista yksinkertaisin on ortografinen projektio. Ortografisessa projektiossa pisteen etäisyys katsojasta jätetään kokonaan huomiotta. Käytännössä katselukoordinaatistoon transformoidun (x,y,z)-koordinaatin z-komponentti poistetaan ja projisoitu piste on vain (x,y). Vaikka ortografisella projektiolla tuotettu kuva ei näytä aidolta kolmiulotteisesta kohteesta muodostetulta kuvalta, sille on paljon käyttöä erilaisissa sovelluksissa. Esimerkiksi 3D-grafiikan suunnittelussa käytettävät sovellukset voivat hyödyntää ortografista projektiota esittäessään grafiikkaa käyttäjälle, koska joissain tilanteissa ilman perspektiiviä olevat kuvat helpottavat suunnittelutyötä.

Ortografisessa projektiossa objektin etäisyys ei vaikuta sen näennäiseen kokoon tuotetussa kuvassa. Arkikokemuksesta tiedämme, että kuvassa lähellä oleva esine täyttää suuremman osan kuva-alueesta kuin kaukana oleva samanlainen esine. Tätä etäisyyden vaikutusta projisoidun kuvan kokoon kutsutaan perspektiiviksi. Vastaavasti projektiota, joka huomioi etäisyyden vaikutuksen, kutsutaan perspektiiviprojektioksi.

Perspektiiviprojektiota määriteltäessä voimme vaikuttaa myös siihen, kuinka laaja katselukulma meillä on kolmiulotteiseen maailmaan. Ajatellaan huonetta, jossa on ikkuna. Ikkunasta on näkymä ulos. Jos viemme kameran linssin kiinni ikkunaan ja otamme kuvan, saamme kuvan, josta näkee suhteellisen laajalle aluelle ulkomaailmaan. Jos viemme kameran huoneen takaseinään ja otamme sieltä kuvan, saamme kuvattua huomattavasti kapeamman alueen ulkomaailmaa.

Perspektiiviprojektiossa projektiotason etäisyys katselupisteestä on tärkeä ominaisuus. Jos projektiotaso sijaitsisi täsmälleen katselupisteessä, mitään tasoa ei syntyisi. Kuva 2.1 selventää tilannetta. Kuvasta nähdään, kuinka perspektiiviprojektiossa kaikki kolmiulotteisen maailman pisteet kuvataan projektiotasolla oleviksi pisteiksi. Tämän kuvauksen tuloksena pääsemme eroon etäisyyskomponentista ja jäljelle jää vain (x,y)-koordinaatit, jotka voidaan esittää näyttölaitteen pinnalla. Kuvasta 2.1 voidaan nähdä myös, että projektiotason etäisyys katselupisteestä määrää myös katselukulman laajuuteen. Kun tuomme projektiotasoa lähemmäs, katselukulma laajenee. Vastaavasti viemällä sitä kauemmas,



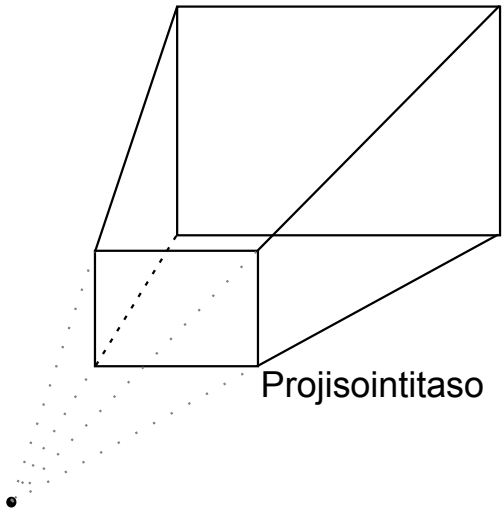
Kuva 2.1. *Katselupiste sijaitsee kuvassa alhaalla keskellä pisteessä (0,0,0). Parametri e ilmaisee projektiontason etäisyyden katselupisteestä. Kulma α on katselukulman suuruus asteina. Kuva on mukailtu lähteestä [13, s. 113]*

katselukulma kaventuu. Projektiotason etäisyyden ja katselukulman välillä on lausekkeen 2.8 mukainen yhteys. [13, s. 114]

$$e = \frac{1}{tan(\alpha/2)} \tag{2.8}$$

Kun perspektiiviprojektion katselukulma on valittu, voidaan tutkia, mitkä kaikki kolmiulotteisen maailman pisteet voivat päätyä projektiotasolle näkyviin. Tämän alueen havaitaan oleva katkaistun pyramidin muotoinen. Katselupiste sijaitsee pyramidin kärjessä. Kohta, josta pyramidi on katkaistu, muodostaa projektiotason ja kaikki pyramidin alaosan sisällä olevat alueet ovat näkyvissä katsojalle ja ne tulee projisoida projektiotasolle. Kuvassa 2.2 on havainnollistettu katselupyramidin muotoa. Kuvassa näkyy myös taso, joka rajaa kauimman mahdollisen etäisyyden, josta objekti voi vielä olla näkyvissä katsojalle. Tämä perustuu siihen, että etäisyyden kasvaessa objektit muuttuvat pienemmän näköisiksi ja jossain vaiheessa objekti muuttuu näyttölaitteen erottelutarkkuutta pienemmäksi. Tällä kauimmalla näkyvällä etäisyydellä ei sinänsä ole muuta merkitystä kuin se, että sen avulla on mahdollista optimoida 3D-grafiikan piirtoa ja jättää piirtämättä liian kaukana olevia kohteita. Englanninkielisessä kirjallisuudessa katselupyramidi tunnetaan nimellä *view frustum*.

Pisteen projisointi projektiotasolle onnistuu käyttämällä kuvausta, joka kuvaa pyramidin muotoisen koordinaatiston suorakulmaisen särmiön muotoiseksi koordinaatistoksi.



Kuva 2.2. *Katselupyramidin (view frustum) muoto. Katselupiste sijaitsee pyramidin sivujen leikkauspisteessä. Pyramidin katkaisukohta toimii projisointitasona. [13, s. 112]*

Tällöin esimerkiksi kaikki pyramidin oikeanpuoleisella tasolla olevat pisteet kuvautuvat tasolle:

$$x = n \tag{2.9}$$

Tässä lausekkeessa n on haluamamme särmiön reunan koordinaatti.

3D-grafiikan käsittelyyn erikoistuneet ohjelmointirajapinnat ja laitteistot ovat käytännössä erikoistuneet käsittelemään normalisoituja koordinaatteja, jolloin meidän kannattaa kuvata katselupyramidin muoto särmiöksi, jonka sivut ovat ovat x = -1, x = 1, y = -1 ja y = 1. Vastaavasti myös lähin mahdollinen etäisyys, eli projektiotaso kuvataan tasoksi z = -1 ja kauin mahdollinen etäisyys on tasolla z = 1. Nämä kuusi tasoa rajaavat sen tilan, jossa objektit ovat näkyviä:

- n: Projektiotason etäisyys kameran sijainnista.
- f: Suurin mahdollinen etäisyys katselupisteestä, joka vielä kuvautuu katselukoordinaatiston sisälle.
- l: Katselusärmiön vasemman reunan X-koordinaatin arvo
- r: Katselusärmiön oikean reunan X-koordinaatin arvo.

- t: Katselusärmiön yläreunan Y-koordinaatin arvo
- b: Katselusärmiön alareunan Y-koordinaatin arvo.

Pisteen transformointi koordinaatistosta toiseen onnistuu aiemmin esitellyllä tavalla kertomalla pisteen 4-ulotteinen koordinaatti sopivalla 4x4-matriisilla. Perspektiiviprojektiota varten meidän tulee rakentaa projektiomatriisi, jossa käytetään hyväksi yllä esiteltyjä kuutta rajaavaa tasoa. Käytetään merkintää $\mathbf{M}_{frustum}$ esittämään katselupyramidin projektiomatriisia. Tällöin lauseke 2.10 esittelee OpenGL-rajapinnan[11] käyttämän projektiomatriisin ja kuinka sen avulla pisti $\mathbf{P}$ saadaan projisoitua katselukoordinaatistoon. [13, s.124].

$$\mathbf{P'} = \mathbf{M}_{frustum}\mathbf{P} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2|n|}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{|n|+|f|}{|n|-|f|} & \frac{2|f||n|}{|n|-|f|} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} \tag{2.10}$$

Edellä esitelty transformaatio kuvaa 4-ulotteisin koordinaatin katselukoordinaatistosta suorakulmaiseen homogeeniseen koordinaatistoon. Projektiomatriisin alimman rivin kertoimien vuoksi projisoidun koordinaatin w-komponentiksi tulee -$\mathbf{P}_z$, joka on pisteen etäisyys katselupisteestä negatiivisena. Lopullinen projisoitu koordinaatti edellyttää, että w-komponentti on 1 ja tämä saadaan jakamalle tuloskoordinaatin arvot negatiivilla $\mathbf{P}_z$ arvolla.

Mikäli perspektiiviprojektion tuloksena pisteelle pätee, että -1 $\leq \mathbf{P}_x \leq$ 1, -1 $\leq \mathbf{P}_y \leq$ 1 ja -1 $\leq \mathbf{P}_z \leq$ 1, niin piste sijaitsee alkuperäisen katselupyramidin sisällä. Jos se taas ei ole edellä mainitulla koordinaattialueella, se on näkyvän alueen ulkopuolella. Halutessamme saada perspektiiviprojisoidun pisteen (x,y)-koordinaatin, sovellamme ortograafista projektiota, eli otamme vain transformoidun pisteen x- ja y-komponentit ja jätämme z-komponentin huomiotta. Piste (-1, -1) sijaitsee kuvaruudun vasemmassa alakulmassa ja (1,1) oikeassa yläkulmassa.

2.4 Käänteinen perspektiiviprojektio

Perspektiiviprojektio on kolmiulotteisen pisteen kuvaus kaksiulotteisessa tasossa, jolloin etäisyystieto sulautetaan x- ja y-akselien arvoihin. Tämän jälkeen emme kykene enää palauttamaan pelkästä kaksiulotteisesta koordinaatista alkuperäistä kolmiulotteisen koordinaatin arvoa. On olemassa ääretön määrä kolmiulotteisen avaruuden pisteitä, jotka perpektiiviprojektion tuloksena kuvautuvat samaksi kaksiulotteiseksi pisteeksi. Jos perspektiiviprojektiossa käytetty projektiomatriisi on tiedossa, voimme laskea sen kolmiulotteisen avaruuden suoran, jolta pisteet kuvautuvat tietyksi kaksiulotteisen avaruuden pisteeksi. Tätä kutsutaan käänteiseksi perspektiiviprojektioksi.

Käänteistä perspektiiviprojektiota tarvitaan esimerkiksi tilanteessa, jossa tietokoneen käyttäjä voi hiiren osoittimen avulla vuorovaikuttaa kolmiulotteista grafiikkaa esittävän sovelluksen objekteihin. Tällöin sovelluksen tulee selvittää, mikä kolmiulotteisen maailman piste on piirrettynä siihen näytön koordinaattiin, johon käyttäjä hiiren osoittimella osoitti. Tällöin käänteisen perspektiiviprojektion avulla selvitetään se suora, jonka pisteet projisoituvat tiettyyn näytön koordinaattiin. Seuraavaksi sovelluksen tulee selvittää ne objektit, jotka kyseinen suora läpäisee ja etsiä niistä se lähimpänä katselupistettä oleva. Tämä menetelmä tunnetaan säteen heittämisenä (*ray casting*).

Säteen suunnan löytäminen onnistuu siten, että otetaan kaksi pistettä, joista toinen sijaitsee lähemmällä tasolla eli projektiotasolla, ja toinen piste sijaitsee kauimmalla tasolla. Perspektiiviprojektion lopuksi käytetyn ortografisen projektion ominaisuuksista tiedämme, että 2-ulotteisen tason *(x,y)* piste on peräisin siltä suoralta, joka kulkee pisteiden (x,y,-1) ja (x,y,1) kautta.

Aiemmin nähtiin, kuinka piste saadaan projisoitua lausekkeen 2.11 avulla.

$$\mathbf{P'} = \mathbf{M}_{proj}\mathbf{P} \tag{2.11}$$

Kun projisoidun pisteen **P'** sijainti ja projektiomatriisi tiedetään, on mahdollista laskea alkuperäinen projisoimaton piste. Koska matriisilla jakaminen ei ole matriiseille määritelty operaatio, ratkaisua täytyy lähestyä toisella tavalla. Perspektiiviprojektiomatriisi **M** on 4x4-neliömatriisi, joten sille voi olla mahdollista löytää käänteismatriisi $\mathbf{M}^{-1}$, kunhan matriisin determinantti ei ole nolla. Mikäli determinantti on nollasta poikkeava, lauseen 2.12 mukainen yhteys on voimassa. [13, s. 53]

$$\mathbf{P'} = \mathbf{MP} \Leftrightarrow \mathbf{P} = \mathbf{M}^{-1}\mathbf{P'} \tag{2.12}$$

Otetaan esimerkki, jossa lasketaan se suora, jolta tietty (x,y) piste on peräisin. Määritellään projisointitasolla ja kaukaisella tasolla olevat pisteet $\mathbf{P'}_{near}$ ja $\mathbf{P'}_{far}$:

$$\begin{aligned} \mathbf{P'}_{near} &= (x, y, -1, 1) \\ \mathbf{P'}_{far} &= (x, y, 1, 1) \end{aligned} \tag{2.13}$$

Tehdään kääteinen perspektiiviprojektio kertomalla nämä pisteet käänteisellä projektiomatriisilla $\mathbf{M}^{-1}$.

$$\begin{aligned} \mathbf{P}_{near} &= \mathbf{M}^{-1}\mathbf{P'}_{near} \\ \mathbf{P}_{far} &= \mathbf{M}^{-1}\mathbf{P'}_{far} \end{aligned} \tag{2.14}$$

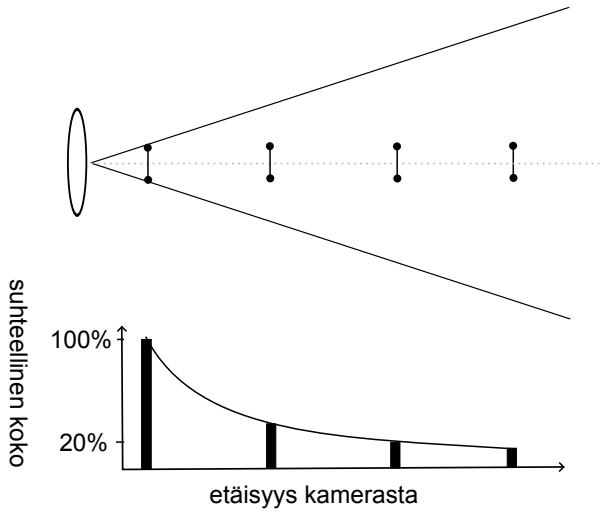
Nyt etsityn (x,y) pisteen käänteisprojisoitu 3D-avaruuden piste löytyy lausekkeen 2.15 esittämältä suoralta.

$$s = \mathbf{P}_{near} + (\mathbf{P}_{far} - \mathbf{P}_{near}) \tag{2.15}$$

2.5 Etäisyyden määrittäminen kuvasta

Kameralla otetusta kuvasta ei suoraan voi päätellä kuin 3-ulotteisen avaruuden suoran, jolta piste on kuvaan päätynyt. Jotta objektin etäisyys kamerasta saadaan selvitettyä, objektista pitää löytää vähintään kaksi kiintopistettä ja tietoa objektin kiintopisteiden todellisesta etäisyydestä toisistaan 3-ulotteisessa avaruudessa.

Eräs tapa selvittää objektin etäisyyttä on mitata objektin kiintopisteiden näennäistä etäisyyttä toisistaan kameran kuvasta ja selvittää millä etäisyydellä objektin on oltava, jotta kyseiset pisteet näyttäisivät olevan mitatulla etäisyydellä toisistaan. Mitä lähempänä objekti on kameraa, sitä suuremmalta se näyttää kuvassa. Kuva 2.3 esittää, kuinka objektin kiintopisteiden etäisyys on kääntäen verrannollinen objektin etäisyyteen kamerasta.



Kuva 2.3. *Objektin koko siitä otetussa kuvassa on kääntäen verrannollinen objektin etäisyyteen kamerasta.*

Oletetaan muuttujan d esittävän objektin kiintopisteiden välistä etäisyyttä toisistaan kuvapisteinä. Valitaan myös muuttuja z, joka esittää objektin etäisyyttä kamerasta. Nämä muuttujat ovat kääntäen verrannollisia, jolloin niiden välillä vallitsee lausekkeen 2.16 mukainen yhteys [23].

$$z = \frac{C}{d} \tag{2.16}$$

Kiintopisteiden välinen etäisyys d kyetään mittaamaan kameran ottamasta kuvasta. Objektin etäisyyden z ratkaiseminen ei kuitenkaan onnistu ennen kuin lausekkeessa esiin-

tyvälle vakiolle C on saatu määritettyä arvo. Tämä saadaan ratkaistua sijoittamalla objekti tunnetulle etäisyydelle kamerasta ja mittaamalla kiintopisteiden välinen etäisyys tässä alkutilanteessa. Tällöin vakio d_0 on kiintopisteiden välinen etäisyys alussa ja z_0 on objektin tunnettu etäisyys alussa. Tämän kalibroinnin tuloksena saamme lausekkeen 2.17, josta objektin etäisyys kamerasta saadaan laskettua aina, kun kiintopisteiden välimatka on mitattu objektista otetusta kuvasta.

$$z = \frac{d_0 z_0}{d} \tag{2.17}$$

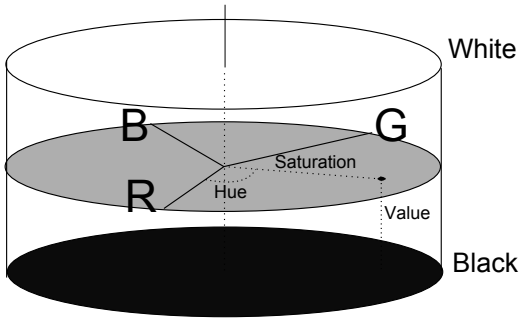
2.6 Värimallit ja muunnokset

Näköaisti perustuu silmässä olevien näköreseptorisolujen kykyyn reagoida valon aallonpituuksiin 400-700 nanometrin aallonpituuskaistalta. Näköreseptorisoluja on kahdenlaisia. Sauvasolut ovat erikoistuneet havaitsemaan valon voimakkuutta koko näkyvän valon aallonpituusalueelta. Ne ovat erittäin herkkiä havaitsemaan pieniäkin valonvoimakkuuksia, mutta ne eivät varsinaisesti kykene aistimaan värisävyjä, koska koko näkyvän valon taajuuskaista saa niissä aikaan reaktion. Sen sijaan silmässä olevat tappisolut mahdollistavat tarkan värinäön ja niitä on kolmea erilaista. Lyhyet tappisolut aistivat sinistä valoa noin 440 nm allonpituudella. Keskimittaiset tappisolut aistivat vihreää valoa, jonka aalonpituus on noin 545 nm. Pitkät tappisolut reagoivat punaiseen valoon 585 nm aallonpituuden lähistöltä. Näitä kolmea väriä kutsutaan pääväreiksi, koska niitä sekoittamalla on mahdollista tuottaa kaikki muut mahdolliset värisävyt. [3, 2. luku]

Tietokonegrafiikassa värit muodostetaan sekoittamalla samoja kolmea pääväriä toisiinsa. Värisävyn muodostamiseen on olemassa kaksi erilaista menetelmää, joista toista kutsutaan summaavaksi sekoittamiseksi ja toista vähentäväksi värien sekoittamiseksi. Summaavassa sekoituksessa kukin väri ajatellaan muodostuvan kolmesta päävärejä säteilevästä valonlähteestä. Päävärejä säteilevien valonlähteiden säteilytehoja säätämällä saadaan kaikki värisävyt aikaiseksi. Mikäli kaikki säteilevät yhtä paljon, tuloksena on valkoista valoa. Vähentävässä sekoittamisessa värin voi ajatella muodostuvaksi siten, että se estää tiettyjä päävärejä heijastumasta pinnasta. Esimerkiksi sininen väri tarkoittaa, että pinta heijastaa vain sinisen värin aallonpituuksia ja imee kaikki muut aallonpituudet sisäänsä. Tietokonegrafiikassa käytetään yleensä summaavaa värien sekoittamista, koska näyttölaitteet ovat tyypiltään valoa säteileviä. [6, s. 200]

Kun kolmen päävärin komponentit sijoitetaan koordinaatistoon, saadaan kolmiulotteinen karteesinen koordinaatisto, jossa punainen, vihreä ja sininen akseli ovat toisiaan vastaan kohtisuorassa. Jokaisen komponentin arvo voi vaihdella välillä [0..1]. Tällaisen RGB-värimallin geometrinen tulkinta on muodoltaan kuutio. Musta sijaitsee origossa, jossa kunkin kolmen komponentin arvo on 0. Valkoinen sijaitsee koordinaatissa (1,1,1) ja harmaan sävyt ovat kuution lävistäjällä pisteestä (1,1,1) pisteeseen (0,0,0). [5, s. 226-227]

Tämän diplomityön kannalta toinen hyödyllinen värimalli on HSV (*Hue, Saturation, Value*). HSV-värimalli koostuu kolmesta komponentista, jotka ovat näkyvän valon spektrillä sijaitseva värisävy (*hue*, H), värikylläisyys (*saturation*, S) ja kirkkaus (*value*, V). H-komponentti on puhdas spektriltä löytyvä värisävy. S-komponentti kuvaa, kuinka paljon valkoista väriä on sekoittunut H-komponentin määräämään värisävyyn ja V-kuvaa kuinka lähellä mustaa tai kirkkainta mahdollista arvoa kyseinen väri on. Koordinaatistona HSV-värimalli muodostaa sylinterin muotoisen koordinaatiston, jossa värisävyn H-komponentti on kiertokulma koordinaatiston pystyakselin ympäri, värin kylläisyyden S-komponentti on etäisyys koordinaatiston pystyakselilta ja kirkkauden V-komponentti on sijainti pystyakselilla. Kuva 2.4 esittää, kuinka HSV-värimallin sylinterikoordinaatisto rakentuu. [2, s. 2262-2263] [5, s. 229]



Kuva 2.4. *HSV-värimallin esittäminen sylinterikoordinaatiston avulla. H-komponentti kuvaa kiertokulmaa keskiakselin ympäri ja toimii värisävyn valitsimena. S-komponentti kuvaa värisävyn puhtautta. V-komponentti on värin kirkaaus. Kuva on tehty mukaillen lähdettä [2, p. 2264].*

Kuvankäsittelyssä ja konenäkösovelluksissa HSV-värimallin hyödyllisyys tulee siitä, että se eriyttää kirkkauden erilliseen komponenttiin kuin missä varsinainen värisävy on. Tämä helpottaa tietyn värin perusteella tehtäviä suodatuksia, koska valaistuksen kirkkaus vaikuttaa vain V-komponenttiin, mutta ei varsinaisen värin määrääviin H- ja S-komponentteihin ja tällöin V-komponentti voidaan jättää tarvittaessa huomiotta. Myös tietyn värin avulla tapahtuva kun segmentointi on laskennallisesti kevyempää, koska se voidaan tehdä vain H-komponentin avulla sen sijaan, että laskutoimituksia pitäisi tehdä esim. RGB-värimallin kaikille kolmelle komponentille. HSV-värimallin heikkoutena on se, että kun värien kylläisyys on pieni, eli värit ovat haaleita, pienikin muutos RGB-väriavaruuden arvoissa saa aikaan suuren muutoksen värisävyn määräävässä H-komponentissa [2, s. 2263].

2.7 RGB - HSV -muunnos

Panin esittää [24, luku 4.1.1] menetelmän, kuinka RGB-värimallin mukainen väri saadaan muunnettua HSV-värinmallin väriksi. Menetelmässä tunnettuina muuttujina ovat R, G ja B, jotka sisältävät muunnettavan värin punaisen, vihreän ja sinisen väriarvon väliltä [0..1]. Tällöin esimerkiksi valkoista vastaava väriarvo on R=1, G=1 ja B=1.

Aluksi lasketaan vakio C, joka toimii HSV-värikomponenttien laskennassa kertoimena:

$$C = max(R,G,B) - min(R,G,B) \tag{2.18}$$

Tämän jälkeen H-komponentti saadaan laskettua H'-apumuuttujan avulla:

$$H' = \begin{cases} määrittelemätön, & \text{jos C = 0} \\ \frac{G-B}{C} mod 6, & \text{jos M = R} \\ \frac{B-R}{C} + 2, & \text{jos M = G} \\ \frac{R-G}{C} + 4, & \text{jos M = B} \end{cases} \tag{2.19}$$

$$H = 60^\circ \times H' \tag{2.20}$$

Värikylläisyys S saadaan laskettua näin:

$$S = \begin{cases} 0, & \text{jos C = 0} \\ \frac{C}{V}, & \text{muulloin} \end{cases} \tag{2.21}$$

Ja lopuksi V-komponentin arvo saadaan laskettua ottamalla RGB-komponenteista suuriman komponentin arvo:

$$V = max(R,G,B) \tag{2.22}$$

3 3D-MOOTTORIN RAKENNE

Kolmiulotteisen grafiikan tuottaminen on huomattavasti monimutkaisempi operaatio kuin kaksiulotteisen tasografiikan käsittely. Kaksiulotteista objekteja piirrettäessä pitää tietää objektien x- ja y-sijainti, leveys, korkeus ja piirtojärjestys. Objektit eivät voi varsinaisesti lävistää toisiaan vaan viimeksi piirretty peittää aiemmin piirretyn objektin niiltä osin, kuin ne leikkaavat toisiaan. Myöskään katselukulmia ja perspektiiviä ei tarvitse huomioida, koska kaksiulotteisessa grafiikassa ei ole kuin leveys- ja korkeussuuntaiset ulottuvuudet.

Kolmiulotteisessa grafiikassa objektit voivat leikata ja lävistää toisiaan monesta kohtaa. Lisäksi objekteja voidaan katsoa eri suunnista ja vaihtelevilla etäisyyksillä, jolloin objektit näyttävät eri kokoisilta, vaikka ne ovatkin saman kokoisia. Kolmiulotteisen grafiikan yhteydessä tulee huomioida myös valonlähteiden ja objektin materiaalin vaikutus valon heijastumiseen objektin pinnasta. Ohjelmistoa, joka osaa tuottaa ruudulle kolmiulotteista grafiikkaa, kutsutaan 3D-moottoriksi. Tässä luvussa esitellään 3D-moottorien yleisiä piirteitä ja kuinka 3D-maailman sisältö saadaan lopulta piirrettyä näytölle.

Tämän luku alkaa 3D-objektien geometriaan liittyvien menetelmien käsittelyllä kohdassa 3.1. Tämän jälkeen käsitellään renderöintiä ja 3D-näkymää kohdissa 3.2 ja 3.3. Näiden jälkeen tutustutaan yleisiin 3D-grafiikkarajapintoihin kohdassa 3.4. Lopuksi kohdassa 3.5 esitellään diplomityön esimerkkisovelluksen toteutuksessa hyödynnettyä OGRE-grafiikkamoottoria.

3.1 Objektien geometrian esittäminen

Kolmiulotteista grafiikkaa voidaan koostaa kahdella erilaisella menetelmällä, vokseli- tai polygonimenetelmällä. Vokselimenetelmässä maailma koostuu pienistä 3D-avaruuteen sijoitetuista pisteistä, joille piirtovaiheessa annetaan väritys ja tilavuus. Kun useita vokseleita sijoitetaan halutunlaisesti toistensa lähelle, saadaan aikaiseksi 3D-objektin ulkoasu. Vokselimenetelmä on laskennallisesti erittäin raskas, koska siinä 3D-objektit koostuvat sadoista, jopa tuhansista pienistä 3D-objekteista, joiden ulkoasu ja valaistus täytyy piirrettäessä laskea. Yleisin vokselin muoto on pieni kuutio, joita pinotaan ja asetetaan toistensa lähelle siten, että tuloksena on halutun kaltainen 3D-objekti. Vokselin koko vaikuttaa siihen, kuinka rosoiselta ja palikkamaiselta objekti näyttää. Koska laskentatehon tarve kasvaa vokselien koon pienetessä, käytännössä on pakko löytää kompromissi näyttävyyden ja tarjolla olevan laskentatehon välillä. Monikulmiomenetelmä on eniten käytetty menetelmä 3D-grafiikan tuottamiseen. Siinä objektin geometria muodostetaan toisiinsa

kytkettyjen monikulmioiden avulla. Menetelmän suurimpana etuna on, että yksi monikulmio muodostaa yhtenäisen pinnan, jonka koko voi olla mielivaltainen ja jolla vallitsee yhtenäiset valoa absorboivat ja heijastavat ominaisuudet.

Kun objektin geometria halutaan esittää monikulmioina, ensimmäinen tärkeä elementti on monikulmioiden kulmapisteet. Kulmapisteitä kutsutaan vertekseiksi ja ne ovat kolmiulotteisessa koordinaatistossa sijaitsevia (x,y,z) -pisteitä. Objektin muoto saadaan sijoittelemalla verteksejä siten, että ne vastaavat halutun objektin muotoa.

Pelkät verteksit eivät vielä riitä objektin 3D-mallin tuottamiseen, koska ne itsessään ovat pisteitä, joita ei piirretä näkyviin. Varsinainen objekti saadaan aikaiseksi, kun useita verteksejä yhdistetään monikulmioiksi, jotka yhdessä muodostavat objektin pinnan. Esimerkiksi kuvassa 3.1 on esitetty, kuinka verteksijoukon ja kolmioiden avulla saadaan muodostettua mielivaltaisen muotoinen monikulmio.

Vaikka useampikulmaisten monikulmioiden käytölle ei ole mitään periaatteellisia esteitä, on olemassa hyvät syyt sille, että käytännössä vertekseistä muodostetaan aina kolmion muotoisia monikulmioita. Näitä syitä ovat mm. seuraavat:

- Kolmiot ovat aina konvekseja eli kuperia. Kuperat monikulmiot ovat laskennallisesti edullisempia piirtää, koska niiden reunat eivät voi leikata toisiaan. [27, pp. 39-40]

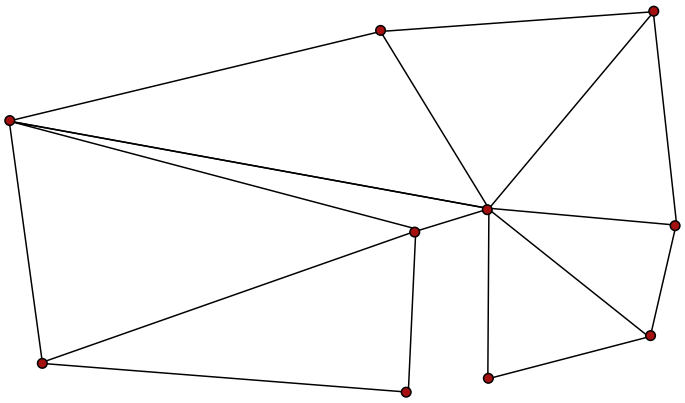
- Kolmion, jonka pinta-ala on nollaa suurempi, kulmapisteet muodostavat aina tasopinnan.

- Mikä tahansa monikulmio voidaan aina jakaa joukoksi kolmiota. Grafiikkarajapintojen toteutukset jakavat monikulmiot aina kolmioiksi, joten suorituskyvyn parantamiseksi kannattaa käyttää kolmioita jo alusta asti. [16, p. 4]

3.2 Renderöinti

Renderöinti on operaatio, joka tehdään 3D-näkymälle, kun se halutaan saattaa kuvaruudulle näkyväksi. Nykyaikaiset tietokoneet sisältävät erityisen näytönohjain-piirin, joka on erikoistunut piirtämään 3D-grafiikkaa itsenäisesti. Tietokoneen suoritin valmistelee piirto-operaatiot, eli kuvaa näytönohjaimelle mitkä verteksit, millaisilla tekstuureilla ja millä transformaatiomatriiseilla kerrottuina. Lopuksi näytönohjain alkaa piirtää annettuja geometrioita itsenäisesti ja huomattavasti nopeammin kuin tietokoneen suoritin pystyisi niitä piirtämään.

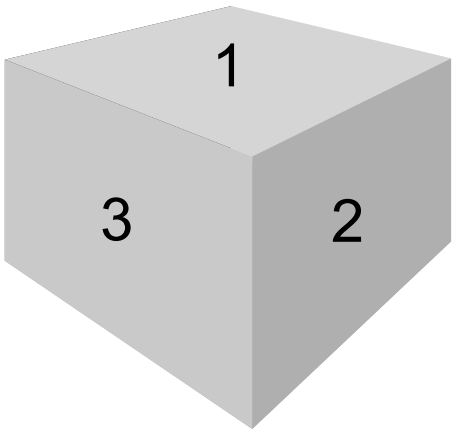
Yleensä 3D-grafiikan piirrossa on käytössä Z-puskurointi, jolloin näytönohjain pitää kirjaa piirtämiensä pikseleiden z-koordinaateista eli etäisyyksistä kameraan. Mikäli eri geometrioita piirrettäessä päädytään piirtämään pikseliä samaan kohtaan ruutua kuin mihin on aikaisemmin piirretty, aikaisempien piirtokertojen z-arvo ratkaisee, piirretäänkö pikseli oikeasti kuvaruutuun vai ei. Mikäli aikaisemmin piirretty pikseli on z-arvoltaan



Kuva 3.1. *Esimerkkikuva siitä, kuinka kolmioiden avulla voidaan tehdä monikulmio.*

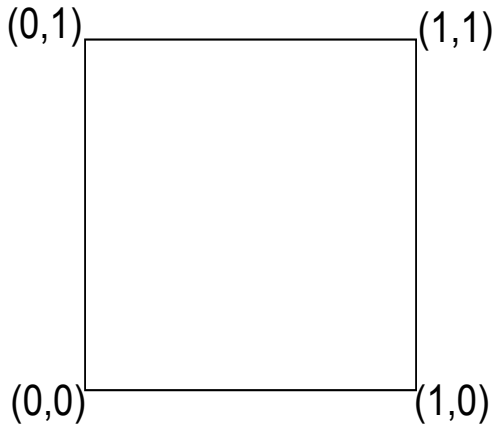
pienempi, uutta pikseliä ei piirretä vanhan päälle, koska se on kauempana kuin aikaisempi pikseli. Z-puskurin ansioista geometrioiden piirtojärjestys on vapaa ja silti lähempänä katsojaa olevat kohteet peittävät kauempana olevat kohteet taakseen. [27, s. 146]

Vaikka Z-puskurin avulla piiloon jäävät pikselit eivät tulekkaan näkyviin, niiden käsittely näytönohjaimessa hidastaa näkymän piirtoa. 3D-moottorin tulee osata jättää piirtämättä monikulmiot, jotka eivät voi näkyä katsojalle. Esimerkiksi kuution muotoisella objektilla on yhteensä 6 tasopintaa, mutta katsoja voi nähdä niistä maksimissaan vain 3 kerrallaan. Tätä on havainnollistettu kuvassa 3.2.



Kuva 3.2. *Riippumatta mistä suunnasta kuutiota katsotaan, katsoja voi nähdä korkeintaan 3 tasoa.*

Teksturointi on oleellinen osa renderöintiä. Teksturoinnissa bittikarttakuva sovitetaan täyttämään piirrettävän monikulmion ala sen sijaan, että monikulmio täytettäisiinkin kauttaaltaan vain tietyllä värillä. Bittikarttakuvaa, joka sovitetaan monikulmioon, kutsutaan tekstuuriksi. Teksturointi on tehokas tapa saada lisää värikkyyttä ja monipuolisuutta 3D-grafiikkaan ja käytännössä kaikki pelit viimeisten 10 vuoden ajalta käyttävät teksturointia. [1, 7. luku]



Kuva 3.3. *Tekstuurikoordinaattien sijoittuminen tekstuuriin. Mukailtu lähteestä Benstead et al. [1, kappale 7]*

Tekstuurit ovat muodoltaan suorakulmaisia bittikarttoja, mutta niistä ei ole pakko käyttää suorakulmaisia alueita. Monikulmiota teksturoitaessa jokaiseen verteksiin voidaan liittää tieto myös ns. tekstuurikoordinaatista, joka on (u,v)-koordinaattipari. Tekstuurikoordinaatti määrittää sen tekstuurin kohdan, josta luetaan väri kyseisen verteksin määrittämälle monikulmion kohdalle. Kuva 3.3 esittää tekstuurikoordinaattien sijoittumista tekstuuribittikartan sisällä. [1, 7. luku]

Tekstuurikoordinaattien käsittelyyn liittyy seuraavat periaatteet [27, s. 233]:

- Tekstuurikoordinaatit ovat realilukuja väliltä [0,1]. Normalisoidun koordinaatiston käyttäminen mahdollistaa tekstuurien resoluution eli koon muuttamisen ilman, että käytettyihin koordinaatteihin tarvitsee tehdä muutoksia.

- Tekstuurikoordinaatiston piste (0,0) riippuu käytettävästä grafiikkarajapinnasta. OpenGL määrittelee, että sijainti (0,0) on tekstuuribittikartan vasemmassa alakulmassa. Direct3D-rajapinnan[17] yhteydessä piste (0,0) sijaitsee tekstuurin vasemmassa yläkulmassa.

Kun monikulmion jokaiseen verteksiin on asetettu yksi tekstuurin koordinaatti, niin kuinka selvitetään ne tekstuurin koordinaatit, joita käytetään täyttämään kolmion sisä-

osat? Vastaus on interpoloimalla. Monikulmion piirtäminen tapahtuu pikseli kerrallaan, ja kunkin piirrettävän pikselin kohdalla selvitetään sen suhteellinen etäisyys muista vertekseistä. Tämän jälkeen selvitetään kuhunkin verteksiin liitetyt tekstuurikoordinaatit ja lasketaan samassa suhteessa oleva etäisyys kunkin kulmapisteen tekstuurikoordinaateista.

3.3 3D-näkymä

3D-näkymä koostuu kaikista niistä objekteista, jotka voivat päätyä piirrettäviksi, kun näkymä lopulta piirretään ruudulle. Renderöintioperaation aluksi 3D-näkymä ottaa käytössä olevan 3D-kameran ja tutkii mitkä kaikki objektit ovat kameran katselupyramidin sisällä. Mikäli objekti ei ole katselupyramidin sisällä, sitä ei ole myöskään tarvetta piirtää. Kun kameran näkökentässä olevat objektit on selvitetty, 3D-näkymän tehtävänä on valmistella renderöintioperaatio kullekin objektille ja laskea objektin tarvitsemat transformaatiomatriisit.

Tavallisesti näkymässä olevat objektit voidaan luokitella kolmeen ryhmään. Kiinteiden objektien tasopinnat eivät sisällä läpinäkyviä kohtia, ja niissä käytetään vain yhtä tekstuuria. Toiseen ryhmään kuuluu objektit, jotka ovat läpinäkymättömiä, mutta niiden teksturoinnissa käytetään useita alpha-kanaavaa, eli osittaista läpinäkyvyyttä sisältäviä tekstuureita. Kolmannen ryhmän muodostavat läpinäkyvät objektit, joita piirrettäessä niiden takana olevat objektit näkyvät ainakin osittain läpi. [15, 4. luku]

Marucchi-Foino suosittelee [15, 4. luku] erilaisille objekteille tällaista piirtojärjestystä:

1. Piirrä kiinteät objektit lähimmästä kauimpaan. Z-puskurin ansiosta ensin lähelle piirretyt objektit peittävät kauemmaksi piirrettävät tasopinnat taakseen, jolloin z-testin jälkeen pikseleitä ei oikeasti tarvitse piirtää. Tämä nopeuttaa piirtoa jonkin verran.

2. Alpha-kanavaa sisältävät objektit lähimmästä kauimpaan.

3. Läpinäkyvät objektit kauimmasta lähimpään.

Piirto-operaatioiden valmistelussa objektien piirtojärjestyksellä on paljon vaikutusta. On hyvä huomata, että piirtorajapinnat, kuten OpenGL tai DirectX, eivät tarjoa 3D -näkymän käsittelyyn paljoakaan toiminnallisuutta, vaan sovelluskehittäjän on itse suunniteltava, millaisilla tietorakenteilla ja algoritmeilla hänen 3D-näkymänsä toimii [33, s. 4]. Tarjolla on kuitenkin useita ilmaisiakin 3D-moottoreita, kuten esimerkiksi OGRE, jota käytettiin tämän diplomityön yhteydessä esimerkkisovelluksen tekemiseen.

3.4 Renderöintirajapinnoista

Yleisesti tunnettuja renderöintirajapintoja on kaksi erilaista, joista toinen on nimeltään OpenGL ja toinen Direct3D. Nykyisin rajapintojen ominaisuuksissa ei ole suuria eroja,

mutta Direct3D:n kerrotaan olevan jonkin verran parempi, koska se osaa hyödyntää säikeistystä OpenGL:ää paremmin [25]. Kumpikin rajapinta mahdollistaa kuitenkin nopean ja näyttävän 3D-grafiikan tuottamisen.

OpenGL on alunperin Silicon Graphicsin ja nykyään Khronos Group nimisen yhteenliittymän hallinnoima 3D-grafiikan ohjelmointirajapinta. OpenGL:lle ominaista on, että Khronos Group tarjoaa vain ohjelmointirajapintaa, mutta ei lainkaan rajapinnan toteutusta. Toteutusvastuu on aina näytönohjaimen ja laitteiden valmistajilla. OpenGL-rajapinnan avoimuuden vuoksi rajapinnalle on toteutuksia monille erilaisille tietokonejärjestelmille, kuten Windowsille, Linuxille ja mobiilialustoille. [1, s. 7]

Direct3D on Microsoftin kehittämä ja hallinnoima 3D-grafiikan piirtämiseen soveltuva rajapintamäärittely. Koska Microsoft on pitänyt rajapinnan ja sen toteutuksen tiukasti itsellään, Direct3D-rajapinta on käytettävissä vain Microsoftin Windows, Xbox 360 ja Windows Phone 7 käyttöjärjestelmissä. Usein myös Direct3D-rajapinnan avulla saadaan tehtyä näyttävämpää grafiikkaa kuin OpenGL-rajapinnan avulla, koska monet edistyneemmät ominaisuudet ovat Direct3D:ssä tuettuina mutta eivät OpenGL:n perusrajapinnassa. [28, 1. luku]

Nykyään molemmille rajapinnoille yhteistä on, että varsinainen 3D-pisteiden pyörittäminen ja kuvapisteiden piirtäminen tapahtuu niin kutsuttujen varjostinohjelmien (*shader programs*) avulla. Varjostinohjelmat ovat yleensä suhteellisen yksinkertaisia näytönohjaimessa ajettavia ohjelmia, jotka voidaan karkeasti jakaa kahteen ryhmään. Ensimmäinen on verteksivarjostimet, joita käytetään laskemaan objektin jokaiselle verteksille tehtävät transformaatiot. Toinen ryhmä on pikselivarjostimet, joita käytetään laskemaan jokaiselle piirrettävälle kuvapisteelle väri. Varjostinohjelmat ovat sovelluskehittäjän vapaasti kirjoitettavissa ja siksi ne mahdollistavat näyttävien graafisten efektien ohjelmoimisen. Esimerkiksi aiemmin mainittu teksturointi on pikselivarjostimen ohjelmaan kirjoitettu toiminnallisuus, jossa se lukee tekstuurikuvasta kuvapisteen, kenties muuttaa sen värisävyä hieman ja palauttaa lopulta näytölle piirrettävän pikselin väriarvon. Näytönohjaimet kykenevät ajamaan jopa satoja varjostinohjelmia samanaikaisesti. Tämä onkin tarpeen, koska objektissa saattaa olla tuhansia verteksejä ja ruudulla miljoonia pikseleitä. [12, s. 6-9] [14]

Varsinainen 3D-sovellus voi antaa varjostinohjelmille syöteparametreina verteksien sijainteja ja kokoelman vakioita. Vakioiden arvot voivat muuttua renderöintikertojen välillä, mutta yhden renderöintioperaation aikana ne pysyvät samana. Jos esimerkiksi objektissa on 1500 verteksiä, niin objektia piirrettäessä verteksivarjostinohjelma suoritetaan 1500 kertaa. Kullakin ajokerralla verteksin koordinaattina on eri sijainti, mutta jokainen näistä verteksivarjostinohjelmista näkee samat vakiot. Seuraavalla piirtokerralla vakioiden arvoja voidaan taas muuttaa.

Vaikka OpenGL ja Direct3D molemmat tukevat varjostinohjelmia, on rajapintojen käyttämien varjostinohjelmien ohjelmointikieli erilainen. OpenGL:n käyttämä varjostinkieli on nimeltää GLSL ja Direct3D:n HLSL. Ne muistuttavat suuresti toisiaan ja syntak-

siltaan ovat lähellä C-ohjelmointikieltä. GLSL:n ja HLSL:n merkittävimmät erot liittyvät varjostinohjelmien käytössä olevien funktiokirjastojen sisältöihin. [12, s. 13-14]

3.5 OGRE-grafiikkamoottori

OGRE on erittäin suosittu avoimen lähdekoodin 3D-moottori. Lähdekoodi on lisensoitu MIT-lisenssillä, joka sallii lähdekoodin käyttämisen, jakamisen ja muokkaamisen kunhan sovelluksen mukana toimitetaan OGRE:n lisenssiehdot sisältävä tekstitiedosto. OGRE:n kehitys voidaan katsoa alkaneeksi vuonna 2000, jolloin projekti rekisteröitiin Sourceforge-palveluun. Sen jälkeen OGRE:n kehittämiseen on osallistunut useita kehittäjiä ja siitä on muodostunut monipuolinen 3D-grafiikkamoottori, joka on sovitettu toimimaan Linux-, Windows-, MacOS X- ja iOS-alustoilla. Lisäksi tarjolla on vielä epäviralliset Android- ja Windows Phone 8-versiot. [20] [21]

OGRE on toteutettu C++-ohjelmointikielellä. Sen tarjoama ohjelmointirajapinta piilottaa varsinaisen renderöintirajapinnan täydellisesti. Tämän ansiosta OGRE:a käyttävä sovellus osaa automaattisesti hyödyntää niin OpenGL- kuin Direct3D-rajapintaa grafiikan renderöimiseen. Renderöintirajapinnan vaihtaminen ei edes tarvitse sovelluksen uudelleen kääntämistä, koska OGRE lataa käytettävän renderöintirajapinnan ajonaikaisesti, joten periaatteessa sovellus voi lennossa siirtyä OpenGL-rajapinnan käytöstä Direct3D-rajapinnan käyttämiseen.

OGRE tarjoaa materiaalimekanismin, jonka avulla objekteille ja polygoneille voidaan määritellä värit, tekstuurit ja varjostinohjelmat helposti yhdessä tekstitiedossa. Mikäli pelkkä teksturointi, väritys ja valaistuksen käsittely riittää, ei sovelluskehittäjän tarvitse kirjoittaa riviäkään varjostinohjelmien koodia, koska OGRE osaa tuottaa tarvittavat varjostinohjelmat perusmateriaaleille. Mikäli perusmateriaalit eivät riitä, OGRE sallii sovelluskehittäjän kirjoittaa suoraan varjostinohjelmien lähdekoodia. OGRE:n tukemat varjostinkielet ovat HLSL, GLSL ja nVidia-yhtiön kehittämä Cg. Näistä Cg-kielen vahvuus on, että useimmiten sillä kirjoitettua varjostinohjelman saa toimimaan sekä Direct3D-rajapinnan että OpenGL-rajapinnan kanssa. [22]

4 OPENCV-KIRJASTO

OpenCV on BSD-linsenssin alaisuudessa julkaistava avoimen lähdekoodin kirjasto, joka tarjoaa toimintoja reaaliaikaisen konenäön toteuttamiseksi. OpenCV tarjoaa rajapinnat C-, C++- ja Python-ohjelmointikielille, ja se on sovitettu toimimaan lukuisilla työpöytä- ja mobiilikäyttöjärjestelmillä. Kirjaston kehityksen aloitti Gary Bradsky työskennellessään Intelillä. Projektin tarkoituksena oli kiihdyttää konenäön tutkimusta ja kaupallista käyttöä ja sen kautta lisätä kysyntää entistä suorituskykyisemmille suorittimille. Willow Garage on robotiikan kehittämiseen erikoistunut tutkimuslaboratorio, josta tuli OpenCV-kirjaston ylläpidosta vastaava taho vuonna 2008. [32]

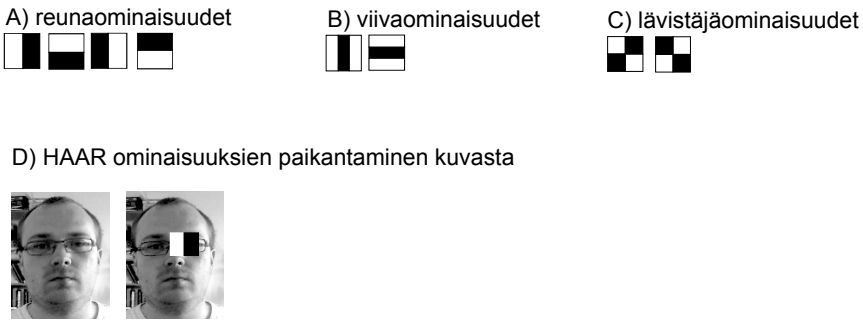
Tässä diplomityössä OpenCV-kirjastoa tarvitaan kahteen eri tarkoitukseen. Ensimmäinen on kasvojen paikantamiseen web-kameran kuvasta. Tämä tapahtuu kohteiden tunnistamiseen sopivalla HAAR Cascades-menetelmällä. Kohdassa 4.1 esitellään HAAR Cascades menetelmän toimintaa ja kohdassa 4.2 esitellään menetelmän rajoituksia. HAAR Cascades-menetelmän lisäksi OpenCV-kirjastoa käytetään myös kasvojen paikantamiseen ihonvärin avulla. Tätä käsitellään tarkemmin kohdassa 4.3.

4.1 HAAR Cascades

OpenCV-kirjaston tarjoama HAAR Cascades -hahmontunnistusalgoritmi on eräs toteutus Viola-Jones -hahmontunnistuskehyksestä. Paul Viola ja Michael Jones esittelivät realiaikaiseen kasvojen paikantamiseen soveltuvan algoritmin, jossa etsitään vierekkäisiä suorakaiteen muotoisia alueita, joissa keskimääräinen kirkkaus vaihtelee siten, että yhden suorakaiteen voidaan sanoa olevan vaalea ja toisen tumma. Näitä vierekkäin olevia tummien ja vaaleiden suorakaiteiden muodostamia ryhmiä kutsutaan nimellä HAAR-ominaisuudet (*HAAR like features*).

Alkuperäisessä Viola-Jones-algoritmissa HAAR-ominaisuudet luokiteltiin kolmeen kategoriaan – viivaominaisuuksiin, reunaominaisuuksiin ja lävistäjäominaisuuksiin. Näiden ominaisuuksien avulla kuvasta yritetään löytää kirkkausvaihteluita. Kun esimerkiksi viivaominaisuuden muodostavaa piirrettä etsitään kuvasta, suorakaide ryhmä sijoitetaan halutussa mittakaavassa kuva päälle ja kaikki valkoisten suorakaiteiden sisään jäävien kuvapisteiden kirkkaudet lasketaan yhteen ja niistä vähennetään mustan suorakaiteen pisteiden kirkkaus. [10, s. I-512]

Suorakaiteen sisällä olevien kuvapisteiden kirkkauden laskeminen saadaan nopeaksi operaatioksi, kun alkuperäisestä kuvasta muodostetaan ensin ns. integraalikuva. Integraa-



Kuva 4.1. *Havainnekuva Viola-Jones-algoritmin käyttämistä HAAR-ominaisuuksista. A on kahdesta suorakaiteesta rakentuva reunaominaisuus, B on esimerkki kolmesta suorakaiteesta rakentuvasta viivaominaisuudesta ja C esittää neljän suorakaiteen muodostamaa lävistäjäominaisuutta. Kuva on piirretty mukaillen lähdettä [9, kohta 2.1.]*

likuvan jokaiseen pisteeseen lasketaan alkuperäisestä kuvasta kaikki vasemmalla ja yläpuolella olevien pikselien kirkkauksien summa [19, s. 21]. Integraalikuvasta minkä tahansa mielivaltaisen suorakaiteen alueen kirkkaus saadaan laskettua vakioajassa neljästä pisteestä luettujen arvojen erotusten avulla. [9, kohta 2.1.]

HAAR-Cascades nimityksen ”cascades”-osalle saadaan merkitys, kun piirteiden analysointi muutetaan kerroksittaiseksi. Ensin kulloisestakin etsintäikkunasta etsitään isomman mittakaavan HAAR-ominaisuuksia. Mikäli haluttuja suuren mittakaavan ominaisuuksia ei löydy, voidaan etsiminen lopettaa siihen. Mikäli etsittyjä ominaisuuksia löytyy, siirrytään etsimään pienempiä ja useampia ominaisuuksia. Tällä tavalla saadaan tehokkaasti hylättyä kulloisenkin etsintäikkunan sisältö ilman, että tarvitsee suorittaa suurta määrää laskutoimituksia. Tällaista monikerroksista HAAR-ominaisuuksien kokoelmaa kutsutaan luokittelijaksi (*classifier*).

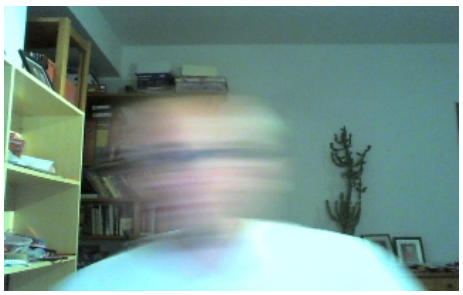
Jotta algoritmi tunnistaisi tietynlaisia hahmoja, se pitää opettaa tunnistamaan niitä. Rezaein ja Kletten mukaan opettamiseen tarvitaan jopa tuhansia kuvia, joissa tunnistettava hahmo esiintyy etsintäikkunan kokoisena ja lisäksi vastaava määrä kuvia, joissa tunnistettavaa hahmoa ei ole [26, s. 176-177].

4.2 HAAR Cascades-menetelmän rajoitukset kasvojen tunnistuksessa

Vaikka HAAR Cascades-menetelmä onkin toimiva menetelmä kasvojen paikantamiseen, siinä on muutamia heikkouksia, jotka hankaloittavat sen hyödyntämistä tosiaikaisessa

kasvojen paikantamisessa. Tässä diplomityössä on tavoitteena kehittää menetelmä, jolla voidaan liikuttaa 3D-maailman kameraa mahdollisimman tosiaikaisesti käyttäjän pään liikkeiden mukaisesti. Tällöin seuraavat HAAR Cascades-menetelmän heikkouden hankaloittavat menetelmän käyttöä kamerakontrollerin ytimenä.

Motion blur-efekti syntyy, kun kuvattava kohde liikkuu nopeasti ja kameran valotusajan aikana kohde ehtii liikkua huomattavan määrän. Tällöin liikkuvasta kohteesta tulee epätarkka ja sumuinen kuva. Kun käyttäjä liikkuu nopeasti. web-kameran ottamassa kuvassa voi olla suuriakin määriä liikkeen aiheuttamaa sumua. Tällöin kasvot eivät erotu kuvasta selkeästi, ja HAAR Cascades-menetelmän käyttö kasvojen paikantamiseen ei onnistu kyseisestä kuvassa. Esimerkiksi kuvassa 4.2 on nähtävissä tilanne, jossa käyttäjä on liikuttanut kasvojaan nopeasti oikealta vasemmalle.



Kuva 4.2. *Esimerkkitilanne liikkeen aiheuttamasta sumusta, jossa käyttäjä on liikuttanut päätään nopeasti oikealta vasemmalle. Kasvojen paikantaminen HAAR Cascades-menetelmällä ei onnistu näin epätarkasta kuvasta.*

Silmälasien vaikutus. HAAR Cascades-menetelmä perustuu oppivaan algoritmiin, joten se on mahdollista kouluttaa tunnistamaan kasvoja, joilla on silmälasit tai joilla silmälaseja ei ole. Silmälasit muodostavat kuitenkin suuren haasteen HAAR Cascades-menetelmällä tapahtuvalle kasvojen tunnistamiselle. Silmälaseja on monenlaisia ja erilaiset muotivillitykset tuovat aina uudenlaisia silmälaseja markkinoille. Tästä syystä HAAR Cascades-algoritmia on vaikea saada koulutettua tunnistamaan kaikenlaisia silmälaseja. Silmälasien linssit aiheuttavat myös heijastuksia, jotka tietyissä valaistusolosuhteissa estävät kasvojen löytämisen kuvasta, koska heijastukset piilottavat esimerkiksi silmäkuoppien tummemmat värisävyt kuvasta.

Pään kallistelun ja kääntelyn vaikutus. Mikäli analysoitavassa kuvassa on kasvot, joita on kallistettu jompaan kumpaan suuntaan, suorassa oleviin kasvoihin koulutettu HAAR Cascades-algoritmin kyky löytää kasvot heikkenee. Jonesin ja Violan mukaan heidän nimeään kantava Viola-Jones algoritmi toimii hyvin vain, kun kasvojen pystylinja on kallistettu alle 15 astetta pystylinjasta [9, kappale 4.1]. He myös mainitsevat, että algoritmia ei ole kannattavaa kouluttaa tunnistamaan useisiin eri asentoihin käännettyjä kasvoja, koska tämä vain heikentäisi algoritmin kykyä tunnistaa kasvoja lainkaan [9, luku 1.].

Kallistettujen kasvojen tunnistamiseksi Jones ja Viola ehdottavat kahta erilaista ratkaisua. Näistä ensimmäinen on ottaa käyttöön Rowleyn, Balujan ja Kanaden esittelemä kol-

mivaiheinen kasvojenpaikannusmenetelmä, jossa alkuperäinen neuroverkkoihin perustuva kasvojenpaikannus korvataan Viola-Jones-menetelmällä. Tässä menetelmässä ensimmäisessä vaiheessa käytetään koulutettua neuroverkkoa, jonka ainoa tehtävä on selvittää kulloisenkin analysointi-ikkunan pyörityskulma. Tämän jälkeen analysointi-ikkunan sisällä oleva kuvadata suoristetaan kiertämällä sitä saadun kiertokulman verran vastakkaiseen suuntaan. Tämän jälkeen voidaan hyödyntää suorassa oleviin kasvoihin koulutettua algoritmia kasvojen paikantamiseen. Tämä on kuitenkin Jonesin ja Violan mukaan laskennallisesti raskaampi, kuin heidän kehittämänsä vaihtoehtoinen menetelmä. [9, luku 1.]

Jonesin ja Violan ehdottama tapa paikantaa kallistettuja kasvoja lähtee siitä, että suoria kasvoja tunnistamaan koulutettu algorimi kykenee tunnistamaan kasvot, joita on kallistettu [-15,15] astetta eli yhdellä koulutetulla luokittelijalla saadaan 30°pyöritys katettua. Halutessa tunnistaa mihin tahansa asentoon kallistettuja kasvoja, tarvitaan 360°/ 30°= 12 kappaletta erilaisia koulutettuja luokittelijoita, joilla saadaan kaikki asennot katettua. Jos yksikään luokittelija ei anna positiivista tulosta, kuvassa ei ole kasvoja lainkaan. [9, alakohdat 4.1.1 ja 4.1.2]

Mikäli käyttäjä on kääntänyt päätään paljon kameran edessä, näkyy hänen kasvoistaan vain sivuprofiili kameralle. Tällöin kasvoja edestäpäin tunnistamaan koulutettu luokittelija ei enää löydä kuvasta kasvoja. Tilanteen korjaaminen vaatii, että tunnistusta pitäisi tehdä myös kasvojen sivuprofiileihin koulutetun luokittelijan avulla. Kaikki tällainen moninkertainen algoritmin ajaminen vaatii entistä enemmän laskentatehoa ja heikentää siten mahdollisuuksia käyttää menetelmää tosiaikaisessa kasvojen paikantamisessa.

4.3 Ihoalueiden paikantaminen

Kun tarkastellaan tyypillistä web-kameran näkemää kuvaa (kohta A kuvassa 4.3), havaitaan, että kuvassa näkyvät yleensä käyttäjän kasvot ja ne ovat suurin yhtenäinen ihonvärinen alue. Ihmisen ihonväriä on mahdotonta määritellä muutaman raja-arvon sisällä olevaksi väriavaruuden osaksi, koska joidenkin iho on huomattavasti tummempi kuin toisilla ja punaisuuskin vaihtelee yksilöittäin. Lisäksi valaistusolosuhteet ja kameran tekniikka vaikuttavat siihen, miltä henkilön iho lopulta web-kameran kuvassa näyttää. Näihin muuttuviin tekijöihin sopeutumiseksi tässä työssä ihonvärin paikannusta lähestytään kaksivaiheisesti. Menetelmä perustuu Michael Swainin ja Dana Ballardin esittelemään menetelmään, jolla objekti saadaan paikannettua kuvasta histogrammien avulla [29]. Tässä työssä heidän menetelmäänsä sovelletaan siten, että kasvojen löytämiseksi aluksi käytetään edellä esiteltyä HAAR Cascades-menetelmää. Tällä menetelmällä löydetyistä kasvoista lasketaan käyttäjän ihonvärin histogrammi ja jatkossa histogrammin takaisinprojektiolla saadaan kuvasta esille todennäköisimmät ihoalueet.

Värihistogrammi on tilastollinen jakauma kuvassa olevien pikseleiden väriarvoista. Se saadaan kun kuvan väriavaruus jaetaan äärelliseen määrään värilokeroita ja lasketaan kuinka monta pikseliä kuhunkin lokeroon sijoittuu. Lokeroon sijoitettujen pikselien määrä

Kuva 4.3. *Esimerkkikuvat ihonvärin paikannuksessa tapahtuvasta kuvankäsittelyn vaiheista. A esittää alkuperäistä kamerasta luettua kuvaa. B on ihonvärin histogrammista tehty takaisinprojisoitu kuva. C esittää takaisinprojisoitua kuvaa, josta on poistettu kohinaa mediaanisuotimella. D on lopullinen binäärikuva, josta kasvojen alue paikannetaan.*

kertoo kyseisen värikaistaleen yleisyyden analysoidussa kuvassa. Histogrammin lokeroon sijoitetun arvon voi myös ajatella toimivan todennäköisyytenä, että kuvasta satunnaisesti poimittu pikseli kuuluu lokeron edustamaan värialueeseen. [5, s. 171]

Histogrammin takaisinprojisointi (*histogram backprojection*) tarkoittaa menetelmää, jossa on käytössä aiemmin määritelty histogrammi. Käsiteltävää kuvaa luetaan pikseli kerrallaan ja katsotaan, mihin histogrammin lokeroon kyseinen pikseli kuuluisi. Mikäli pikseli kuuluisi histogrammin yleisimmän värin lokeroon, kirjoitetaan käsiteltään kuvaan valkoinen pikseli. Jos väri kuuluu histogrammin harvinaisimman värin lokeroon, kirjoitetaan kuvaan musta pikseli. Muut lokerot esitetään harmaasävyinä valkoisen ja mustan väliltä. Kun koko kuva on käyty läpi, tuloksena on harmaasävy kuva, joissa on valkoista niissä kohdissa, joissa on histogrammin yleisintä väriä ja mustaa niissä, joissa harvinaisinta.

Tässä työssä histogrammin laskemista varten kasvokuva muunnetaan HSV - väriavaruuteen ja histogrammi lasketaan erikseen H- ja S-kanaville. V-kanava jätettiin pois, koska se sisältää vain värin kirkkauden, jolloin se ei vaikuta varsinaisesti värin sävyyn. OpenCV

```
CvHistogram* createHistogram(IplImage *hsvImage)
{
    // Prepare HSV buffers for histogram analysis.
    IplImage *tmpHueImage = cvCreateImage(cvSize(320, 200), 8, 1);
    IplImage *tmpSatImage = cvCreateImage(cvSize(320, 200), 8, 1);
    IplImage *planes[] = { tmpHueImage, tmpSatImage };
    int histSize[] = { 32, 16 }; // 32 levels for Hue, 16 for Saturation
    float hueRanges[] = {0, 180}; // Hue range is 0...180.
    float satRanges[] = {0, 255}; // Saturation range is 0...255.
    float *hsRanges[] = {hueRanges, satRanges};

    // Copy Hue and Saturation channels to temporary buffers.
    cvCvtPixToPlane(hsvImage, tmpHueImage, tmpSatImage, NULL, NULL);
    CvHistogram *histogram = cvCreateHist(2, histSize, CV_HIST_ARRAY, hsRanges, 1);
    cvCalcHist(planes, histogram, 0, 0);

    cvReleaseImage(&tmpHueImage);
    cvReleaseImage(&tmpSatImage);
    return histogram;
}
```

Kuva 4.4. *Funktio, joka käyttää OpenCV-kirjastoa ja laskee annetusta HSV-väriavaruuden kuvasta histogrammin ja palauttaa sen. Histogrammissa H-kanava on kvantisoitu 32:een tasoon ja S-kanava 16:een.*

tarjoaa valmiit työkalut histogrammin laskemiseksi ja se onnistuu esimerkiksi kuvan 4.4 esittämällä funktiolla.

Kun histogrammin takaisinprojisoinnissa käytetään kasvoista muodostettua histogrammia, saadaan harmaasävykuva, joissa käyttäjän ihonväriä sisältävät alueet esiintyy valkoisina ja muut alueet mustina. Toteutus ei siis tee oletuksia, miltä käyttäjän ihonväri näyttää, vaan se selvitetään joka käyttökerta erikseen HAAR Cascades-menetelmällä paikannetusta kuvasta. Kuten kuvan 4.3 kohdasta B on nähtävissä, takaisinprojisoitu kuva sisältää usein havaittavia määriä kohinaa. Kohina näyttää useimmiten olevan muutaman pikselin kokoisia kirkkaita pisteitä muuten tasavärisessä taustassa. Gonzalez ja Woods esittävät, että jos halutaan pitää kuvan reunat terävinä, mutta vähentää kohinaa, kannattaa käyttää mediaanisuodatinta [5, s. 191]. Tämä on juuri se operaatio, joka kuvalle halutaan tehdä, joten mediaanisuodatus on seuraava operaatio, joka takaisinprojisoidulle kuvalle tehdään. Tämän suodatuksen vaikutus on nähtävissä kuvan 4.3 kohdassa C.

Mediaanisuodatus vähentää kohinaa, ja nyt eri harmaasävyt on selkeästi nähtävissä yksivärisinä alueina. Viimeinen operaatio, joka kuvalle tehdään ennen kasvojen alueen paikannusta on tehdä siitä kaksivärinen. Tämä onnistuu valitsemalla raja-arvo, jota kirkkaammat pikselit kirjoitetaan valkoisiksi ja tummemmat mustiksi. Lopputulos on nähtävissä kuvassa 4.3 kohdassa D.

5 KAMERAKONTROLLERI

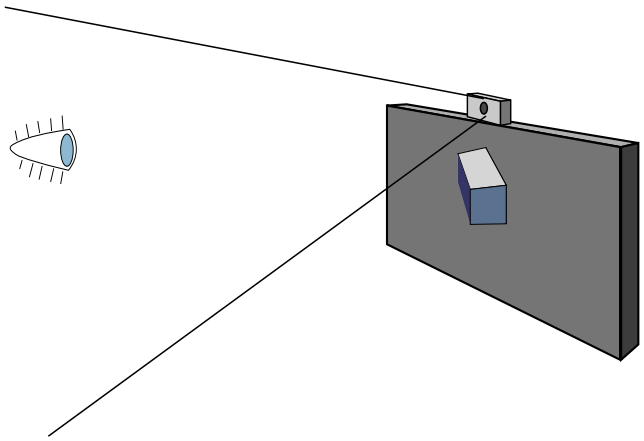
Kamerakontrollerin toteutuksessa on huomioitava kolmen erillisen osa-alueen asettamat vaatimukset. Nämä osa-alueet ovat laitteistolle asetetut vaatimukset, suorituskyvylle asetetut vaatimukset ja toteutustekniikalle asetetut vaatimukset. Laitteistovaatimukset perustuvat tietokoneen ja kameroiden hintatasolle asetettuihin vaatimuksiin. Suorituskyvyn vaatimukset liittyvät tavoiteltuun käyttötarkoitukseen, jotka asettavat asettavat vaatimuksia sille, kuinka nopeasti kasvojen paikannuksen on tapahduttava. Toteutustekniikan vaatimukset liittyvät käytettäviin ohjelmointikieliin ja käyttöjärjestelmiin.

Tässä luvussa käsitellään kamerakontrollerin laitteistovaatimuksia kohdassa 5.1. Kohdassa 5.2 esitellään kamerakontrollerin suorituskyvylle asetetut vaatimukset ja selvitetään eri menetelmien nopeudet suorituskykyyn liittyen. Kohdassa 5.3 esitellään menetelmä, jolla kamerakontrolleri arvioi käyttäjän pään etäisyyttä. Kamerakontrolleri toteutustekniikkaa ja ohjelmointikieltä käsitellään kohdassa 5.4, jonka jälkeen esitetään kamerakontrollerin julkinen rajapinta kohdassa 5.5. Lopuksi arkkitehtuuri esitellään kohdassa 5.6.

5.1 Kamerakontrollerin laitteistovaatimukset

Kamerakontrollerin tehtävänä on käyttää kameraa kuvamaan käyttäjän pään sijaintia ja kuvata pään liikkeet 3D-maailman kameran liikkeiksi. Kun web-kamera sijoitetaan näyttölaitteen yläreunaan, sen avulla saadaan kuvattua riittävän laaja alue näyttölaitteen edestä, jotta käyttäjän pään liikkeet saadaan selvitettyä. Kuvassa 5.1 on esitetty havainnekuva järjestelmän rakenteesta.

Laitteiston puolesta järjestelmä on yksinkertainen, koska se tarvitsee vain web-kameran. Järjestelmän kehitysvaiheessa käytettiin kahta erilaista tietokonekokoonpanoa, joista toinen oli Fujitsu Esprimo P1510-pöytätietokone, johon oli kytketty Creative Live CHAT HD-merkkinen web-kamera. Toinen kokoonpano koostui pelkästä Asus X53E-merkkisestä kannettavasta tietokoneesta, jonka näytön yläreunaan oli valmiiksi integroitu web-kamera. Kumpikin laitteisto kuuluu kirjoitushetkellä halvemman keskitason hinta- ja suorituskykyluokkaan. Tämä on hyvä perusta järjestelmän kehittämiselle, koska tietokoneiden suorituskyky kasvaa jatkuvasti, jolloin kehitetyn kamerakontrolleri kohdelaitteistona voi lähitulevaisuudessa olla myös halvimpien hintaluokkien laitteet. Taulukossa 5.1 esitetään työssä käytettyjen laitteiden tekniset tiedot.



Kuva 5.1. *Järjestelmän laitteistokokoonpano sisältää näytön, kameran ja käyttäjän. Näytöllä näkyy 3D grafiikkaa. Näytön yläpuolelle sijoitettu kamera kuvaa käyttäjää ja paikantaa käyttäjän kasvojen sijaintia kuvassa. Tämän jälkeen ohjelmisto arvioi käyttäjän pään sijainnin 3D-avaruudessa ja päivittää ruudulla näkyvän 3D-grafiikan vastaamaan sitä, kuin kohdetta katsottaisiin arvioidusta sijainnista.*

5.2 Kamerakontrollerin suorituskyvyn vaatimukset

Jotta käyttäjä todella kokisi suuremman syvyysvaikutelman, tulee 3D-maailman kameran liikkeiden seurata käyttäjän liikkeitä riittävän lyhyellä viiveellä. Riippuu käyttäjästä ja käytettävän sovelluksen luonteesta, millainen viive on riittävän lyhyt. Nopeatempoisessa pelissä jo muutaman kymmenen millisekunnin viive on liian pitkä. Sen sijaan verkkaisemmassa 3D-mallinnuksessa parin sadan millisekunnin viiveet saattavat vielä olla hyväksyttäviä.

Tässä työssä kamerakontrollerin päivittymistavoitteeksi asetettiin 10 kuvaa sekunnissa. Tämä nopeus tarkoittaa sitä, että yhden web-kameran ottaman kuvan käsittelyyn on käytettävissä noin 67 millisekuntia. Tämä tavoitenopeus on valittu siksi, että se on riittävän lyhyt, jotta käyttäjälle kyetään tarjoamaan riittävän nopea vaste hänen liikkeidensä ja ruudulla tapahtuvien muutosten välillä. Toisaalta tämä nopeus ei ole liian liian suuri, jotta kaikkea mahdollista laskentatehoa ei tarvitse käyttää videokuvan analysoimiseen vaan laskentatehoa jää myös varsinaisen sovelluksenkin käyttöön. Kyseinen nopeus ei myöskään ylitä käytössä olleiden web-kameroiden videokuvausnopeutta.

Kyetäksemme valitsemaan tekniikat, joilla käyttäjän pään liikkeitä seurataan, meidän tulee ensin selvittää kunkin eri tekniikan tarjoama paras mahdollinen suorituskyky. Alakohdassa 5.2.1 esitetään pelkällä HAAR Cascades-menetelmällä tehtyä kasvojen paikan-

Taulukko 5.1. *Järjestelmän kehityksessä käytettyjen tietokoneiden tekniset ominaisuudet.*

Ominaisuus	**Fujitsu Esprimo**	**Asus X53E**
Prosessori	Intel Pentium G5950, 2 ydintä, 2800MHz	Intel Pentium B960, 2 ydintä, 2200MHz
Muisti	4 Gt	4 Gt
Näytönohjain	NVIDIA GeForce GT220	Intel HD Graphics 3000
Web-kameran resoluutio	Creative, 1280 x 720	Integroitu 640x480

tamisen suorituskykyä. Sen jälkeen alakohdassa 5.2.2 on selvitetty ihonvärin avulla tapahtuvan kasvojen paikantamisen suorituskykyä.

5.2.1 HAAR Cascades-menetelmän suorituskyky

OpenCV-kirjaston HAAR Cascades menetelmällä tapahtuvan kasvojen paikantamisen suorituskykyä mitattiin siten, että tehtiin sovellus, joka lukee web-kamerasta kuvan. Tämän jälkeen käytettiin OpenCV:n `cvDetectObjects()`-funktiota kasvojen löytämiseen, ja samalla mitattiin aika, kauanko funktiolla meni etsinnän tuloksen palauttamiseen. Etsinnässä käytettävät lisäparametrit rajasivat etsinnän löytämään vain yhdet kasvot. OpenCV:n mukana tulee useita kasvojen paikantamiseen sovelutuvia cascades-tiedostoja, mutta tässä työssä käyttöön valittiin `haarcascades_frontalface_alt2.xml`.

Kaikkiaan ajettiin neljä erilaista testiä. Ensimmäinen testi mittasi 640x480-resoluutiolla otetun kuvan tutkimista, jossa oli kasvot löydettävissä. Toinen testi mittasi 320x240-resoluutioon skaalatun kuvan tutkimiseen menevää aikaa, jossa oli kasvot löydettävissä. Kolmas testi mittasi 640x480-resoluutiolla otetun kuvan tutkimista, jossa ei ollut kasvoja. Lopuksi neljäs testi mittasi 320x240-resoluutioisen kuva analysointia, kun siinä ei ollut kasvoja läsnä. Sovellus luki web-kamerasta kuvan, etsi kasvot, kirjasi etsintään käytetyn ajan muistiin ja aloitti silmukan alusta. Silmukka käytiin kunkin testin kohdalla 100 kertaa läpi ja tuloksena saatiin näiden 100:n kuvan analysointiin käytetty yhteisaika. Ohjelmassa oli myös varmistus sille, että mikäli oltiin mittaamassa tilannetta, jossa kasvoja ei löydy, mutta algoritmi löysikin kasvot, niin kyseinen mittaus hylättiin ja tilalle luettiin uusi kuva. Nämä testit ajettiin aiemmin mainitulla Fujitsu Esprimo-tietokonella. Alla olevassa taulukossa 5.2 on esitetty mittauksen tulokset.

Taulukosta voidaan nähdä yksi ilmeinen ongelma, joka kohdataan HAAR Cascades-menetelmää käytettäessä. Kasvojen etsimiseen kuluva aika riippuu siitä, löytyykö kuvasta kasvoja vai ei. Tilanteessa, jossa kasvot löytyvät, etsintäajat ovat käytännössä riitävän hyvät, jotta aiemmin mainittuun päivitysnopeuteen voidaan päästä. Ongelman muodostaakin se, että läheskään aina kuvasta ei ole kasvoja löydettävissä ja näissä tilanteissa kuvan ana-

Taulukko 5.2. *HAAR Cascades-menetelmän suorituskyky kasvojen paikannuksessa. Taulukosta on nähtävissä resoluution vaikutus kasvojen etsintään kuluneeseen aikaan. Ajat ovat millisekunteja per analysoitu kuva.*

	640x480	**320x240**
Kasvot löytyivät	44,64ms	28,39ms
Kasvot eivät löytyneet	470,81ms	80,97ms

lysointiin kuluva aika on liian pitkä. Esimerkiksi aiemmin esitellyt hankaluudet tunnistaa paljon käännettyjä kasvoja saavat aikaan sen, että käyttäjän kallistaessaan päätä liikaa, kamerakontrollerin kyky seurata liikettä heikkenee ja sovellus alkaa hidastella, koska kuvan analysointi kestää niin kauan. Resoluution pienentäminen parantaa tilannetta huomattavasti, mutta silti analysointiajat vaihtelevat liian paljon, jotta tasainen päivitysnopeus olisi tavoitettavissa.

5.2.2 Ihonvärin avulla paikantamisen suorituskyky

Ensimmäinen vaihe ihonvärin avulla paikannuksessa on on kasvojen ihon histogrammin muodostaminen. Kamerakontrolleria tehtäessä kokeiltiin useita erilaisia kvantisointitasojen määriä ja lopulta 16-tasoiset histogrammit kasvokuvan H- ja S-kanavista osoittautuivat toimivimmaksi. Histogrammi laskettiin HAAR Cascades-menetelmällä paikannetuista kasvoista kuvan 4.4(sivu 28) mukaisella funktiolla.

Kun kasvojen alueen histogrammi oli laskettu ja alustava kasvojen sijainti oli saatu selvitettyä, siirryttiin varsinaiseen ihoalueen paikannuksen suorituskykytestiin. Tällöin webkamerasta luettiin kuva, jolle tehtiin seuraavanlaiset operaatiot

1. Kuva skaalattiin kokoon 320x200. Tämä vaihe tehtiin vain testissä, jossa mitattiin skaalatun kuvan analysointinopeutta.

2. Kuva konvertoitiin HSV-formaattiin, jotta ihonväri saadaan paremmin selville värisävyn ja -kylläisyyden avulla.

3. Kuvan H- ja S-kanavat eroteltiin kahdeksi erilliseksi kuvaksi `cvCvtPixToPlane()`-funktion avulla. Tämä vaihe tarvittiin, koska kasvojen histogrammi oli laskettu erikseen H- ja S-kanaville.

4. Käytettiin OpenCV:n `cvCalcBackProject()`-funktiota takaisinprojisoidun kuvan muodostamiseen. Tuloksena on harmaasävykuva, joissa valkoiset kohdat esittävät todennäköisimpiä ihoalueita ja mustat sellaisia, jotka eivät vastaa ollenkaan ihon histogrammia.

5. Kohinan vähentämiseen takaisinprojisoitu kuva suodatettiin mediaanisuodattimella, jonka ikkunan koko oli 7x7 pikseliä.

6. Lopullinen binäärikuva muodostettiin `cvThreshold()`-funktion avulla. Pikselit, joiden arvo on 180 tai enemmän, muutetaan 255:n ja muut 0:aan.

7. Viimeisena vaiheena käytettiin OpenCV:n `cvMeanShift()`-funktiota, jolla lasketaan annetun sijainnin ympäriltä suurimman yhtenäisen värialueen sijainti. Tämä sijainti oli ihonvärin avulla tapahtuvan kasvojen paikannuksen arvio kasvojen sijainnista. Saatu suorakulmio otettiin talteen ja sitä käytettiin seuraavan web-kamerasta luetun kuvan arvaukseksi kasvojen sijainnista.

Suorituskyvyn mittaus laski edellä mainittujen vaiheiden suoritukseen kuluneen ajan millisekunteina. Vastaavalla tavalla kuin HAAR Cascades-menetelmän kohdalla, tässäkin testissä mitattiin 100:aan analyysiin kulunut aika. Nämä tulokset on esitetty taulukossa 5.3. Jälleen ajettiin testit, joissa kasvoalueet löytyivät ja testit, joissa kasvoja ei löydy. Testit ajettiin samalla tietokoneella, jolla taulukossa 5.2 esitetyt HAAR Cascades-menetelmän suorituskykytestit ajettiin.

Kasvojen löytyminen selvitettiin siten, että kun HAAR Cascades-menetelmällä oli saatu kasvojen alustava sijainti selville, seuraavan 100 web-kameralta luetun kuvan ajan käyttäjä oli mahdollisimman liikkumatta kameran edessä ja ihonvärin paikannuksessa saadusta suorakulmiosta tarkistettiin, että suorakulmion keskipiste pysyy alle 10 prosentin sisällä alkuperäisestä sijainnista. Jos siirtymä oli suurempi, kyseinen mittaus jätettiin huomiotta ja jatkettiin lukemalla uusi kuva web-kameralta.

Testit, joissa kasvoja ei löydy, toteutettiin siten, että testiohjelma pysäytettiin viideksi sekunniksi, kun HAAR Cascades-menetelmällä oli saatu alustava kasvojen sijainti selvitettyä. Tämän viiden sekunnin aikana web-kameran linssiin kiinnitettiin musta kartonkipaperi, jonka tarkoituksena oli estää kaiken valon pääsy kameran linssin sisään. Tällä tavalla varmistettiin, että mitään kasvojen ihoalueen väristä ei pääse kameran kuvaan mukaan.

Taulukko 5.3. *Ihonvärin perusteella tapahtuvan kasvoaluiden paikantamiseen kulunut aika keskimäärin sekä 320x200- että 640x480-resoluutiisia kuvia käytettäessä. Aikayksikkönä on millisekunti.*

	640x480	**320x200**
Kasvot löytyivät	46,01ms	17,17ms
Kasvot eivät löytyneet	42,74ms	15,26ms

5.3 Pään sijainnin arviointi

Jotta käyttäjän pään liikkeet näyttölaitteen edessä saadaan jäljitettyä, pitää kameralta luetusta kuvasta kyetä päättelemään käyttäjän pään sijainti kolmiulotteisessa avaruudessa.

Jos pään etäisyyden arviointi jätetään pois ja tyydytään etsimään vain XY-tasossa tapahtuvat pään liikkeet, riittää että löydetään kuvassa 4.3 (sivu 27) esitetystä d-kohdan kasvokuvasta suurimman valkoisen alueen keskipiste. Tämä onnistuu esimerkiksi OpenCV:n `cvMeanShift()`-funktion avulla. Tämän jälkeen käytetään lauseketta 2.15 (sivu 12) selvittämään se 3D-avaruuden suora, jolla kasvojen keskipiste sijaitsee. Koska etäisyyttä ei arvioida, oletetaan kasvojen etäisyydeksi jokin vakio.

Kamerakontrolleriin haluttiin ottaa myös etäisyyden arviointi mukaan, koska mahdollisuus tarjota laajempi näkymä 3D-maailmaan käyttäjän siirtyessä lähemmäs näyttöään lisää virtuaalimaailman uskottavuutta siinä määrin paljon. Alkuperäinen suunnitelma oli, että selvitetään kasvoalueen rajaava suorakaide (*bounding box*) ja tämän suorakaiteen koko olisi mahdollista kuvata etäisyydeksi. Tämä lähestymistapa ei kuitenkaan käytännössä onnistunut, koska rajaavan suorakaiteen määrittäminen web-kameran kuvasta ei onnistunut riittävän luotettavasti. Ongelmaksi muodostui valaistusoloihin sopeutuminen. Kun käyttäjä liikkuu näytön edessä hänen päänsä menee välillä valonlähteen ja webkameran väliin, jolloin kuva muuttuu tummemmaksi. Toisinaan taas naaman kääntäminen valonlähteen suhteen muuttaa naaman ihoalueen valaistuksia niin paljon, että osa kasvojen ihoalueen väreistä siirtyvät pois aluksi arvioidulta ihon värialueelta. Kun ihoalueen paikannuksessa laajennettiin ihonväriksi tulkittavaa värialuetta, mukaan alkoi tulla paljon häiritsevää epätarkkuutta. Esimerkiksi tyypilliset puupintaiset esineet alkoivat tulla mukaan prosessoituihin kuviin, koska ne olivat niin lähellä ihon väriä (katso kuvan 4.3 kohta d). Rajaavan suorakaiteen koko vaihteli kameralta luettujen ruutujen välillä liian paljon, jotta niistä olisi voinut luotettavasti saada pään etäisyyttä selville.

Yksi ratkaisu olisi sijoittaa riittävän hyvä valaistus kameran taakse valaisemaan käyttäjän kasvoja tasaisesti. Tämä ei kuitenkaan ole hyväksyttävä ratkaisu, koska se rajoittaisi järjestelmän käytön vain "täydellisiin"valaistusoloihin. Ratkaisuksi valittiin vihreä nappi, joka kiinnitetään esimerkiksi lakkiin tai kuulokkeisiin, jota käyttäjän on pidettävä päässä. Vaikka ratkaisu ei välttämättä miellytäkään muotitietoisimpia käyttäjiä, niiden avulla kamerakontrollerille kyetään kuitenkin tarjoilemaan se lisätieto, mitä tarvitaan käyttäjän pään etäisyyden arviointiin.

Ensin ihonvärin avulla kuvasta paikannetaan kasvojen keskipiste. Tämän jälkeen kuvasta poistetaan kaikki muut, paitsi värialueet, jotka ovat HSV-väriavaruudessa välillä H=[70..120] ja S=[30,100]. Tämän värialueen todettiin toimivan riittävän hyvin, jolloin kuvaan jää käytännössä vihreät kohteet jäljelle. Koska kasvojen läheisyydessä ei ole muuta vihreää kuin lakkiin kiinnitetty nappi, voidaan olettaa, että ainoa vihreä kohde on nappi, jonka keskipiste saadaan myös selvitettyä `cvMeanShift()`-funktiolla. Kun kasvojen keskipiste ja napin keskipiste on selvillä, voidaan käyttää aiemmin esiteltyä lauseketta 2.17 kasvojen etäisyyden laskemiseen.

5.4 Toteutustekniikan vaatimukset

Kamerakontrollerin toteutuksen keskeisimpänä tavoitteena oli tehdä luokkakirjasto, joka olisi helposti sovitettavissa useisiin erilaisiin 3D-moottoreihin. Vaikka referenssitoteutus tehtiinkin Ogre3D:n avulla rakennetun sovelluksen ympärille, kamerakontrollerissa itsessään ei ole mitään riippuvuuksia Ogre3D:hen.

Totetuskieleksi valittiin C++ lähinnä kahdesta syystä. Ensimmäinen syy C++:n käytölle on se, että monet grafiikkarajapinnat, pelimoottorit ja 3D-moottorit käyttävät joko C tai C++ -kieltä pääasiallisena toteutustekniikkana. Esimerkiksi OpenGL, DirectX, Ogre3D ja Unity [31] tarjoavat kaikki C++ -yhteensopivan rajapinnan, ja C++-pohjainen kamerakontrolleri on helpommin istutettavissa näihin kaikkiin. Toinen tärkeä syy C++:n valinnalle on kuvien prosessoinnissa käytetty OpenCV-kirjasto, joka itsessään on toteutettu C- ja C++ -kielillä. Vaikka OpenCV tarjoaa myös Python-pohjaisen rajapinnan, niin yhdessä grafiikkarajapintojen C++ -yhteensopivuuden kanssa C++ katsottiin parhaaksi kehityskieleksi tähän projektiin.

Kohdealustana olivat Windows 7 pohjaiset tietokoneet, ja kehitysympäristönä toimi Microsoftin Visual C++ 2010 Express Edition [18]. Kyseinen ympäristö valittiin, koska Visual C++ tiedettiin hyväksi työkaluksi ja OpenCV:n ja Ogre3D:n käyttöönotto osoittautui juuri niin suoraviivaiseksi operaatioksi kuin kyseisten kirjastojen mukana tulleet asennusohjeet antoivat ymmärtää.

Kamerakontrollerista ei haluttu tehdä vain Windowsissa toimivaa, joten toteutuksen alustariippumattomuuteen kiinnitettiin huomiota. Toteutuksessa ei käytetä Visual C++:n laajennoksia, vaan kaikki käytetyt tietorakenteet ja luokat perustuvat C++98-standardiin. Tällä rajauksella kamerakontrolleri saadaan toimimaan käytännössä kaikilla alustoilla, joilla OpenCV- ja Ogre3D-kirjastotkin toimivat. Koska C++98-standardi ei tarjoa säikeistystä eikä semaforeja, näiden ominaisuuksien aikaansaamiseksi käytettiin POSIX-standardin [7] määrittelemiä rajapintoja. Visual C++ ei tue POSIX-rajapintoja, joten Windowsia varten käyttöön ladattiin avoimen lähdekoodin pthreads-win32-kirjasto [8].

5.5 Kamerakontrollerin rajapinta

Kamerakontrollerin rajapinta on haluttu pitää yksinkertaisena, jotta sen käyttäminen olisi helppoa. Kontrollerin julkinen rajapinta koostuu kahdesta luokasta, joista toinen on `CameraController` ja toinen on rajapintaluokka `ICameraControllerObserver`. Näiden lisäksi on kamerakontrollerin asetuksien välittämiseen käytettävä `CameraSettings` luokka. Kontrollerin käyttö koostuu seuraavista vaiheista:

1. Toteutetaan `ICameraControllerObserver`-rajapinta omassa sovitinluokassa. Sovitinluokka on 3D-moottorista riippuvainen ja sen tehtävänä on muuntaa kamerakontrollerilta saatavat kamerasijainnit kulloisenkin sovelluksen kameran liikkeiksi. Kuvassa 5.3 on esimerkki tällaisen sovitinluokan toteutuksesta.

2. Instantioidaan `CameraController`-luokka. Useimmiten tämä kannattaa tehdä sovitinluokan sisällä, koska sillä tavalla kamerakontrolleriin liittyvät riippuvuudet saadaan rajattua sovelluksessa vain sovitinluokkaan.

3. Luodaan asetukset sisältävä `CameraSettings`-olio ja käynnistetään kamerakontrolleri kutsumalla `startController()`-funktiota.

4. Kutsutaan `CameraController::update()`-metodia tavoitellun näytönpäivitystaajuuden mukaisen viiveen välein. Tämä funktio lukee viimeisimmän sijaintitiedon kamerakontrollerista ja tekee takaisinkutsun `ICameraControllerObserver` -rajapinnan kautta mikäli kameran sijainti on muuttunut.

5.6 Kamerakontrollerin arkkitehtuuri

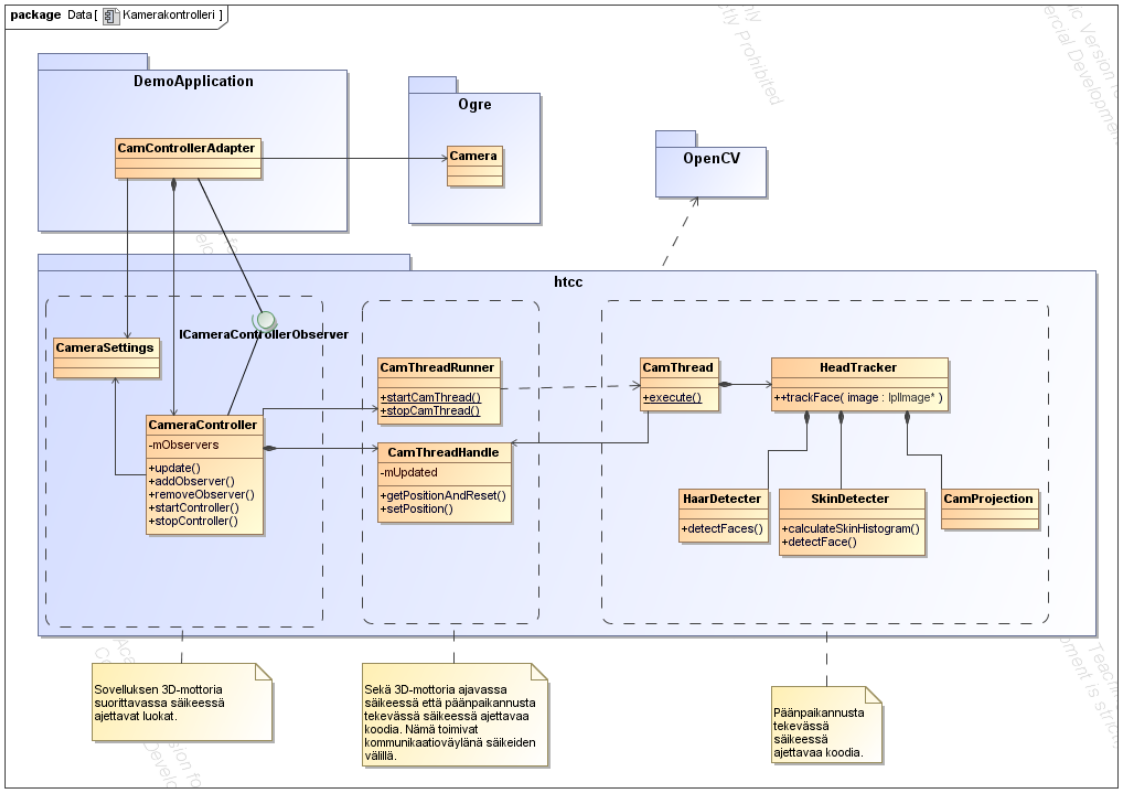
Kuvassa 5.2 on nähtävissä toteutetun kamerakontrollerin luokkakaavio. Kamerakontrollerin julkinen rajapinta koostuu `CameraController`- ja `CameraSettings`-luokista. Lisäksi on `ICameraControllerObserver`-rajapintaluokka. Kaikki muut luokat liittyvät kamerakontrollerin sisäiseen toteutukseen, eivätkä ole näkyvissä kirjaston ulkopuolelle.

`CameraController`-luokan rajapinta, sisältää metodit, joilla kamerakontrolleri konfiguroidaan, käynnistetään ja sammutetaan. Käynnistäminen tarkoittaa tässä yhteydessä sitä hetkeä, jolloin kamerakontrolleri käynnistää web-kameran ja alkaa analysoida sieltä saatavaa kuvadataa. Vastaavasti sammuttaminen tarkoittaa, että web-kamera sammutetaan.

Koska web-kameralta saatavan kuvan analysointi on raskasta, kamerakontrollerin toteutuksessa huomioitiin mahdollisuus ajaa kuvan analysointia rinnakkain varsinaisen 3D-sovelluksen kanssa. Kamerakontrolleri ajaa pään paikannusta eri säikeessä kuin missä 3D-moottoria ajetaan. Julkisen rajapinnan sisältävä `CameraController`-luokka instantioidaan sovelluksen 3D-moottoria ajavan säikeen puolella. Kun kamerakontrolleri käynnistetään, se käynnistää kuvadatan prosessointia varten uuden säikeen. Uuden säikeen sisältämä koodi sijaitsee `CamThread`-luokan sisällä. Näiden kahden säikeen välissä toimii `ThreadHandle`-luokan olio, joka sisältää säikeiden välisessä kommunikoinnissa käytettävät tietorakenteet ja semaforit.

Säikeiden välisestä kommunikoinnista vastaava `ThreadHandle`-luokka käyttää semaforeja eristämään säikeiden välisiä tiedonvaihtotapahtumia. Säikeiden välillä siirtyvä tietomäärä on pyritty pitämään mahdollisimman pienenä. Käytännössä kameran sijaintitieto siirretään kolmen `double`-tyyppisen muuttujan avulla säikeestä toiseen. Aina, kun analysoiva säie saa yhden web-kamerasta luetun kuvan analysoitua ja kasvojen sijainnin määrättyä, se lukitsee semaforin, kirjoittaa sijainnin `ThreadHandle`-luokan jäsenmuuttujiin ja merkitsee `ThreadHandle::mUpdated`-muuttuja arvoon `true` ja vapauttaa semaforin.

Sovelluksen 3D-moottorin puolella luodulla `CameraController`-oliolla on rajapinnassaan `update()`-metodi, jota tulee kutsua tasaisin väliajoin. Sopiva kutsumisväli on



Kuva 5.2. *Komponenttikaavio kamerakontrollerin arkkitehtuurista. Kaaviossa näkyvä DemoApplication edustaa esimerkkisovellusta, johon kamerakontrolleri on integroitu mukaan. Alempi htcc-paketti sisältää varsinaisen kamerakontrollerin toteutuksen. Kaaviosta on jätetty selkeyden vuoksi osa funktioista ja luokista pois.*

sovelluksen tavoitteleman päivitysnopeuden mukaisella taajuudella. Kyseinen funktio tarkistaa, että onko `ThreadHandle::mUpdated`-muuttuja asetettu arvoo `true`. Mikäli on, lukitaan edellä mainittu semafori, luetaan kolme käyttäjän pään sijainnin kertovaa koordinaattia, asetetaan `mUpdated` arvoon `false` ja vapautetaan semafori. Lopuksi `update()`-metodi tiedottaa asetettua kuuntelijaa `ICameraControllerObserver` -rajapinnan kautta. Toteuttu kuuntelija voi sitten liikuttaa sovelluksen 3D-moottorin kameran sijaintia kamerakontrollerin kertoman sijainnin mukaiseksi.

Sovitinluokan toteuttaminen on tarpeen, jotta kamerakontrollerin saa toimimaan monenlaisten 3D-moottorien kanssa. Kamerakontrolleri ei tiedä, millainen 3D-moottori on käytössä vai onko mitään. Se vain antaa koordinaatteja ulos niin tiuhaan kuin pystyy ja se on varsinaisen sovitinluokan tehtävä skaalata kuuntelijarajapinnan kautta tulevat koordinaatit oikean tyyppisiksi. Sovitinluokan toteutus ei välttämättä ole monimutkaista. Kuvassa 5.3 on esimerkki sovitinluokasta, joka sovittaa kamerakontrollerin Ogre3D-moottorin kanssa. Tuloksena on kamera, joka liikkuu käyttäjän pään mukaisesti mutta pitää katseen kohdistettuna 3D-maailman origoon.

```
class HTSimpleAdapter : public htcc::ICameraControllerObserver
{
    Ogre::Camera *mCamera;
    htcc::CameraController *mController;

public:
    HTSimpleAdapter(Ogre::Camera *cam) :
        mCamera(cam),
        mController(new htcc::CameraController)
    {
    }

public: // from ICameraControllerObserver
    virtual void cameraPositionChanged(double x, double y, double z,
                                       double deltaX, double deltaY, double deltaZ)
    {
        mCamera->move(Ogre::Vector3(1.9*(-deltaX), 1.9*(-deltaY), 1.9*(-deltaZ)));
        mCamera->lookAt(Ogre::Vector3(0,0,0));
    }

public:
    void update() { mController->update(); }

    void start() {
        htcc::CameraSettings camSettings;
        mController->addObserver(this);
        mController->startController(camSettings);
    }
    void stop() {
        mController->removeObserver(this);
        mController->stopController();
    }
};
```

Kuva 5.3. *Esimerkki sovitinluokasta. Luokka instantioi `CameraController`-luokan ja toteuttaa `ICameraControllerObserver`-kuuntelijarajapinnan. Kuuntelijarajapintaa kutsuttaessa se liikuttaa Ogre3D:n kameraa.*

6 ARVIOINTI

Kamerakontrollerin arviointi tapahtuu pääosin subjektiivisilla mittareilla. Suorituskyvyn osalta on kuitenkin mahdollista saada mitattua dataa ja nämä mittaukset toimivat pohjana kamerakontrollerin arvioinnille. Tässä luvussa käsitellään ensin kamerakontrollerin mittaustuloksia kohdassa 6.1. Sen jälkeen kohdassa 6.2 arvioidaan, kuinka hyvin kamerakontrolleri täytti asetetut vaatimukset. Lopuksi kohdassa 6.3 ehdotetaan aiheita kamerakontrollerin jatkokehitykseksi.

6.1 Kamerakontrollerin suorituskyvystä

Valmiille kamerakontrollerille tehtiin joukko suorituskykyä mittaavia testejä, joiden tarkoituksena on tarjota dataa kamerakontrollerin arviointia varten. Mitattuja ominaisuuksia ovat prosessorikuorma, päivitysnopeus ja vaikutus 3D-moottorin piirtonopeuteen. Prosessorikuorma selvitettiin seuraamalla prosessorin kuormituksen muuttumista, kun kamerakontrolleri käynnistetään. Päivitysnopeuden selvittämisessa mitattiin, kuinka monta kertaa sekunnissa kamerakontrolleri kykenee antamaan uuden sijainnin. Kamerakontrollerin vaikutus 3D-moottorin piirtonopeuteen selvitettiin siten, että katsotaan näkymän keskimääräinen piirtonopeus ennen kamerakontrollerin käynnistymistä ja piirtonopeus kamerakontrollerin käynnistämisen jälkeen.

Mittausvaiheessa huomattiin, että työssä käytetyn Creativen web-kameran kuvanopeus riippui suuresti ympäristön valaistuksesta. Normaalissa huoneenvalossa se kykeni tarjoamaan hyvälaatuista kuvaa, mutta vain noin 10 kuvaa sekunnissa. Kun kameraa käytettiin päivänvalossa ikkunan lähellä se kykenee luvattuun 30 kuvaan sekunnissa. Koska tällä on vaikutusta kamerakontrollerin käyttäjän kokemaan vasteaikaan ja toisaalta myös laskentatehon tarpeeseen, molemmat tapaukset mitattiin erikseen.

Taulukossa 6.1 olevista tuloksista näkee Fujitsu-tietokoneen osalta, että heikommassa valaistuksessa kamerakontrollerin päivitysnopeus on alhaisempi kuin kirkkaassa valossa. Tämä johtuu siitä, että heikossa valaistuksessa kameran kuvakenno tarvitsee pidemmän ajan kuvan ottamiseen, ja kontrolleri ei pysty lukemaan kuvia kameralta niin nopeasti kuin se pystyisi niitä prosessoimaan. Hyvässä valaistuksessa kamerasta kyetään lukemaan useampia kuvia sekunnissa jolloin sekuntia kohden on myös enemmän prosessoitavaa ja tästä syystä hyvä valaistus kasvattaa myös prosessorikuorman 62%:sta 70%:iin.

Mielenkiintoinen yksityiskohta on, että vaikka kamerakontrolleri vaatii hyvässä valaistuksessa enemmän prosessoriaikaa kuin heikossa valossa, valaistuksella ei ole vaikutus-

Taulukko 6.1. *Kamerakontrollerin suorituskyvyn mittaustulokset. Taulukossa on nähtävissä arvot tilanteissa, joissa kamerakontrolleri ei ole käynnissä tai joissa se on käynnissä ja valaistus on joko normaali huoneenvalaistus tai tavallista kirkkaampi valaistus.*

	CPU-kuorma (%)	Päivitysnopeus (sijainteja/sekunti)	Piirtonopeus (ruutuja/sekunti)
Fujitsu Esprimo			
Ei käynnissä	57%	-	302
Käynnissä (huonevalo)	62%	6,7	221
Käynnissä (valoisassa)	70%	15,7	220
Asus X53E			
Ei käynnissä	52%	-	314
Käynnissä (huonevalo)	61%	9,8	286
Käynnissä (valoisassa)	67	13,4	262

ta 3D-moottorin päivitysnopeuteen. Tämä johtuu kuitenkin siitä, että kamerakontrolleria varten käynnistettävä säie vaatii oman osansa prosessoriajasta, ja se on pois 3D-moottoria suorittavan säikeen suoritusajasta. Toisaalta esimerkkisovelluksen 3D-moottoria suorittava säie käyttää kaiken tarjolla olevan prosessoriajan, joten se varaa yhden prosessoriytimeltä aina kaiken sille annettavan ajan.

6.2 Arviointi

Jos käyttäjän kasvojen etäisyyden arviointi jätetään huomiotta, kasvojen paikantaminen saadaan saada riittävän tarkaksi käytännön käyttökohteita ajatelleen. Käyttäjän liikkuessa eri suuntiin kameran edessä 3D-maailman kamera liikkuu vastaavasti ja käyttäjän ollessa paikallaan näkymä ei elä häiritsevästi. Mittausvirheistä aiheutuva vapina saatiin hyväksyttäviin rajoihin keskiarvoistamalla sijaintia kolmen edellisen sijainnin avulla. Tavoitteena oli 10 sijaintipäivitystä sekuntia kohden, ja hyvän web-kameran kanssa kamerakontrolleri yltää siihen. Valitettavasti vaatimattoman web-kameran kanssa huoneenvalossa kamerakontrolleri ei yllä tavoiteltuun nopeuteen.

Pään etäisyyden arvioinnissa ei päästy tavoiteltuun tarkkuuteen. Tämä johtui siitä, että kameralta luettavista peräkkäisissä kuvissa kasvojen keskipisteeseen kuin myös vihreän napin sijainnin määrityksessä tapahtuu aina mittausvirhettä. Lisäksi etäisyyden määrityksessä tulee vastaa se ikävä piirre, että kameraan nähden suuret etäisyyden muutokset näkyvät 2D-kuvassa vain erittäin pieninä näkyvinä muutoksina. Kasvojen keskipisteen ja vihreän napin keskinäisen etäisyyden arviointi tehtiin 2D-kuvasta, jolloin väkisin mu-

kaan tulevat pienet mittausvirheet näkyvät suurina muutoksina lasketussa kasvojen 3D-sijainnissa. Vain tilanteessa, joissa kasvot ovat erittäin lähellä kameraa, etäisyyden laskenta saadaan kelvollisen tarkaksi. Tämä taas on ristiriidassa sen vaatimuksen kanssa, että kamerakontrollerin käytön tulisi olla käyttäjälle mahdollisimman vaivatonta. Tarkka etäisyyden määrittäminen vaatisi lisäselvityksiä. Tämän selvän ongelman pienentämiseksi kamerakontrolleri toteutettiin siten, että mikäli vihreää nappia ei löydetä kuvasta, etäisyytenä käytetään tiettyä kiinteää etäisyyttä. Tämän ansiosta kamerakontrolleri on käyttökelpoinen vaikka etäisyyden määrittämisvaihe jäisikin tekemättä.

Kolmantena vaatimuksena oli, että kamerakontrollerin integroiminen olemassaolevaan 3D-moottoriin tulee olla helppoa. Vaikka ”helppo” on erittäin subjektiivinen käsite, työn tuloksena syntynyt dynaamisesti linkitettävä luokkakirjasto on helposti sovitettavissa olemassa olevaan 3D-sovellukseen. Tämä onnistuu sovitinluokan avulla, joka instantioi ja käynnistää kamerakontrollerin, toteuttaa kamerakontrollerin kuuntelijarajapinnan ja osaa muuntaa kuuntelijarajapinnan kautta saadut koordinaatit kulloisenkin sovelluksen kameran sijainniksi.

Kasvojen keskipisteen ja vihreän napin paikantamiseen käytettiin värirajausta. Tämän menetelmän huono puoli on se, että web-kameran kuvaan saattaa hyvinkin päätyä muitakin ihonvärisiä tai vihreitä alueita, kuin kasvojen iho ja vihreä nappi. Tällaiset tilanteet aiheuttavat ongelmia ja kamerakontrolleri saattaa hukata kasvot kokonaan alkaa seurata väärää kohdetta. Myös monien kameroiden tukema automaattinen valkotasapainon säätäminen aiheutti ongelmia ja se oli pakko ottaa kameran asetuksista pois päältä. Tämän jälkeenkin valaistus voi olla niin epätasainen, että kasvojen ollessa tietyssä asennossa ne kyetään paikantamaan, mutta toiseen asentoon käännettynä värisävyt muuttuvat ja kamerakontrolleri hukkaa kasvot.

6.3 Jatkokehitysajatuksia

Koska kasvojen keskipisteen seuraaminen onnistui suhteellisen tarkasti, yksi selkeä jatkotutkimuksen kohde on mahdollisuus käyttää stereokameraa etäisyystiedon laskemiseksi. Kun kahden kameran keskinäinen etäisyys ja kulma tiedetään, siitä pitäisi olla mahdollista selvittää kummankin kameran kuvassa näkyvän kohteen etäisyys kameroista. Analysoitavan kuvadatan määrä kuitenkin kaksinkertaistuu, jolloin tarvitaan tehokkaampia menetelmiä kuvien esiprosessointiin ja mahdollisesti säikeistystä pitää kehittää tehokkaammaksi.

Tässä diplomityössä toteutua kamerakontrolleria voisi luontevasti laajentaa katseen seuraamisen ominaisuuksilla. Tällöin käyttäjän liikuttaessa silmiä ja kohdistaessaan katsettaan eripuolille näyttöä, sovellus tietäisi aina, mihin kohtaan näyttöä käyttäjä katsoo. Nykyään pöytäkoneiden näyttökoot ovat kasvaneet huomattavan suureksi, ja alle 22 tuuman näyttöjä ei kaupoissa enää paljoa näy. Tällaiset näytöt ovat jo niin suuria, että käyttäjän katse ei voi olla kohdistuneena kuin pieneen osaan näytön alaa kerrallaan. Sovellusten käytettävyyttä voitaisiin parantaa kun tiedetään mihin käyttäjä katsoo ja vain tällä alueel-

la esitetään muuttuvaa tietoa. Toisaalta käyttäjän huomio voitaisiin varastaa esittämällä animoitua sisältöä sellaisella näytön alueella, joka sijaitsee näkökentän reunoilla.

Jos käyttäjällä on käytössään aitoon 3D-näkymään kykenevä näyttö, se olisi mahdollista muuttaa ikäänkuin oikeaksi ikkunaksi virtuaalimaailmaan. Tällöin näytöllä näkyvä grafiikka eläisi aidosti käyttäjän pään liikeiden mukaisesti. Tässä diplomityössä ei selvitetty menetelmiä, joilla käyttäjän pään liikkeet ja ruudulla näkyvä 3D-grafiikka saadaan vastaamaan toisiaan täsmällisesti vaan tyydyttiin kokeilemalla saatuihin X-, Y- ja Z-arvojen kertoimiin, joilla näkymän liikkeet saatiin vastaamaan pään liikkeitä. Jatkossa kannattaisi tutkia, kuinka 3D-maailman koordinaatisto saadaan vastaamaan kameran projektiomatriisin arvoja, jotta täydellisen tarkka yhteys kasvojen liikkeiden ja 3D-grafiikan katselukulman välillä saadaan aikaiseksi.

7 YHTEENVETO

Tosiaikaisella kasvojen sijainnin paikantamisella on mahdollista tuoda lisää syvyyttä 3D-grafiikkaa käyttäviin sovelluksiin, koska näytöllä näkyvää sisältöä voidaan katsella helposti eri suunnista vain päätä liikuttamalla. Esimerkiksi 3D-mallinnuksessa työstettävää mallia voisi katsella eri suunnista vain näytön edessä liikkumalla tai peleissä esteiden ohi olisi mahdollista kurkistaa vain päätään liikuttamalla.

Tämän diplomityön yhteydessä toteutettiin C++-ohjelmointikielellä kamerakontrollerin sisältävä luokkakirjasto, jonka avulla on mahdollista seurata käyttäjän pään liikkeitä. Kamerakontrolleri käyttää OpenCV-kirjaston Viola-Jones -pohjaista hahmontunnistusalgoritmia käyttäjän kasvojen paikantamiseen. Kyseinen algoritmi sisälsi kuitenkin puutteita, joiden vuoksi se ei sopinut ainoaksi kasvojen paikannukseen käytettäväksi menetelmäksi. Esimerkiksi silmälasien aiheuttamat heijastukset ja pään kallistaminen saivat aikaan ongelmia kasvojen paikantamisessa. Hahmontunnistuksesta oli kuitenkin hyötyä, koska sen avulla saadaan selville kasvojen sijainti kamerakontrollerin käynnistyessä. Aluksi paikannetusta kasvojen alueesta saadaan selvitettyä käyttäjän ihonväri sen hetkisessä valaistuksessa, ja samalla ihonväristä saadaan laskettua värihistogrammi. Toinen kasvojen paikantamiseen käytettävä menetelmä on ihonvärin avulla tehtävä kasvojen etsiminen. Varsinaisessa ihoalueiden paikantamisessa käytetään histogrammin takaisinprojektoimista, jolloin vain ihonväriset alueet jää kuvaan jäljelle ja tämän alueen keskipiste toimii kasvojen keskipisteenä. Ihonvärin avulla tapahtuvat värirajauksen yhteydessä kuitenkin huomattiin, että ihonväriset alueet käyttäjän selän takana tulivat värirajattuun kuvaan ja osaltaan heikensivät paikannuksen tarkkuutta.

Kasvojen lisäksi kuvasta etsitään myös kirkkaan vihreän väristä pienempää kohdetta kasvojen keskipisteen yläpuolelta. Käytännössä kirkkaan värinen kohde on noin 3 cm halkaisijaltaan oleva nappi, joka kiinnitetään hattuun tai kuulokkeisiin, joita käyttäjän on pidettävä päässään, mikäli etäisyyden arviointia halutaan hyödyntää. Kamerakontrollerilla on kaksi toimintamoodia sen mukaan, löydetäänkö vihreää nappia vai ei. Jos löydetään, niin kasvojen keskipisteen ja napin välisen etäisyyden avulla arvioidaan kasvojen etäisyys web-kamerasta. Jos nappia ei löydy, kasvojen etäisyydeksi oletetaan etukäteen valittu etäisyys.

Kamerakontrolleri ajaa kasvojen paikannusta erillisessä säikeessä ja etenkään moniydinprosessorissa se ei oleellisesti hidasta 3D-moottoria ajavan säikeen toimintaa. 3D-moottoria varten tulee tehdä sovitinluokka, joka toteuttaa kamerakontrollerin kuunteli-

jarajapinnan ja konvertoi kamerakontrollerilta saadut kasvojen paikkatiedot käytettävän 3D-moottorin kameran parametreiksi. Kamerakontrolleri ei ota kantaa käytettävään 3D-moottoriin.

Käytännössä web-kameran tulee olla hyvä, jotta se kykenee tarjoamaan riittävän kuvanopeuden. Vaatimattomilla kameroilla heikossa valaistuksessa kameralta saatiin liian hitaasti kuvadataa, jotta kamerakontrollerin avulla olisi saatu pehmeä kasvojen seuranta aikaiseksi. Suurin osa kameroista sisältää myös automatiikkaa valkotasapainon ja kirkkauden säätämiseen ja tämä automatiikka aiheuttaa ongelmia kamerakontrollerille ja saa sen hukkaamaan joko kasvojen keskipisteen tai vihreän napin. Valkotasapainon muutuminen vaikuttaa myös kuvassa olevien ihoalueiden värisävyyn, jolloin ne eivät ole enää samalla väriavaruuden kaistalla kuin aiemmin. Myös kasvojen keskipisteen ja vihreän napin paikannuksessa tulee usein muutamien pikseleiden virheitä, joka kuitenkin näkyy häiritsevänä 3D-näkymän kameran liikkumisen eteen ja taakse. Hyvässä valaistuksessa kamerakontrolleri kuitenkin pääosin tekee tehtävänsä.

Tämän diplomityön yhteydessä tuli selkeästi esille monia niistä haasteista, joiden vuoksi kasvojen paikannus yhden kameran avulla ei ole arkipäivää. Käyttäjän kasvoista pitäisi kyetä löytämään useampia kuin yksi jäljitettävä kohde, jotta käyttäjän pään kolmiulotteista sijaintia voisi arvioida. Lisäksi on mahdollista, että kameran ottamiin kuviin tulee esimerkiksi useita kasvoja tai taustalla olevia kasvojen kaltaisia kohteita, jolloin pitäisi jollain tavalla kyetä löytämään ne kasvot, joita jäljitetään. Myöskään ihonvärin avulla tehtävä suodatus ei tarjoa luotettavaa menetelmää paikannukselle, koska ihmisen ihonväri riippuu yksilön ominaisuuksien lisäksi valaistuksesta. Laskentatehoa ei ole rajattomasti käytettävissä, joten yhtä web-kameralta saatavaa kuvaa ei voi kovin kauaa analysoida. Tämä diplomityö voi toimia hyvänä pohjana näyttäessään, että yhdelläkin web-kameralla voidaan paikantaa kasvoja ja liikuttaa kameraa vastaavasti. Lisäksi tässä työssä tulee esille myös monet niistä haasteita, jotka on ratkaistava ennen kamerakontrollerin tuotteistamista.

LÄHTEET

[1] Benstead, L., Astle, D., Hawkings, K. 2009. Beginning OpenGL Game Programming. 2nd edition. Course Technology. Boston, MA, USA. 319 p.

[2] Cheng, H.D., Jiang, X.H., Sun, Y., Wang, J. 2001. Color image segmentation: advances and prospects. The journal of the pattern recognition society. Vol 34 (2001). Elsevier Ltd. pp. 2259-2281.

[3] Christie, R. M. Colour Chemistry. Royal Society of Chemistry. 2001. 218 p.

[4] Finney, K. Advanced 3D Game Programming. Boston, MA, USA. Course Technology. 616 p.

[5] Gonzalez, R., Woods, R. Digital Image Processing. USA 1993. Addison-Wesley. 716 p.

[6] Hashimoto, A. Visual Design Fundamentals: A Digital Approach. Hingham, MA, USA. 2003. Charles River Media. 365p.

[7] IEEE Std 1003.5b-1996. IEEE Standard for Information Technology- POSIX R ADA Language Interfaces- Part 1: Binding for System Application Program Interface (API)- Amendment 1: Realtime Extensions. American National Standard (ANSI). 1996. 529 p.

[8] Johnson, R. pthreads-win32 - a POSIX threads library for Microsoft Windows. Viitattu 9.9.2012. http://sources.redhat.com/pthreads-win32/

[9] Jones, M., Viola, P. Fast Multi-view Face Detection. TR2003-96. August 2003. Mitsubishi Electric Research Laboratories. Viitattu 15.7.2012. http://www.merl.com/papers/docs/TR2003-96.pdf

[10] Jones, M., Viola, P. Rapid Object Detection using a Boosted Cascade of Simple Features. Computer Vision and Pattern Recognition. 2001. pp. I-511 - I-518

[11] Khronos Group. OpenGL - The Industry Standard for High Performance Graphics. Viitattu 6.11.2012. http://www.opengl.org/

[12] Korkama, T. 2010. Varjostinohjelmointi. Seminaaritutkielma. Helsingin yliopisto, Tietojenkäsittelytieteen laitos. Viitattu 5.11.2012. http://www.cs.helsinki.fi/u/tkorkama/Seminaari.pdf

[13] Lengyel, E. 2003. Mathematics for 3D Game Programming and Computer Graphics. 2nd ed. Hingham, USA. Charles River Media. 569 p.

[14] Luebke, D., Humphreys, G. How GPUs Work. University of Virginia. Viitattu 7.11.2012. http://www.cs.virginia.edu/~gfx/papers/pdfs/59_HowThingsWork.pdf

[15] Marucchi-Foino, R. 2012. Game and Graphics Programming for iOS and Android with OpenGL ES 2.0. Wrox Press. Saatavissa: http://common.books24x7.com/toc.aspx?bookid=45960

[16] McReynolds, T., Blythe, D. 2005. Advanced Graphics Programming Using OpenGL. San Francisco. Morgan Kaufman. 672 p.

[17] Microsoft. Direct3D. Viitattu 6.11.2012. http://msdn.microsoft.com/en-us/library/windows/desktop/hh309466(v=vs.85).aspx

[18] Microsoft. Visual C++ 2010 Express. Viitattu 9.9.2012. http://www.microsoft.com/visualstudio/en-us/products/2010-editions/visual-cpp-express

[19] Niskanen, J. Konenäköön perustuva eleohjaus mobiililaitteella. 2012. Diplomityö. Tampereen teknillinen yliopisto. Tieto- ja sähkötekniikan tiedekunta.

[20] OGRE. OGRE - Open Source 3D Graphics Engine. Viitattu 5.11.2012. http://www.ogre3d.org

[21] OGRE. Licensing - OGRE. Viitattu 5.11.2012. http://www.ogre3d.org/licensing

[22] OGRE. OGRE Manual v1.8. Viitattu 5.11.2012. http://www.ogre3d.org/docs/manual/manual_18.html#Declaring-Vertex_002fGeometry_002fFragment-Programs

[23] Opetushallitus. Etälukio - Pitkä matematiikka - Funktiot ja yhtälöt 1. Viitattu 16.10.2012. http://www02.oph.fi/etalukio/pitka_matematiikka/kurssi1/maa1_teoria3.html

[24] Panin, G. 2011. Model-based Visual Tracking: the OpenTL Framework. John Wiley & Sons. 318 p.

[25] Parrish, K. 2011. Carmack: Direct3D now better than OpenGL. Tom's Hardware. Viitattu 5.11.2012. http://www.tomshardware.com/news/john-Carmack-DirectX-OpenGL-API-Doom,12372.html

[26] Rezaei, M., Klette, R. 3D Cascade of Classifiers for Open and Closed Eye Detection in Driver Distraction Monitoring. Coputer Analysis of Images and Patterns 14th Internation Conference, CAIP 2011, Seville, Spain, August 29-31, 2011, Proceedings, Part II. Springer-Verlag. 2011. Berliini.

[27] Sherrod, A. 2008. Game Graphics Programming. Boston, MA, USA. Course Technology. 673 p.

[28] Sherrod, A., Jones, W. 2012. Beginning DirectX 11 Game Programming, 2nd edition. Course Technology. 385 p.

[29] Swain, M.J, Ballard, D.H. Indexing via color histograms. Proceedings, Third International Conference on Computer Vision. 1990. sivut 390-393. Saatavissa: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=139558&tag=1

[30] Tremblay, T. 3D Basics - Graphics Programming and Theory. GameDev.net. Viitattu 25.10.2012. http://www.gamedev.net/page/resources/_/technical/graphics-programming-and-theory/3d-basics-r673

[31] Unity Technologies. UNITY: Game Development Tool. Unity Techonologies. San Francisco, USA. Viitattu 29.8.2012. http://unity3d.com/unity/

[32] Willow Garage. OpenCV. Viitattu 10.6.2012. http://www.willowgarage.com/pages/software/opencv

[33] Zerbst, S., Duevel, O. 2004. 3D Engine Programming. Course Technology Crisp. Boston, USA. 896 p.