Teemu Lantiainen

REAALIAIKAISTA KAMERAN- JA VIDEONKÄSITTELYÄ JAVASCRIPTILLA

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma

Marraskuu 2013

MIKKELIN AMMATTIKORKEAKOULU
Mikkeli University of Applied Sciences

KUVAILULEHTI

MIKKELIN AMMATTIKORKEAKOULU Mikkeli University of Applied Sciences	**Opinnäytetyön päivämäärä** **29.11.2013**
Tekijä(t) Teemu Lantiainen	**Koulutusohjelma ja suuntautuminen** Tietojenkäsittelyn koulutusohjelma

Nimeke

Reaaliaikaista kameran- ja videonkäsittelyä JavaScriptilla

Tiivistelmä

Kameran hallinta on jo pitkään ollut web-kehityksen tavoitelluimpia ominaisuuksia. Tähän asti kehittäjät ovat joutuneet käyttämään erilaisia kolmannen osapuolen selainlisukkeita, mutta nykyään jo useat selaimet tukevat getUserMedia-nimistä metodia, joka mahdollistaa kameran hallinnan puhtaalla JavaScriptillä ilman ylimääräisiä lisukkeita. Myös kuvien ja videoiden reaaliaikaisen käsittelyn mahdollisuudet selaimessa ovat kasvaneet canvas-elementin kehityksen myötä. Canvas-elementin avulla on mahdollista manipuloida videoita ja kuvia aivan kuvapistetasolta lähtien.

Koska getUserMedia liittyy uusiin laiteläheisiin rajapintoihin, esittelen työssä hieman erilaisia JavaScript-rajapintoja, joiden avulla kehittäjät voivat hyödyntää laitteiden tarjoamaa dataa verkkosovelluksissaan. Työn tutkimusongelmana on, kuinka laitteen kameraa hallitaan puhtailla selainohjelmointitekniikoilla ja kuinka sitä voidaan hyödyntää verkkosovelluksessa. Käytännön toteutuksena teen mikkeliläiselle Tertin kartanolle virtuaalisen puutarhakierros-sovelluksen, jossa hyödynnetään getUserMedia-metodia ja QR-koodeja, sekä reaaliaikaista videonkäsittelyä canvas-elementtien avulla.

Työn tulokset olivat lupaavia, sillä kameran videokuvan toiston suorituskyky oli hyvä eri selaimilla ja eri tehoisilla laitteilla. Mobiililaitteilla ilmeni ongelmia videon manipuloinnin suorituskyvyn kanssa, mutta sitä on mahdollista optimoida paljonkin. Selaintuen tulevaisuuskin näyttää valoisalta, sillä työtä aloittaessa mobiilipuolelta vain Googlen Chrome-selain tuki getUserMedia-rajapintaa, mutta työn valmistuttua myös Mozilla oli lisännyt tuen omaan Firefox-selaimeensa. Eri selainten tuki ja toteutukset voivat kuitenkin erota toisistaan huomattavasti, jonka yhtenäinen standardi toivottavasti muuttaa.

Asiasanat (avainsanat) JavaScript, canvas, video. kamera		
Sivumäärä 46	**Kieli** Suomi	**URN**
Huomautus (huomautukset liitteistä)		
Ohjaavan opettajan nimi Janne Turunen	**Opinnäytetyön toimeksiantaja** Tertin kartano	

DESCRIPTION

MIKKELIN AMMATTIKORKEAKOULU
Mikkeli University of Applied Sciences

Date of the bachelor's thesis
29 November 2013

Author(s)
Teemu Lantiainen

Degree programme and option
Business information technology

Name of the bachelor's thesis

Capturing camera feed and real-time manipulation of video with JavaScript

Abstract

The concept of capturing live audio and video has always been one of the most sought-after features of web development. For years web developers have had to rely on third-party plugins like Flash and Silverlight to make it possible, but the growth of the HTML5 standard has brought several new JavaScript APIs (Application Programming Interface) giving web developers the ability to access device hardware. One of these is called getUserMedia which gives access to the device's camera and microphone without any third-party plugins. Real-time manipulation of video and image has also improved with the development of HTML canvas element and APIs related to it. They allow developers to manipulate videos and images on the pixel level.

The purpose of this thesis was to find out if it was possible to capture video with pure JavaScript, and if it was, how it could be used in web applications. The main focus was on getUserMedia API and video manipulation with canvas element as I tried to include some augmented reality-like features into the application. Since getUserMedia is one of the APIs giving access to device hardware, I introduced several of them in Chapter 2. As the practical part of my thesis I created a virtual garden tour guide application to Tertti Manor, a tourism company located in Mikkeli, Finland. The application makes use of getUserMedia API, QR codes and real-time video manipulation with canvas elements.

The results of this thesis were promising. The performance of video feed provided by getUserMedia was good across different mobile and desktop platforms and browsers. However, the performance of real-time video manipulation on mobile devices turned out to be not so good. It is good to note though, that I did not do optimization that much. The future of browser support for getUserMedia also seems bright. At the beginning of this thesis Google's Chrome browser was the only one to support it on mobile devices, but during the work Mozilla added support for it in their Firefox browser. However, support and implementations can vary quite a lot between different browsers at this point until the final official standard arrives.

Subject headings, (keywords) JavaScript, canvas, video, camera		
Pages 46	**Language** Finnish	**URN**
Remarks, notes on appendices		
Tutor Janne Turunen	**Bachelor's thesis assigned by** Tertti Manor	

SISÄLTÖ

1 JOHDANTO

Verkko-ohjelmointitekniikat ovat kehittyneet viime vuosina hurjasti. Monia ennen pelkästään käyttöjärjestelmäkohtaisilla ohjelmointikielillä toteutettavissa olleita ominaisuuksia on siirtynyt ja siirtyy koko ajan myös selainohjelmoinnin puolelle, kun erilaiset uudet standardit ja JavaScript-rajapinnat syntyvät ja kehittyvät. W3C ja sen rinnalla toimivat erilaiset työryhmät kehittävät jatkuvasti HTML5-standardia ja sen rinnalla mm. erilaisia laiteläheisiä JavaScript-rajapintoja, jotka mahdollistavat esimerkiksi laitteen valosensorin datan lukemisen. Isot, verkko-ohjelmointitekniikoita käyttävät palvelut ja sovellukset kuten Mozillan Firefox OS, vauhdittavat osaltaan tekniikkojen kehitystä. Firefox OS on mobiilikäyttöjärjestelmä, jonka sovellukset on luotu käyttäen tavallisia verkkosivujen luomiseen käytettäviä tekniikoita, kuten HTML-merkkauskieltä, CSS-tyylimäärityksiä ja JavaScript-ohjelmointikieltä.

Tässä opinnäytetyössä on tarkoitus tutkia tarkemmin, kuinka laitteen yhtä eniten käytettyä sensoria - kameraa - hallitaan selaimessa saatavilla olevilla rajapinnoilla ja luodaan sovellusprototyyppi Tertin kartanon tarvetta varten. Kiinnostuin tästä aiheesta ja yleisesti interaktiivisesta selainohjelmoinnista Verkkomultimedia-kurssilla, jossa käytiin ohjelmointi- ja teoriatöillä läpi RIA (Rich Internet Application)-sovelluksia. Lisäksi HTML5-spesifikaatiohin ja rajapintoihin tutustuminen lisäsi innostusta ja kiinnostusta entisestään.

Työn toisessa luvussa käyn hieman läpi erilaisia sensoreita, joita nykyisistä mobiililaitteista löytyy ja sitä, kuinka niitä käytetään JavaScriptin avulla. En keskity luvussa kertomaan, kuinka nämä sensorit toimivat, vaan siihen, kuinka niiden tarjoama data saadaan käytettäväksi, mutta kerron kuitenkin lyhyesti mitä eri sensorit tekevät ja kuinka niitä on mahdollista käyttää hyödyksi.

Kolmannessa luvussa pohjustan työssä käytettäviä tekniikoita, ja kerron niihin liittyvää historiaa sekä eri ominaisuuksia ja toimintaperiaatteita. Näiden tekniikoiden tarkempi tarkastelu ja toiminnallisuuksien läpikäynti tapahtuu kuitenkin itse työtä käsittelevässä neljännessä luvussa. Nelosluvun ensimmäinen osa sisältää itse työn toiminnallisuuden ja sen sisältämät koodit kuvina sekä auki selitettyinä. Se sisältää tarkemmin selitettynä kolmosluvussa pohjustettujen tekniikoiden käytön sekä hieman niihin liittyvää lisätietoa sekä työn lopputuloksen. Neljännen luvun toinen osa sisältää ko-

keellisen ominaisuuden, johon sain idean Android-käyttöjärjestelmän efektistä, jossa kotinäytön kuva kääntyy hieman sisäänpäin, kun yritetään vierittää reunimmaisista ikkunoista ”ulos” laitteen reunoilta. Kokeilin, kuinka tällaista visuaalista efektiä voisi hyödyntää yhdessä laitteen kallistuksen kanssa kuvien tai videon esittämisessä.

2 SENSOREISTA YLEISESTI

Laitteiston hallinta ja niistä saatavan datan hyödyntäminen ovat suuressa roolissa mobiilisovelluksia kehitettäessä. Nykyisin lähes jokaisesta älypuhelimesta ja tablettietokoneesta löytyvät kamera, digitaalinen kompassi, kiihtyvyysanturi, etäisyysanturi, valontunnistin sekä gyroskooppi (Uj-Jaman 2011). Lisäksi useat kannettavat tietokoneet tarjoavat näitä samoja tietoja.

Tähän asti ohjelmoija, joka on halunnut hyödyntää sovelluksissaan mobiililaitteiden ja kannettavien tietokoneiden tarjoamaa sensoridataa, on joutunut kirjoittamaan jokaiselle alustalle oman sovelluksen käyttöjärjestelmäkohtaisella ohjelmakoodilla (käytetään myös termiä natiivikoodi tai natiivisovellus); Androidille Javalla, iOS:lle Objective-C:llä ja Windows Phonelle C#:lla. Tämä aiheuttaa sen, että kehittämiseen menee kolme kertaa kauemmin ja haluttaessa jakaa sovellusta, täytyy se jakaa Windows Phonen ja iOS:n tapauksissa niiden oman kauppapaikan kautta. Androidin tapauksessa sovelluksen voi jakaa Google Playn lisäksi missä vain pelkällä sovelluksen apk-tiedostolla. APK-tiedosto eli Android Application Package file on Android-käyttöjärjestelmän käyttämä tiedostomuoto, jota käytetään sovelluksien jakamiseen ja asentamiseen. Korvaamalla perinteinen natiivisovellus puhtaalla selaintoteutuksella, voidaan kiertää nämä hidasteet ja kehitys, päivitys ja jakaminen suoraviivaistuvat ja helpottuvat, kun käyttäjän tarvitsee vain näppäillä tiensä verkkosivulle.

Kovaa vauhtia yleistyvän HTML5-standardin mukana on tullut, ja on tulossa, JavaScript-ohjelmointirajapintoja, joiden kautta näitä samoja antureita voidaan lukea puhtaasti suoraan selaimen kautta. Tämän työn kirjoitushetkellä, syksyllä 2013, iso osa rajapinnoista on vielä kehitysvaiheessa, eivätkä kaikki selaimet tue niitä. Lisäksi useat näitä rajapintoja tukevat selaimet vaativat selainvalmistajakohtaisen etuliitteen (englanniksi vendor prefix) rajapintojen toimimiseen, koska yhteistä standardia ei vielä ole (Teeple 2012). Monet näistä rajapinnoista tarjoavat niin sanottua korkean tason dataa

(englanniksi high level data), joka tarkoittaa että data ei ole täysin raakaa dataa suoraan laitteistosta, eivätkä ne ota kantaa siihen, mikä kyseistä dataa tarjoaa.

2.1 Gyroskooppi ja paikkatieto

Gyroskooppeja löytyy tavallisesti kalliimmista kannettavista laitteista. Tavallisista mekaanisista gyroskoopeista poiketen kannettavien laitteiden gyroskooppi on normaalisti elektroninen, esimerkiksi iPhone 4:ssa on MEMS-pohjainen gyroskooppi (Micro-Electro-Mechanical System), jossa yhdistyvät elektroniset ja mekaaniset komponentit hyvin pienessä, mikrometri-luokkaa olevassa mittakaavassa (iPhone 4 gyroscope…2010). Gyroskooppi mittaa laitteen kallistusta. HTML5:n mukana on tullut tapa lukea laitteen tarjoamaa orientaatiodataa. Tämä on window-olion tapahtuma, jota kuuntelemalla saadaan haluttu data. Kallistumisen lisäksi tämä tapahtuma kertoo laitteen suunnan pohjoisen suhteen, eli se tarjoaa myös kompassidataa.

```
window.addEventListener('deviceorientation', function(event){
    var suunta      = event.alpha;
    var xKallistus  = event.beta;
    var yKallistus  = event.gamma;
}, false);
```

KUVA 1. Deviceorientation-tapahtuma

Kuvassa 1 on esitetty, kuinka laitteen kallistusdataa saadaan luettua deviceorientation-tapahtuman kautta. Laitteen kääntely suorittaa tämän tapahtuman. Tapahtumaan kiinnitetyn funktion argumentin ominaisuuksista löytyy alpha-, beta- ja gamma-ominaisuudet, jotka sisältävät eri akseleiden kallistusdatan. Alpha-ominaisuus sisältää laitteen suunnan maapallon pohjoisnapaan nähden. Kun laitteen kärki osoittaa kohti pohjoisnapaa, on alpha-ominaisuuden arvo 0. Mitä lähempänä nollaa tai 360:tä arvo on, sitä paremmin laitetta osoitetaan pohjoisnapaa päin.

Beta-ominaisuus kertoo laitteen x-akselin suuntaisen kallistuksen. Se kertoo, kuinka paljon laitteen kärkeä tai alaosaa on nostettu tai käännetty alaspäin. Kun laite makaa tasaisella pinnalla, tämän arvo on 0 astetta. Kun laitteen kärkeä käännetään alaspäin, arvo muuttuu kohti 180:tä, ja ylöspäin kallistettaessa kohti -180:tä.

Gamma-ominaisuus pitää sisällään y-akselin kallistuksen määrän. Kallistus on 0 astetta, kun laitteen molemmat sivut ovat yhtä kaukana maan pinnasta. Kulma kasvaa kohti 90:tä kun laitteen oikeaa kylkeä käännetään alaspäin, ja vastaavasti -90:tä kohti kun vasenta kylkeä käännetään alaspäin (Orientation and motion data explained 2013). Kaikki selaimet eivät kuitenkaan käsittele tätä tapahtumaa täysin samalla tavalla. Esimerkiksi Google Chromessa tämä tapahtuma ajetaan hitaammin kuin Mozilla Firefoxissa eli päivitysnopeus ei ole yhtä suuri molemmissa. Kuvassa 2 on havainnollistettu eri akselit ja mikä arvo muuttuu mihinkin suuntaan käännettäessä.

KUVA 2. Laitteen eri akselit ja kallistukset havainnollistettuna (Orientation and motion… 2013)

Paikkatieto on yksi käytetyimmistä laitteen tarjoamista informaatioista mobiilisovelluskehityksessä ja sillä onkin hyvin monimuotoiset käyttökohteet. Paikkatiedon avulla voidaan rakentaa opassovelluksia, jotka esimerkiksi näyttävät kahvilat, jotka ovat tietyn välimatkan päässä käyttäjän laitteen sijainnista, tai vaikkapa hyvin suosittuja matkanseurantasovelluksia, joita käytetään hyvin paljon esimerkiksi urheilussa. Laitteen paikallistamista varten HTML5:n mukana on tullutkin Geolocation-niminen JavaScript-rajapinta. Tämä rajapinta mahdollistaa käyttäjän laitteen sijaintitietojen saamisen leveys- ja pituuspiirien asteina. Geolocation on kehitysasteeltaan W3C:n suositus (englanniksi recommendation), eli selainvalmistajien olisi hyvä lisätä sille tuki (Geolocation API…2013). Yli 80 % selaimista tukeekin sitä, eli lähes jokainen versio jokaisesta selaimesta. Ainoa selain, joka ei tue sitä ollenkaan, on Opera Mini (Can I use Geolocation 2013). Kuten aiemmin esitelty deviceorientation, tämäkään rajapinta ei

ota kantaa siihen, millä tavalla tai mikä laitteen sijaintitiedon tarjoaa. Laitteen sijaintitiedot voi saada jopa neljällä eri tavalla:

- laitteen IP-osoitteen avulla
- langattoman verkon avulla
- laskemalla lähellä olevien matkapuhelintukiasemien avulla
- laitteen GPS-sirun avulla

(You Are Here… 2013)

Kuten monet muutkin käyttäjän tietoja käsittelevät rajapinnat ja sovellukset, paikannusmahdollisuus voi herättää monissa pelkoa yksityisyyden suojan heikentymisestä. W3C kuitenkin määrittelee hyvin tarkasti näihin liittyvien rajapintojen ja muiden tekniikoiden turvallisuusmääritykset. Yleinen määräys selainvalmistajille on, etteivät näiden kaltaiset järjestelmät saa käyttää kyseessä olevaa dataa ilman käyttäjän erikseen antamaa lupaa. Geolocation-rajapinnan tapauksessa selain esittää käyttäjälle vahvistusikkunan ruudun yläreunassa, jonka kautta käyttäjä voi halutessaan antaa sivustolle luvan käyttää omaa sijaintiaan tai estää sen käytön. Tämän lisäksi vahvistusikkuna on

- ei-modaalinen, eli se ei estä käyttäjää vaihtamasta selaimen ikkunaa tai välilehteä, tai käyttämästä muita sivuston toimintoja
- välilehtikohtainen, eli se näkyy vain siinä välilehdessä tai ikkunassa, jossa se aktivoitiin
- estävä, eli sivusto ei saa sijaintitietoja ennen käyttäjän hyväksyntää
- ehdoton, eli sitä ei voi ohittaa ohjelmakoodilla, jotta haittasivustot eivät voisi käyttää sijaintitietoja käyttäjän tietämättä

(Geolocation API Specification 2013; You Are Here… 2013).

```
// Päivittyy vain sivunlatauksessa
navigator.geolocation.getCurrentPosition(naytaSijainti);
// Päivittyy jatkuvasti
navigator.geolocation.watchPosition(naytaSijainti);

function naytaSijainti(position) {
    var latitude    = position.coords.latitude;
    var longitude   = position.coords.longitude;
    console.log("Lat: "+ latitude + " Long: " + longitude);
}
```

KUVA 3. Geolocation-rajapinnan käyttö

Geolocation-olio löytyy myöhemmin esiteltävän getUserMedia-metodin tavoin navigator-olion alta. Geolocation-oliolla on kaksi mahdollista metodia, joilla sijaintitiedot voidaan saada, nämä ovat esitetty kuvassa 3. GetCurrentPosition-metodi kertoo sijainnin vain kerran per sivunlataus. Tässä esimerkissä metodille on annettu argumentiksi vain sijainnin kertova funktio, mutta sille on myös mahdollista antaa virheidenkäsittelijäfunktio sekä olio, joka voi sisältää erilaisia asetuksia, kuten määrityksen käyttää parasta mahdollista tarkkuutta.

GetCurrentPosition-metodilla voidaan kuitenkin saada laitteen edellinen sijainti automaattisesti lisäämällä asetusolioon ”maximumAge”-määrityksen. Tämän avulla selain voi hakea laitteen edellisen sijaintitiedon välimuistista annetun ajan välein. Tämä on käytännöllinen silloin, kun riittää, että tiedetään missä laite oli edellisen päivityksen aikana. Mikäli laitteen sen hetkinen sijainti täytyy olla reaaliajassa käytettävissä, on parempi käyttää watchPosition-metodia.

WatchPosition-metodi ottaa argumenteiksi samat kuin getCurrentPosition. Tätä metodia kutsutaan aina, kun laite liikkuu tai tarkempi sijainti on saatavilla. Sijainnin kertovan funktion parametrina on Position-olio, jonka alta löytyy coords.latitude- ja coords.longitude-ominaisuudet. Muita mahdollisia coords-olion alta löytyviä ominaisuuksia ovat mm. **accuracy**, joka kertoo sijainnin tarkkuuden metreinä, **heading**, joka kertoo liikkumissuunnan myötäpäivän suuntaan kasvavina asteina pohjoisen suhteen sekä **speed**, joka kertoo nopeuden m/s-nopeudella. On kuitenkin hyvä huomata, että näistä vain latitude, longitude ja accuracy ovat ainoat ominaisuudet, jotka antavat arvoja joka kerta, muiden toiminta riippuu käytettävän laitteen tuesta (You Are Here… 2013).

Geolocation-rajapinta palauttaa kuitenkin pelkkiä koordinaattitietoja asteina. Tämä toimii niissä tapauksissa, joissa pelkkä paikannus koordinaattipisteillä on riittävä, mutta mikäli on tarve saada esimerkiksi tarkempi osoite vaikkapa verkko-ostoksien lomakkeiden täyttämiseen, koordinaateista ei ole apua. Tätä varten on olemassa mm. erilaisia kolmansien osapuolien rajapintoja. Esimerkiksi Google Mapsin rajapintakokoelmasta löytyy Geocoder-niminen olio, jonka metodeilla koordinaattipisteet voidaan muuntaa selkokieliseksi osoitteeksi, tai toisin päin (Korpela 2011, 245 – 247).

Myös W3C kehittää tällaista järjestelmää, joka kuitenkin toimii hieman eri tavalla. W3C:n ratkaisu on nimeltään Geolocation API v2, joka lisää jo olemassa olevaan Geolocation-rajapintaan mahdollisuuden palauttaa tarkempia osoitetietoja suoraan ilman konversioita. Siinä missä koordinaattitiedot saadaan aiemmin esitetyllä tavalla position-olion coords-ominaisuuden avulla, tarkemmat osoitetiedot saadaan position olion address-ominaisuuden kautta. Address-ominaisuus sisältää seuraavat ominaisuudet:

- country (ISO 3166-standardin mukainen kaksikirjaiminen tunnus)
- region (esim. osavaltion nimi Yhdysvalloissa)
- county (regionin sisällä oleva alue)
- city
- street
- streetNumber
- premises (voi sisältää mm. rakennuksen nimen)
- postalCode

On kuitenkin hyvä huomata, että tämä lisäys jo valmiiseen Geolocation-rajapintaan on vasta kehitysvaiheessa, eikä tarkan osoitteen saaminen ole välttämättä mahdollista tällä menetelmällä (Geolocation API Specification Level 2 2011).

2.2 Muut sensorit

Paikkatieto- ja kallistussensoreiden lisäksi kannettavat laitteet on varustettu kappaleen 2 alussa listatuilla sensoreilla. Näistä monelle on jo olemassa JavaScript-rajapinnat tai kuunneltavat tapahtumat. Liiketunnistin mittaa laitteen liikettä ja siihen kohdistuvia kiihtyvyysvoimia, sekä sen orientaatiota. Liiketunnistimen data saadaan kuuntelemalla window-olion tasolla devicemotion-tapahtumaa. Tapahtumankäsittelijään liitetyn funktion argumenttina on eventData-tyyppiä oleva olio, jonka ominaisuuksista löytyvät kiihtyvyys ja kiihtyvyys painovoiman kanssa eri akseleiden suhteen, sekä orientaatiodata, joka käsiteltiin edellisessä kappaleessa. Mikäli laite ei kykene tarjoamaan kiihtyvyyden arvoa ilman painovoiman aiheuttamaa lisäystä (esimerkiksi laitteessa ei ole gyroskooppia), voidaan acceleration-ominaisuuden rinnalla käyttää accelerationIncludingGravity-ominaisuutta. Tällöin acceleration-ominaisuuden tarjoama data on ”null”, tyhjä. Tämäkin rajapinta on vielä kehitysvaiheessa (W3C:n kehitysasteena working draft), joten kaikki selaimet eivät välttämättä tue tätä (DeviceOrientation Event… 2011; LePage 2011).

```
window.addEventListener('devicemotion', function(eventData){
    // Arvot ovat m/s^2
    var xKiiht = eventData.acceleration.x;
    var yKiiht = eventData.acceleration.y;
    var zKiiht = eventData.acceleration.z;
    var paivitysTaajuus = eventData.interval;
}, false);
```

KUVA 4. Devicemotion-tapahtuma ja sen sisältämä data

Etäisyysanturin (englanniksi proximity sensor) dataa voidaan lukea kahdella eri tavalla, jotka molemmat antavat hieman erilaista tietoa. Liike- ja orientaatiodatan tavoin etäisyysanturin tiedot saadaan kuuntelemalla tiettyä tapahtumaa, joita tässä tapauksessa on kaksi: deviceproximity ja userproximity. Deviceproximity tarjoaa tietoa siitä, kuinka kaukana käytettävä laite on jostain lähellä olevasta esteestä, etäisyys ilmoitetaan senttimetreinä. Tällä saadaan myös selville maksimi- ja minimietäisyydet, jotka sensori voi havaita, mikäli laite kykenee ilmoittamaan ne, muuten arvot ovat nolla.

Userproximity kertoo, onko laite lähellä jotain esinettä, kuten käyttäjän korvaa puhuttaessa. Sen antama ”near”-ominaisuus on totuusarvo-tyyppinen, jolloin arvolla ”true” laite on lähellä esinettä, ja arvolla false taas ei. Nämä tapahtumat kulkevat W3C:ssä nimellä Proximity Events, jonka kehitysaste on suositusehdokas (englanniksi Candidate Recommendation) ja sen viimeisin päivitys on julkistettu 1.10.2013. Kirjoitushetkellä vain Mozillan Firefox-selain tukee niitä (Proximity 2013; Proximity Events 2013).

```
window.addEventListener('userproximity', function(event) {
  if (event.near) {
    console.log("Laite on lähellä käyttäjää!");
  }
  else {
    console.log("Laite ei ole lähellä käyttäjää!");
  }
});

window.addEventListener("deviceproximity", function(event){
    console.log("Kohde on " + event.value + " cm päässä.");
});
```

KUVA 5. Userproximity- ja deviceproximity-tapahtumat

Valontunnistin mittaa laitetta ympäröivän valon määrää, jonka avulla tehdään mm. mobiililaitteiden automaattinen näytön kirkkauden säätö. Valontunnistin mittaa valonmäärää lukseina. Niin etäisyys- kuin kiihtyvyysantureiden tapaan, valontunnistimen arvoja luetaan erilaisten tapahtumien kautta. Etäisyysanturin tavoin käytössä on kaksi hieman erilaista tapahtumaa: devicelight ja lightlevel, jotka molemmat tarjoavat samaa dataa hieman eri muodossa.

Devicelight tarjoaa valonmäärän laitteen valontunnistimen antamana luksimääränä. On hyvä huomata, että tämäkin anturin toiminta eroaa hieman eri laitteiden välillä, eivätkä kaikki välttämättä anna täysin samoja arvoja samoissa tilanteissa. Lightlevel tarjoaa valonmäärän yksinkertaisena kolmiportaisena asteikkona: **dim** (himmeä), **normal** (normaali) ja **bright** (kirkas). Vaikka eri laitteiden sensorit voivatkin antaa erilaisia arvoja samoissa tilanteissa, W3C suosittelee seuraavia vaihteluvälejä edellä mainitulle asteikolle:

- dim = alle 50 luksia
- normal = 50 - 10000 luksia
- bright = yli 10000 luksia

(Ambient Light Events 2013).

```
window.addEventListener("devicelight", function(event){
    console.log("Valonmäärä on " + event.value + " luksia");
});

window.addEventListener("lightlevel", function(event){
    console.log("Kirkkauden taso: " + event.value);
});
```

KUVA 6. Valontunnistimen tarjoaman datan lukeminen

Näiden lisäksi kehittäjillä on käytössä jo iso liuta erilaisia laiteläheisiä, JavaScript-rajapintoja, kuten esimerkiksi laitteen akun varausta mittaava Battery Status API, joka myös osaa kertoa kuinka kauan menee, ennen kuin laitteen akku on ladattu täyteen tai kuinka kauan menee että akku tyhjenee. Contacts Manager API tarjoaa rajapinnan laitteen yhteystietojen tutkimiseen ja sekä niiden muokkaamiseen. Calendar API mahdollistaa laitteen kalenterisovelluksen tietojen käytön ja Messaging API mm. tekstiviestien ja sähköpostin käsittelyn. Monet näistä kuitenkin ovat vielä kehitysvaiheessa, eikä ole varmuutta tulevatko ne edes koskaan standardeiksi, mutta on kuitenkin hyvä

tietää, että tällaisia natiivisovelluksien käyttämiä rajapintoja on saatavilla myös selaimessa (All Standards and Drafts 2013).

3 KÄYTETTÄVÄT TEKNIIKAT

Tässä luvussa käydään läpi työssä käytettävät tekniikat sekä hieman niiden taustoja ja historiaa. Työn runkona toimii HTML-merkkauskielen seuraava standardiversio, 5, ts. HTML5. Se tarjoaa uusia ohjelmointirajapintoja, joita tähän asti on ollut saatavilla vain natiivisovelluksissa. Lisäksi työn toiminnallisuus toteutetaan JavaScript-ohjelmointikielellä, jonka kautta HTML5:n tarjoamia rajapintoja voidaan käyttää.

3.1 HTML5

Tässä luvussa käydään läpi HTML5:n liittyvien eri osien taustaa sekä sen tarjoamia uusia elementtejä, joita tässä työssä käytetään. Vaikka HTML5 on seuraava askel sen edeltäjästä HTML4:stä, on se kuitenkin paljon muutakin kuin pelkkä verkkosivujen merkkauskieli. Se on mm. kehitysalusta, jonka päälle voidaan rakentaa monialustaisia (englanniksi multi-platform) sovelluksia. Kehitysalusta, joka koostuu HTML-merkkauskielen lisäksi verkkosivujen tyylityksistä (CSS, Cascade Style Sheets) ja JavaScript-ohjelmointikielestä (Marshall 2011). W3C:llä on tarkoitus saada HTML5-standardi valmiiksi vuoden 2014 loppuun mennessä (Plan 2014 2012).

W3C eli World Wide Web Consortium on vuonna 1997 perustettu kansainvälinen yhteisö, joka kehittää internetin standardeja ja jonka tavoitteena on varmistaa internetin pitkäkestoinen kasvu. Yksi W3C:n tavoitteista on mahdollistaa internetin käyttö mahdollisimman usealla alustalla ja laitteella, kuten kännyköillä, älypuhelimilla ja televisioilla. W3C:n perustaja ja toinen sen johtajista on WWW:n luoja Tim Berners-Lee (Facts About W3C 2013; W3C Mission 2013).

3.1.1 Historia

HTML:n historia kantaa aina 1990-luvun alkuun, jolloin verkkosivujen merkkaukselle ei vielä ollut virallista eikä epävirallista standardia. HTML-standardin määrittely ja vakiinnuttaminen tehtiin vuosien 1995–1997 aikana (Korpela 2011, 24). Vuonna 1995

julkaistiin HTML 2.0, kaksi vuotta tästä, vuoden 1997 tammikuussa HTML 3.2 (Raggett 1997), ja saman vuoden joulukuussa HTML 4.0 (HTML 4.0 Specification 1997).

Tästä eteenpäin vuosia 1998–2003 voisi kutsua ”pysähtyneisyyden ajaksi”, jolloin HTML sai hyvin vähän päivityksiä, viimeisimmän päivitysversion ollessa 4.01. Samoina vuosina W3C kehitti XHTML-standardia, joka on XML-pohjaiseksi sovitettu HTML 4.01. XHTML:n ensimmäinen versio oli 1.0 ja myöhemmin muutettuna XHTML 1.1. W3C kehitti myös XHTML 2.0-versiota, joka kuitenkin päätyi umpikujaan (Korpela 2011, 24). XHTML:n kehityksen he lopettivat lopulta vuonna 2006 ja alkoivat yhdessä WHATWG:n kanssa työstää nykyistä HTML5-standardia (Silverman 2012).

WHATWG (Web Hypertext Application Technology Working Group) on Applen, Mozillan ja Operan jäsenistä koostuva yhteisö. Se perustettiin vuonna 2004 sen jälkeen, kun W3C:n järjestämässä kokouksessa ei päästy yhteisymmärrykseen HTML:n tulevaisuudesta. Enemmistö halusi, että HTML4:n seuraaja rakennettaisiin uudelle pohjalle, kun taas myöhemmin WHATWG:n perustaneet henkilöt jäivät vähemmistöön ehdotuksellaan, että seuraaja pitäisi rakentaa jo olemassa olevan perustan päälle (Korpela 2011, 58).

WHATWG:n tarkoituksena on kehittää HTML5-standardia. WHATWG kuitenkin käyttää ns. elävää standardia (englanniksi living standard) HTML:n kehityksessä. Tämä tarkoittaa sitä, että he eivät periaatteessa käytä versionumeroita, kuten HTML5:ttä, vaan puhuvat pelkästä HTML:stä. Elävä standardi myös tarkoittaa, että standardi elää koko ajan ja sitä päivitetään jatkuvasti ja siihen lisätään uusia ominaisuuksia palautteiden, mahdollisuuksien ja tarpeellisuuden mukaan. Usein puhuttaessa HTML5:stä tarkoitetaankin HTML:n viimeisimpiä muutoksia (FAQ 2013).

Vuonna 2008 julkaistiin Ian Hicksonin kirjoittama ensimmäinen luonnos HTML5:stä. Vuoden 2011 marraskuussa Adobe ilmoitti lopettavansa Flashin kehittämisen mobiililaitteille ja siirtyvänsä kehittämään HTML5-alustaa. Tämä ja Applen Steve Jobsin ilmoitus hylätä Flash ja alkaa tukea HTML5-teknologiaa Applen mobiililaitteissa antoi lisäpotkua HTML5:n leviämiselle. Asiantuntijoiden mukaan HTML5 on jatkuvasti kehittyvä ja joka ei koskaan tule olemaan täysin valmis (Silverman 2012). Korpela

(2011) kirjoittaakin: ”HTML5 rakentuu vähittäisen kehityksen (evoluution) eikä kumouksen (revoluution) ajatukselle.”.

3.1.2 Canvas-elementti

Canvas-elementti on ehkäpä yksi kiinnostavimmista HTML5:n mukana tulleista uusista HTML-elementeistä. Canvas-elementtiä voisi nimensä mukaisesti kutsua virtuaaliseksi piirustusalustaksi, koska sille voidaan piirtää grafiikkaa käyttämällä JavaScriptia. Sitä hyödyntäen voidaan piirtää mm. graafeja, luoda animaatioita ja manipuloida kuvia ja videota reaaliajassa (Canvas 2013).

Canvas-elementti on alun perin vuonna 2004 Applen kehittämä tekniikka heidän Mac OS X:n dashboard-sovelluksiaan sekä Safari-selaintaan varten, jonka muut selainvalmistajat vaiheittain Mozilla etunenässä ottivat mukaan omiin selaimiinsa (Heyes 2008).

TAULUKKO 1. Työpöytäselainten tuki canvas-elementille (Can I use the HTML5 canvas element? 2013)

Selain	Selainversio (julkaisuvuosi)
Internet Explorer	**9.0** (2011)
Mozilla Firefox	**2.0** (2006)
Google Chrome	**4.0** (2010)
Apple Safari	**3.1** (2008)
Opera	**9.0** (2006)

Taulukosta 1 voidaan nähdä tämän hetken viiden suosituimman työpöytäselaimen tuen canvas-elementille (2D-konteksti). Taulukkoon on listattu selainten ensimmäiset versiot, jotka tukivat canvas-elementtiä. Vaikka canvas-elementti onkin alun perin Applen kehittämä, Safari ei ollut ensimmäinen selain, joka tuki sitä. Internet Explorer-riin canvas-tuki saapui kovin myöhään, vaikka selainperhe on vanha. Internet Explorerin 7- ja 8-versiot tukevat canvas-elementtiä kolmannen osapuolen explorercanvas-kirjastolla (Explorercanvas 2009).

TAULUKKO 2. Mobiiliselainten tuki canvas-elementille (Can I use the HTML5 canvas element? 2013)

Selain	Selainversio (julkaisuvuosi)
Chrome for Android	**29.0** (2013)
Firefox for Android	**24.0** (2013)
Blackberry selain	**10.0** (2013)
Android selain	**4.2** (2012)
Opera Mobile	**14.0** (2013)
Opera Mini	**5.0–7.0** (2010–2012) *
iOS Safari	**7.0** (2013)
IE Mobile	**10.0** (2012)

Taulukkoon 2 on listattu suurimpien mobiiliselainten tuki canvas-elementille (2D-konteksti). Toisin kuin taulukkoon 1, tähän taulukkoon on listattu selainten uusimpia versioita, koska päivitystietoja canvas-elementistä on kovin niukasti. Opera Mini poikkeaa muista siinä, että vaikkakin se tukee canvas-elementtiä, ei se kuitenkaan pysty pyörittämään animaatioita tai muita monimutkaisempia sovelluksia (Can I use the HTML5 canvas element? 2013).

Canvas-elementti on tämän työn kannalta hyvin tärkeä ja oleellinen osa. Sitä tarvitaan, kun halutaan manipuloida kamerasta tulevaa videokuvaa tai tavallista videota reaaliajassa. Tässä työssä sitä käytetään mm. ns. chroma keyingin käyttämiseen. Chroma keyingin tarkoittaa jonkun tietyn värin poistamista videokuvasta, esimerkiksi näyttelijän taustalla oleva vihreä kangas. Taustavärin poistamisen jälkeen jäljelle jäävän kuvan taustalle voidaan liittää esimerkiksi eri taustakuva, kuten kuvasta 7 voidaan nähdä. Periaatteessa mikä tahansa tasainen väri sopisi taustakankaaksi, mutta kirkas sininen ja vihreä ovat vakiintuneet, koska ne ovat värejä, jotka ovat mahdollisimman kaukana ihmisihon värisävyistä (Malone 2010).

KUVA 7. Chroma keyn käyttö (Malone 2010)

Työssä canvas-elementtiä käytetään myös QR-koodin lukemiseen kameran videokuvasta. Yksinkertaisesti esitettynä canvas-elementin sisältö lähetetään base64-koodattuna QR-koodin lukijalle, joka palauttaa koodin sisällön. Näistä tarkemmin työn myöhemmissä vaiheissa.

3.1.3 Video-elementti

Video-elementti on audio-elementin ja edellä mainitun canvas-elementin kanssa yksi keskeisimmistä uusista HTML-elementeistä. Video-elementti mahdollistaa videon ”upotuksen” (englanniksi embed) suoraan verkkosivulle ilman mitään selainlisukkeita (englanniksi plugin). Käytetyin selainlisuke videon toistoon on Adoben Flash player (Web Browser Plugin Market Share 2012). Flash on ongelmallinen erityisesti mobiilialustoilla, joilla akun kesto on isossa roolissa. Se on usein hyvin akkusyöppö sekä vie melko paljon tehoja, varsinkin tehottomammilla laitteilla, joka johtaa laitteen yleiseen hidasteluun ja Flashilla tuotetun sisällön pätkimiseen. Nämä ovat asioita, joita HTML5 video pyrkii muuttamaan (Adobe Flash: Why does Flash drain… 2013).

Lähestulkoon kaikki videoiden suoratoistopalvelut, kuten YouTube, Vimeo ja Metaface käyttävät Flash playeria videosoittimissaan (Video on the web 2013). HTML5:n yleistyessä nopeasti monet tällaiset palvelut päivittävät soittimiaan toimimaan ilman Flash playeria, puhtaasti selaimen omilla tekniikoilla. YouTube ja Vimeo julkaisivat ensimmäiset HTML5-versiot omista soittimistaan vuonna 2010 (Dybwad 2010).

Video-elementin käyttö ei kuitenkaan ole täysin mutkatonta. Vaikkakin työn kirjoitushetkellä syksyllä 2013 kaikki suurimmat työpöytä- ja mobiiliselaimet (pois lukien Opera Mini) tukevat video-elementtiä (Can I use the HTML5 video element? 2013),

kaikki niistä eivät kuitenkaan tue samoja videoformaatteja. Tämä johtaa siihen, että sisällöntuottajan on luotava videonsa monessa eri formaatissa tukeakseen kaikkia selaimia. Onneksi kuitenkin video-elementille voi kuvassa 8 esitetyllä tavalla antaa useampia lähdetiedostoja, jolloin selain valitsee niistä ensimmäisen, jonka se voi toistaa.

```
<video width="320" height="280" controls>
    <source src="video.mp4" type="video/mp4"/>
    <source src="video.ogg" type="video/ogg"/>
    <source src="video.webm" type="video/webm"/>
</video>
```

KUVA 8. Kuinka video-elementille annetaan useampi lähdetiedosto

Lähdetiedostojen järjestykselläkin on kuitenkin väliä, sillä iOS 3.2:ssa oli ohjelmointivirhe, jonka takia käyttöjärjestelmä otti huomioon vain ensimmäisen lähdetiedoston. Mikäli ensimmäiseksi oli asetettu tiedosto, jota iOS ei osannut toistaa, ei se normaalin käytännön mukaan hypännyt sen yli ja valinnut seuraavaa toimivaa, vaan lopetti videon toiston. Tämä virhe korjattiin iOS:n 4.0-versiossa, mutta mikäli haluaa olla varma ja ottaa huomioon käyttäjät, jotka eivät ole päivittäneet laitteitaan, kannattaa ensimmäiseksi asettaa MP4-tiedosto, koska se on ainoa formaatti, jota iOS:n Safari-selain tukee, kuten taulukosta 4 voidaan havaita (Video on the web 2013).

TAULUKKO 3. Työpöytäselainten tuki eri videoformaateille (Media formats supported … 2013)

Selain / Formaatti	MP4	WebM	Ogg
Internet Explorer (10.0)	Kyllä	Ei *	Ei
Mozilla Firefox (24.0)	Kyllä *	Kyllä	Kyllä
Google Chrome (29.0)	Kyllä	Kyllä	Kyllä
Apple Safari (6.0)	Kyllä	Ei *	Ei *
Opera (16.0)	Ei	Kyllä	Kyllä

Taulukossa 3 on listattu suosituimpien työpöytäselainten tämän hetkisten versioiden tuki HTML5 video-elementin eri videoformaateille. Mozillan Firefox-selain ei sisällä sisäänrakennettua tukea MP4-tiedostoille patenttiongelmien välttämiseksi, vaan luottaa toistamisessa laite- tai käyttöjärjestelmätukeen:

- Windows 7 & 8:lla Firefox 21 eteenpäin

- Windows Vistalla Firefox 22 eteenpäin
- Androidilla Firefox 20 eteenpäin, tällä alustalla ei laitekiihdytystä
- Firefox OS:lla Firefox 15 eteenpäin
- Windows XP, Linux ja Mac OS X-tuki on työn alla (Wijering 2013).

Internet Explorer ja Applen Safari eivät toista WebM- tai Ogg-tiedostoja vakiona. Internet Explorer tarvitsee WebM-tiedoston toistoa varten erikseen asennetun koodekin ja Applen Safari tarvitsee WebM- ja Ogg-tiedostojen toistoa varten erikseen asennetut selainlisukkeet (Media formats supported by the HTML audio and video elements 2013).

TAULUKKO 4. Mobiiliselainten tuki eri videoformaateille (Can I use the HTML5 video element? 2013)

Selain / formaatti	MP4	WebM	Ogg
Chrome (Android) (29.0)	Kyllä	Kyllä	Ei
Firefox (Android) (24.0)	Kyllä *	Kyllä	Kyllä
Blackberry selain (10.0)	Kyllä	Ei	Ei
Android selain (4.2)	Kyllä	Kyllä	Ei
Opera Mobile (14.0)	Kyllä	Kyllä	Ei
iOS Safari (7.0)	Kyllä	Ei	Ei
IE Mobile (10.0)	Kyllä	Ei	Ei

Taulukoon 4 on listattu suosituimpien mobiiliselainten tämän hetkiset versiot. Firefoxin mobiiliversio on selaimista ainoa, joka toistaa kaikkia formaatteja ja samalla ainoa, joka toistaa Ogg-tiedostoja. Koska Firefox luottaa käyttöjärjestelmä- ja laitetukeen MP4-tiedostojen toistossa, saattaa sen kanssa esiintyä ongelmia mobiilialustoilla, kuten värien vääristymistä (H.264 video in Firefox for Android 2012).

Taulukoissa 3 ja 4 esitetyt videoformaatit ovat seuraavat:

- MP4 = MPEG 4-tiedosto H.264 videokoodekilla ja AAC tai MP3 äänikoodekilla
    - Patentoitu, sisältää lisenssimaksun
- WebM = WebM-tiedosto VP8 videokoodekilla ja Vorbis äänikoodekilla
    - Lisenssivapaa

- Ogg = Ogg-tiedosto Theora videokoodekilla ja Vorbis äänikoodekilla
    - Lisenssivapaa

(HTML5 Video 2013; Video on the web 2013)

Näiden edellä mainittujen asioiden lisäksi video-elementtiä käytettäessä on myös useimmissa tapauksissa otettava huomioon vanhemmat (useimmiten Internet Explorer 8 ja vanhemmat) selaimet, jotka eivät tue HTML5 videota. Tämä voidaan hoitaa monellakin tapaa, mutta useimmiten käytetään kuvassa 9 olevaa tapaa, jossa annetaan latauslinkki videoon ja/tai käytetään ns. ”fallback”-keinona Flash-videosoitinta.

```
<video controls poster="video.jpg" width="854" height="480">
  <source src="video.mp4" type="video/mp4">
  <source src="video.webm" type="video/webm">
  <object type="application/x-shockwave-flash" data="player.swf"
width="854" height="504">
    <param name="allowfullscreen" value="true">
    <param name="allowscriptaccess" value="always">
    <param name="flashvars" value="file=video.mp4">
    <!--[if IE]><param name="movie" value="player.swf"><![endif]-->
    <img src="video.jpg" width="854" height="480" alt="Video">
    <p>Your browser can’t play HTML5 video. <a href="video.webm">
Download it</a> instead.</p>
  </object>
</video>
```

KUVA 9. Video-elementin sisällä oleva Flash-soitin varmistuskeinona (Simple HTML5 video player… 2011)

Sen lisäksi, että video-elementillä voidaan upottaa videoita verkkosivuille ilman selainlisukkeita, yksi sen hienoimmista ominaisuuksista on se, että sitä voidaan ohjata JavaScriptilla. Soittimessa on kyllä omat ohjainpainikkeet, mutta halutessaan kehittäjä voi luoda oman käyttöliittymän videosoittimelle käyttämällä HTML-merkkausta, CSS-tyylejä ja JavaScriptia (Using JavaScript to control the HTML5 video player 2013).

Yksinkertaisen Play/Pause-ohjauksen lisäksi JavaScriptilla on mahdollista vaikuttaa lähes kaikkiin videon ominaisuuksiin, kuten esimerkiksi volyymiin, lähdetiedostoon ja toistonopeuteen (HTML Audio/Video DOM Reference 2013). Lisäksi yhdistämällä canvas-elementin video-elementin kanssa, voidaan videota manipuloida reaaliajassa. Miltä kuulostaa esimerkiksi videon muokkaus negatiiviksi samalla kun se pyörii?

Tässä työssä video-elementtiä käytetään näyttämään laitteen kamerasta tuleva videosyöte. Syöte asetetaan soittimen lähdetiedostoksi, jonka jälkeen se näkyy soittimessa, ja jonka kautta se voidaan ottaa esimerkiksi canvas-elementille muokattavaksi. Video- ja canvas-elementit toimivatkin erinomaisesti yhdessä.

3.2 JavaScript

JavaScript on dynaamisesti tyypitetty, olio-pohjainen ja yksi maailman suosituimmista ohjelmointikielistä, jonka suorittamisesta vastaa selain. Yleisin käyttötarkoitus JavaScriptille on selainohjelmoinnissa (ns. front-end/client-side scripting/coding), mutta sitä voidaan soveltaa myös web-kehityksen ulkopuolella. Esimerkiksi hyvin suosittu Unity-pelinkehitysympäristö tukee JavaScriptia yhtenä skriptauskielenään C#:n ja Boo Script:n lisäksi (Unity scripting 2013).

JavaScriptin kehitti alun perin Netscapen insinööri Brendan Eich vuonna 1995 ja se julkaistiin ensimmäisen kerran Netscape 2:n kanssa vuoden 1996 alussa. Nimestään huolimatta JavaScript ei ole sukua Sun Microsystemin Java-ohjelmointikielelle, mutta JavaScript kylläkin lainaa syntaksiaan Javasta ja C-kielestä (A re-introduction to JavaScript 2013). JavaScript perustuu ECMAScriptiin, jonka uusin versio on 5.1 (JavaScript Language Resource 2013). Tämä tarkoittaa, että JavaScriptin ydinominaisuudet- ja toiminnot periytyvät ECMAScriptistä, joiden lisäksi kieleen on lisätty JavaScript-kohtaisia ominaisuuksia.

ECMAScript on skriptauskielien standardi, johon JavaScriptin lisäksi perustuvat mm. Adoben ActionScript ja Microsoftin JScript. Näitä kieliä voisikin ajatella ECMAScriptin ”murteina”. ECMA tulee sanoista European Computer Manufacturer’s Association (What is ECMAScript? 2012).

Vaikkakin JavaScript on olio-pohjainen ohjelmointikieli, siinä ei ole luokkia samaan tapaan kuin esimerkiksi Javassa. Luokka-toiminallisuus pystytään kuitenkin luomaan käyttämällä olioprototyyppejä (Object.prototype 2013). JavaScript eroaa muista olio-pohjaisista ohjelmointikielistä myös siinä, että se on datatyypitön/dynaamisesti tyypitetty ohjelmointikieli, toisin sanoen ohjelmoijan ei tarvitse erikseen määritellä muuttujan tyyppiä (A re-introduction to JavaScript 2013). Kuvassa 10 on esitetty tyypitetyn ja tyypittömän/dynaamisesti tyypitetyn muuttujan eroa. Kuvan tapauksessa JavaSc-

ript-tulkki (selain) tulkitsee muuttujan merkkijonoksi, koska sen arvo on lainausmerkkien sisällä. JavaScriptin ”var” tarkoittaa muuttujaa (englanniksi variable).

```
// Esimerkki tyypitetystä merkkijono-tyyppisestä muuttujasta Java-kielessä
String nimi = "Teemu";
// Esimerkki tyypittömästä muuttujasta JavaScript-kielessä
var nimi = "Teemu";
```

KUVA 10. Tyypitetty muuttuja vs. tyypitön muuttuja

Tällä on kuitenkin omat haittansa. Siinä missä esimerkiksi C++-koodi käännetään (englanniksi compile) alustakohtaiselle konekielelle, JavaScript ”tulkitaan” (englanniksi interpret) ja tämän tulkinnan hoitaa selain. Tämä johtaa siihen, ettei JavaScript-koodista saada niin suorituskykyistä, verrattuna käännettäviin ohjelmointikieliin (What’s the difference between… 2013). JavaScriptin tapauksessa selain hoitaa muistinhallinnan ja päättelee muuttujan arvosta mitä tehdä, joka vaativimmissa sovelluksissa voi ja näkyy hidasteluina; tyypitetyissä ohjelmointikielissä suorituskyky on paljon parempi (Vroom 2012).

JavaScript, kuten muutkin ohjelmointikielet, sisältää sisäänrakennettuja olioita sekä niiden ominaisuudet ja metodit. JavaScriptissa tällaisia ovat mm.:

- numero (number)
- totuusarvomuuttuja (boolean)
- merkkijono (string)
- päivämäärä (date)
- matriisi (array)
- matemaattinen (math)

sekä niiden ominaisuudet ja metodit.

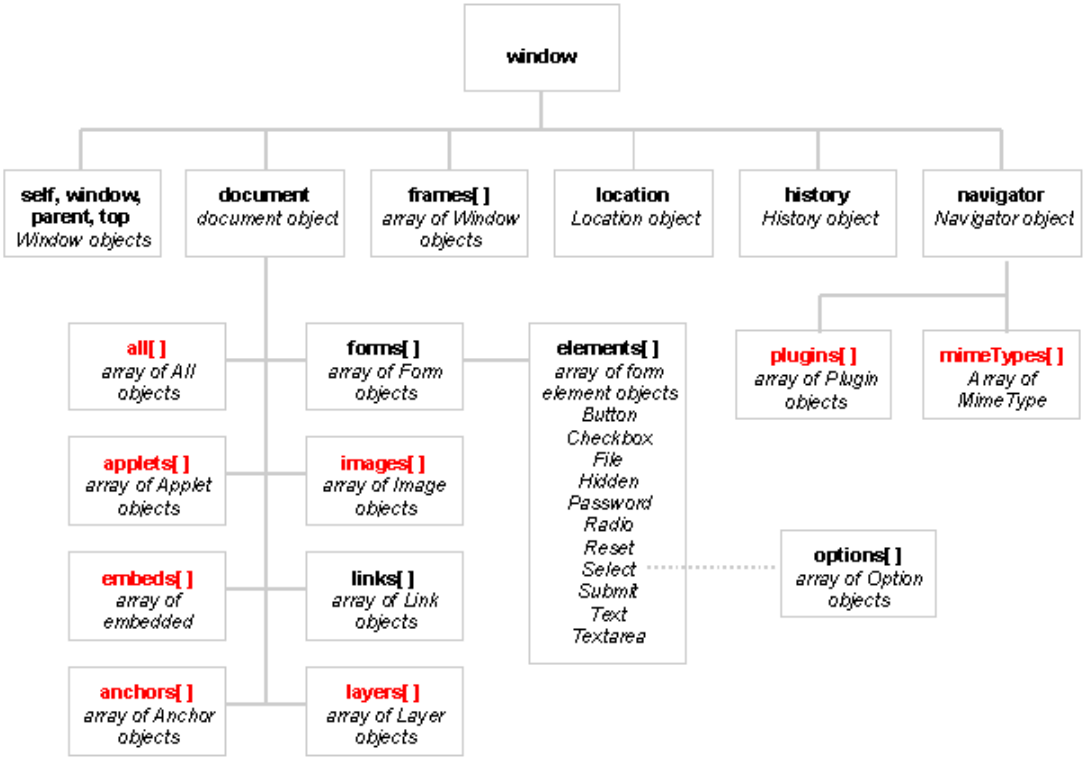
Edellä mainittujen olioiden lisäksi JavaScriptissa on sisäänrakennettuna olioita, jotka antavat tietoja käyttäjän laitteesta. Ylimmän tason olio on window-olio, joka sisältää tiedot sillä hetkellä avatusta ikkunasta. Se myös sisältää seuraavia aliobjekteja ja vastaavasti niiden ominaisuudet ja metodit:

- **window, self:** window-olio itse. Sisältää mm. metodit dialogi-ikkunoiden avaamiseen (alert(), confirm() ja prompt())
- **document:** sisältää tiedon kaikista sivulla olevista lomakkeista, kuvista, linkeistä, syöttökentistä jne

- **location:** sisältää tiedot sen hetkisen ikkunan tai välilehden verkko-osoitteesta
- **history:** sisältää tiedon käyttäjän selaushistorian pituudesta sen hetkisessä ikkunassa tai välilehdessä sekä metodit historialistassa liikkumiseen eteen- ja taaksepäin
- **screen:** sisältää tiedon käyttäjän laitteen näytöstä, sen resoluution, käytettävissä olevan leveyden ja korkeuden sekä näytön värisyvyydestä.
- **navigator:** sisältää tiedot käyttäjän selainohjelmasta, kuten minkä valmistajan selain, selaimen versionumero ja mm. mitä selainlisukkeita on asennettuna.

(JavaScript and HTML DOM Reference 2013)

Kaikkia näitä kutsutaan window-olion alta, esimerkiksi window.location.href (joka palauttaisi sen hetkisen verkko-osoitteen kokonaisuudessaan), mutta window-sanan voi jättää pois, koska kaikki alkavat window-sanalla, joten JavaScript sallii tämän (The hierarchy of JavaScript Objects 2006). Kuvassa 11 on edellinen hierarkia havainnollistettu kaaviona.



KUVA 11. JavaScriptin oliohierarkia (Javascript Lesson 9 2008)

Työn kannalta tärkein olio on navigator-olio, koska sen alta löytyy metodi getUserMedia, joka mahdollistaa käyttäjän kameran kuvan käytön. Tämä työ koostuu suurimmaksi osaksi JavaScriptistä, HTML-merkkauksen ollessa hyvin pienessä roolissa;

”pakollista” (ts. tyylittelyjen ulkopuolista) HTML-merkkausta tarvitaan vain 5-10 riviä.

3.3 Kuvan ja äänen kaappaus

Kuvan ja äänen kaappaus ja hallinta ovat pitkään olleet haluttu asia web-kehityksessä. Tähän asti kehittäjät ovat kuitenkin joutuneet käyttämään erillisiä selainlisukkeita, kuten Adobe Flash ja Microsoft Silverlight, tämän saavuttamiseksi, jota monet, kuten Bidelman (2012) ja Chinnathambi (2013) pitävät huonona asiana aikakaudella, jolloin selainlisukkeiden tarvetta yritetään vähentää. Ehkä käytetyin niistä on Adoben Flash, johon luultavasti suurin osa netin käyttäjistä on joskus törmännyt, mm. videosoittimissa.

HTML5:n mukana on kuitenkin tullut erilaisia rajapintoja käyttäjän laitteen kameran hallintaan. Nämä rajapinnat ovat kehittyneet muutaman viime vuoden ajan vilkkaasti. Monet tahot ovatkin luoneet omia määrityksiään koskien näitä ja muita laiterajapintoja. Tämä aiheutti sen, että erilaisia määrityksiä alkoi olla liikaa, jolloin W3C loi viralisen tahon, Device APIs Policy Working Groupin (DAPWG), jonka tarkoituksena on yhdistää ja standardisoida eri laitedataa käsitteleviä rajapintoja koskevat määritykset yhdeksi kokonaisuudeksi (Bidelman 2012).

3.3.1 Input- ja device-elementit

DAPWG:n ensimmäinen askel standardisoida median kaappaus selaimessa oli HTML Media Capture. Tämä toimii lisäämällä tavalliseen HTML-tiedostonsyöttöelementtiin uuden accept-attribuutin. Kuvan 12 HTML-elementin accept-attribuutin arvo antaa selaimelle vihjeen näyttää pelkästään kuvatiedostot tiedostonvalintaikkunassa.

```
<form action="server.php" method="post" enctype="multipart/form-data">
    <input type="file" name="kuvia" accept="image/*"/>
</form>
```

KUVA 12. Input-elementin accept-attribuutti

Mikäli käyttäjä haluaa nauhoittaa ääntä laitteensa mikrofonilla tai ottaa kuvan tai nauhoittaa videota laitteensa kameralla, lisätään edellisen esimerkin elementtiin capture-attribuutti kuvan 13 osoittamalla tavalla.

```
<form action="server.php" method="post" enctype="multipart/form-data">
    <!-- Huom. oikeat arvot accept-attribuutille -->
    <input type="file" name="audio" accept="audio/*" capture> <!-- Ääni -->
    <input type="file" name="video" accept="video/*" capture> <!-- Video -->
    <input type="file" name="kuva" accept="image/*" capture>  <!-- Kuva -->

    <!-- W3C:n vanhemman määrityksen mukainen merkintätapa -->
    <input type="file" name="audio" accept="audio/*" capture="microphone">
    <input type="file" name="video" accept="video/*" capture="camcorder">
    <input type="file" name="kuva" accept="image/*" capture="camera">
</form>
```

KUVA 13. Input-elementin capture-attribuutti

Tämä määrää input-elementtiä klikattaessa selaimen avaamaan laitekohtaisen dialogi-ikkunan, josta käyttäjä voi valita haluamansa lähteen kuvalle tai äänelle, esimerkiksi joko laitteen kiintolevy tai kamera. W3C:n turvallisuusmäärityksissä on myös määrätty, ettei selain saa aktivoida esimerkiksi kameraa, ennen kuin käyttäjä on antanut siihen luvan (HTML Media Capture 2013).

Ensimmäinen alusta, joka tuki tätä tapaa oli Android 3.0:n vakio, Webkit-moottorin päällä pyörivä Android Browser. Nykyään sitä tukevat seuraavat selaimet:

- Safari ja Chrome Mobile iOS 6:ssa ja uudemmissa
- Chrome Mobile Android 3.0:ssa ja uudemmissa
- Firefox Mobile Android 3.0:ssa ja uudemmissa
- Opera 16 Android 3.0:ssa ja uudemmissa

Kuitenkaan näistä kaikki eivät tue määrittelyä täysin samalla tavalla. Osa toteutti määrittelyn osittain ja jotkut toteuttivat sen vanhan määrittelyn pohjalta, jonka kaltainen merkintätapa on nähtävissä kuvasta 13 (Bidelman 2012).

HTML Media Capture oli kuitenkin hyvin rajoittava tekniikka, koska se ei esimerkiksi tukenut reaaliaikaista videokuvan suoratoistoa. Tämän johdosta kehitettiin standardia uutta elementtiä varten, joka oli device-elementti. Tämä elementti antoi sivulle oikeuden käyttää laitetta, esimerkiksi päätelaitteen kameraa. Mahdollisia datalähteitä, joita elementti pystyi käyttämään, olivat mm.

- **fs - File system**, laitteen kiintolevy

- **media**, laitteen kamera tai mikrofoni tai muu vastaava laite
- **rs232,** tietokoneen tietoliikenneportti
- **usb,** jokin USB-väylään kytketty laite

(Video Conferencing with… 2010)

```
<device type=media onchange="update(this.data)"></device>
<video autoplay></video>
<script>
function update(stream) {
    document.getElementsByTagName('video')[0].src = stream.url;
}
</script>
```

KUVA 14. Device-elementin toiminta

Kuvassa 14 on esitetty, kuinka device-elementin lukema data, tässä tapauksessa video, asetetaan video-elementin lähteeksi, jota kautta kuva näkyy käyttäjälle. Data siirretään update-nimisen funktion parametrina, joka asetetaan JavaScriptilla video-elementin lähdedataksi käyttäen videoelementi ”src”-ominaisuutta (src tulee englannin kielen sanasta source, lähde). Update-funktio ajetaan ”onchange”-tapahtumassa, joka tapahtuu aina, kun elementin sisältö muuttuu. Lisäksi video-elementti sisältää ”autoplay”-attribuutin, joka käynnistää videon heti kun se on mahdollista.

3.3.2 getUserMedia()

Yksikään selainvalmistaja ei kuitenkaan koskaan lisännyt device-elementille tukea selaintensa julkaisuversioihin ja WHATWG päätyi hylkäämään sen vuonna 2011 ja siirtyi elementtipohjaisesta ratkaisusta JavaScript-rajapintatoteutukseen navigator.getUserMedia-rajapinnalla. Opera oli yksi selainvalmistaja, joka kehitti selaimestaan prototyypin, joka tuki device-elementtiä (Bidelman 2012; HTML Standard Tracker 2011; Native webcam support… 2011). GetUserMedia-rajapinta liittyy myös vahvasti toiseen hyvin mielenkiintoiseen rajapintakokonaisuuteen, WebRTC:hen, joka tulee sanoista Web Real-Time Communication - Reaaliaikaista kommunikaatiota selaimen välityksellä, jota tässä työssä ei kuitenkaan käsitellä.

GetUserMedia on nykyään käytössä ja sitä kehitetään jatkuvasti. Myös selainvalmistajat lisäävät selaimiansa tukemaan sitä. Työn kirjoitushetkellä vain Mozilla Firefoxin työpöytäselain, Googlen Chromen työpöytä- ja mobiiliversiot sekä Blackberryn selain

tukevat tätä tekniikkaa. Opera tuki getUserMedia-rajapintaa sen 12.0 ja 12.1 versioissaan, joiden jälkeen selain siirtyi käyttämään Opera Presto-moottorin sijaan Chromium/Blink:ia, joihin Opera ei ole sitä vielä toteuttanut, mutta tulee toteuttamaan tulevaisuudessa (Can I use getUserMedia/Stream API? 2013; A first peek at Opera 15… 2013; Opera version history 2013).

GetUserMedia-rajapinnan toimintaperiaate on lähes identtinen device-elementin vastaavaan, mutta siinä missä device-elementille oli mahdollista antaa monia erilaisia laitteita datalähteeksi, getUserMedia tukee vain kameraa ja mikrofonia. Device-elementin kanssa laitetta käytettiin erillisen HTML-elementin kautta, mutta getUserMedia on puhdas JavaScript-toteutus. Kuvassa 15 on esitetty, kuinka laitteen kameran kuva esitetään käyttäjälle käyttäen tätä rajapintaa ja video-elementtiä.

```
<script>
var video = document.getElementById("videoplayer");
navigator.getUserMedia({video: true, audio: false},
    //Success-funktio
    function(mediastream) {
        video.src = window.URL.createObjectURL(mediastream);
    },

    //Error-funktio
    function(error) {
        alert("Virhe: "+error);
    }
);
</script>
<video id="videoplayer" autoplay></video>
```

KUVA 15. getUserMedia-metodin käyttö

On kuitenkin hyvä huomioida, ettei kuvan 15 esimerkissä oteta huomioon selainten eroavaisuuksia. Kun tämäkin rajapinta on vielä kehittyvässä vaiheessa, on hyvä käyttää selainvalmistajien etuliitteitä metodien ja rajapintojen kanssa. Esimerkiksi sekä getUserMedia että window.URL merkitään eri selaimilla hieman eri tavalla. Tämän työn pääpaino on getUserMedia-metodin ympärillä, koska juuri se mahdollistaa kameran käytön.

4 PUUTARHAKIERROSSOVELLUKSEN PROTOTYYPPI

Tässä luvussa rakennetaan virtuaalisen puutarhakierrossovelluksen prototyyppi Tertin kartanolle käyttäen aikaisemmin esiteltyjä tekniikoita. Tertin kartano on mikkeliläinen matkailualan yritys. Sovellus mahdollistaa eräänlaisen lisätyn todellisuuden käytön esimerkiksi erilaisilla esittelykierroksilla. Tertin kartanon tapauksessa tätä sovellusta hyödynnetään käyttämällä QR-koodeja, joiden kautta käyttäjälle esitetään erilaisia videoita alueesta ja sen historiasta. Työ on jatkoa minun heille tekemästäni älypuhelimella tai tablet-tietokoneella käytettävälle puutarhakierrossovellukselle, jossa alueen ilmakuvakarttaan oli aseteltu eri kohteisiin painonappeja, joita painamalla aukesi video, joka kertoi käyttäjälle tietoa kyseisestä paikasta. Tätäkin sovellusta on tarkoitus käyttää mobiililaitteilla.

Työ koostuu yhdestä HTML-tiedostosta sekä yhdestä JavaScript-tiedostosta, johon tulee itse sovelluksen logiikka ja toiminnallisuus. Olisi myös mahdollista kirjoittaa JavaScript-koodi suoraan HTML-tiedostoon, mutta koska koodia on suhteellisen runsaasti, ajattelin että on luettavuuden ja siisteyden kannalta järkevintä kirjoittaa se omaan tiedostoonsa. Käytän työssäni puhdasta JavaScriptia, mutta olisi myös mahdollista käyttää erilaisia JavaScript-sovelluskehyksiä, kuten jQuerya, esimerkiksi helpottamaan elementtien valitsemista ja erilaisten tapahtumakäsittelijöiden käyttöä.

Itse työn tekemiseen käytän kevyttä Notepad++-sovellusta sen syntaksikorostuksen ja kätevien pikanäppäinten takia. Olisi mahdollista käyttää myös jotain raskaampaa sovelluskehitysympäristöä, esimerkiksi Eclipseä, joka ilmoittaisi syntaksivirheistä, mutta näin ettei sille ole tarvetta. Vaikkakaan JavaScriptin suoritukseen ei tarvitse palvelinohjelmistoa, tarvitaan sellainen kuitenkin, kun halutaan ladata sisältöä JavaScriptin avulla. Selaimet estävät ulkoisen sisällön lataamisen ilman palvelinohjelmistoa, esimerkiksi jos haluttaisiin ajaa koodia suoraan vaikkapa muistitikulta. Työn tapauksessa ulkoinen sisältö tarkoittaa laitteen kamerasta tulevaa dataa. Palvelinohjelmistona käytän ilmaista Wampserver-ohjelmistoa ja sovelluksen testaamiseen pääasiassa Mozillan Firefox-työpöytäselaimen versiota 24, joka on uusin julkaisuversio työn kirjoitushetkellä.

```
<!DOCTYPE html>
<html>
<head>
<!--
    Head-tagien sisältö sisältää dokumenttikohtaiset määritykset.
    Esim. JavaScript-tiedostot ja dokumentin otsikko.
-->
<meta charset="UTF-8">
<title>getUserMedia()-työ</title>
<link rel="stylesheet" type="text/css" href="ont-tyyli.css"/>
<!--
    Käytettävien kirjastojen esittely
    qr-min.js on qr-koodien lukemiseen käytettävä kirjasto,
    ont-script.js on työn logiikan ja toiminnallisuuden sisältävä tiedosto.
-->
<script src="js/qr-min.js"></script>
<script src="js/ont-script.js"></script>
</head>
<!-- Body-tagien sisälle tulevat käyttäjälle näkyvät asiat -->
<body>
<label id="tulostus"></label><br/>
<button id="kaynnistaVideo">Käynnistä video</button>
<div id="cont">
    <div id="readarea"></div>
    <video id="camstream"></video>
    <canvas id="videocanvas"></canvas>
</div>

<canvas id="canvas"></canvas>
<video id="video" controls></video>

</body>
</html>
```

KUVA 16. Sovelluksen HTML5-standardin mukainen HTML-pohja

Kuvassa 16 on esitetty HTML5-standardin mukainen HTML-dokumentin pohja. <!DOCTYPE html>-määritys antaa selaimelle ohjeen piirtää dokumentti oikein HTML5-standardin mukaisesti. Dokumenttiin lisätty qr-min.js-tiedosto on QR-koodien lukemiseen tarkoitettu JavaScript-kirjasto. Se on avoimen lähdekoodin kirjasto, lisensoitu Apache License 2.0:lla. Ensimmäinen video-elementti on tarkoitettu näyttämään kamerasta tuleva videokuva. Canvas-elementti, jolla on id-attribuutti ”canvas”, prosessoi kamerasta tulevaa kuvaa ja lukee mahdolliset QR-koodit. Toiset video- ja canvas-elementit on tarkoitettu QR-koodien kautta aukeavien videoiden toistamiseen ja prosessointiin.

4.1 Sovelluksen logiikka ja toiminnallisuus

Sovelluksen toiminnallisuus on kasattu yhteen JavaScript-tiedostoon. Tiedosto koostuu yhdestä olio-tyyppisestä muuttujasta, jonka sisälle kaikki muuttujat ja funktiot tulevat. Hain tämän tyyppisellä ratkaisulla siisteyttä ja olio-pohjaista tuntumaa.

```
var ont = {
    // Yleiset muuttujat, joita sovelluksessa käytetään
    camstream: "",  // Videosoitin, johon kameran videokuva tulee
    video: "",      // Videosoitin, jonka kautta soitetaan videot
    canvas: "",     // Canvas-elementti, joka prosessoi kameran kuvaa
    ctx: "",
    videoCanvas: "",
    videoCtx: "",
    tempCanvas: "",
    tempCtx: "",
    tulostus: "",
    koodinsisalto: false,
    toisto: "", // setTimeout
    vToisto: "",
    rafl: "",   // requestAnimationFrame loop
    vRafl: "",
    tiedostoArr: "",
    tiedosto: "",
    lueKoodi: "",
    kaynnistaVideo: "",
```

KUVA 17. Sovelluksen JavaScript-pohjan muuttujat

Aluksi koko sovellus kääritään ont-nimisen olion sisään. Sen sisälle tulevat muuttujat ja funktiot sen ominaisuuksiksi ja metodeiksi, jotka ovat ns. avain-arvo-pareja (key-value pair), jossa kaksoispisteen vasemmalla puolella on avain ja oikealla puolella arvo. Tällaiseen viitataan olio.ominaisuus; esimerkiksi työn tapauksessa ont.video. Kuvassa 17 on alustettu sovelluksessa käytettävät muuttujat. Kaikki ctx-sanan sisältävät muuttujat tarkoittavat canvas-elementin kontekstia, joille myöhemmissä vaiheissa tullaan piirtämään. Muuttujiin on alustettu valmiiksi käytettävät HTML-elementit ja toistotapahtumat sekä pari muuta käytettävää muuttujaa. Olen erotellut kameran ja videon käsittelyyn tarkoitetut toistotapahtumamuuttujat toisistaan v-etuliitteellä, joka ilmaisee, että kyseessä on videon käsittelyyn tarkoitettu muuttuja.

```
    initialize: function () {
        // Metodi, joka käynnistetään sovelluksen käynnistyksessä
    }, // initialize

    initializeCanvas: function(){
        // Metodi, jonka sisällä alustetaan eri canvas-elementit,
        // sekä niiden ominaisuudet
    }, // initializeCanvas

    loop: function(){
        // Metodi, jonka sisällä kaikki prosessointia vaavivat tehtävät
        // suoritetaan
        // Ns. toistofunktio.
    }, // loop

    streamCameraToCanvas: function(){
        // Piirtää kameran videokuvan canvas-elementille
    }, // streamCameraToCanvas

    setCanvasDimensions: function(){
        // Täällä asetetaan videon esitykseen käytettävien
        // canvas-elementtien koot
    }, // setCanvasDimensions

    convertVideo: function(){
        // Täällä videolle tehdään ns. chroma keying,
        // poistetaan tausta
    } // convertVideo
}; // ont

window.addEventListener('load',ont.initialize,true);
```

KUVA 18. Sovelluksen JavaScript-pohjan metodit

Kuvassa 18 on listattu sovelluksessa käytettävät metodit. Olen pyrkinyt järjestelemään ne siten, että ne olisivat jotakuinkin siinä järjestyksessä missä niitä käytetään. Metodien ja muuttujien nimet voisivat olla mitkä vain, mutta valitsin metodien nimiksi englanninkieliset, koska se tuntuu itsestäni luontevammalta. Sovellus käynnistyy kutsumalla initialize-metodia, jota kutsutaan window-olioon liitetyn tapahtumakäsittelijän kautta. Tapahtumakäsittelijälle on annettu kuunneltavaksi ”load”-tapahtuma, joka tapahtuu, aina kun sivu on ladattu kokonaan.

```
initialize: function () {
    ont.camstream = document.getElementById("camstream");
    ont.tulostus = document.getElementById("tulostus");
    ont.kaynnistaVideo = document.getElementById("kaynnistaVideo");

    window.requestAnimationFrame = window.requestAnimationFrame ||
                                   window.mozRequestAnimationFrame ||
                                   window.webkitRequestAnimationFrame ||
                                   window.msRequestAnimationFrame;

    window.cancelAnimationFrame = window.cancelAnimationFrame ||
                                  window.mozCancelAnimationFrame ||
                                  window.webkitCancelAnimationFrame ||
                                  window.msCancelAnimationFrame;

    navigator.getUserMedia = navigator.getUserMedia ||
                             navigator.mozGetUserMedia ||
                             navigator.webkitGetUserMedia ||
                             navigator.msGetUserMedia;
```

KUVA 19. Initialize-metodin alkuosa

Initialize-metodin alussa sijoitetaan yleisesti käytettyihin muuttujiin viittaukset eri HTML-elementteihin. Tämän jälkeen eri metodeille sijoitetaan sopivimmat arvot. Käyttämällä ||-operaattoria, selain valitsee arvoista sille sopivimman, esimerkiksi jos kyseessä on Mozillan Firefox-selain, joka ei tue requestAnimationFramea ilman etuliitettä, valitsee se moz-etuliitteen omaavan vaihtoehdon. Tämä on hyvin tärkeää vielä tässä vaiheessa, kun tekniikka on kehittyvää eivätkä kaikki selaimet tai selainversiot tue esimerkiksi requestAnimationFramea ilman selainkohtaista etuliitettä. Myös navigator.getUserMedia toimii tällä tavoin.

```
    if(navigator.getUserMedia) // Jos selain tukee getUserMediaa
    {
        navigator.getUserMedia({video: true, audio: false},
            // Onnistui, stream on käytettävissä
            function (mediastream) {
                // Asetetaan kamerasta tuleva kuva videosoittimen lähteeksi
                if(navigator.mozGetUserMedia){
                    // Firefoxia varten
                    ont.camstream.mozSrcObject = mediastream;
                }
                else {
                    // Muut selaimet
                    var url = window.URL || window.webkitURL;
                    ont.camstream.src = url.createObjectURL(mediastream);
                }

                ont.camstream.addEventListener('loadeddata', function(){
                    ont.camstream.play();
                    ont.initializeCanvas(); // Alustetaan canvas-elementit
                });
            },
            // Virhe, stream ei ole käytettävissä, kameraa ei löydy tms
            function (error) {
                ont.tulostus.innerHTML = "Virhe: " + error;
            }
        );
    }
    else // Jos ei tue
    {
        ont.tulostus.innerHTML = "Selaimesi ei tue getUserMediaa.";
        return;
    }
    // Palauttaa qr-koodissa olleen datan
    qrcode.callback = function(a) { ont.koodinsisalto=a; }
}, // initialize
```

KUVA 20. Initialize-metodin loppuosa

Kuvassa 20 on initialize-metodin loppuosa. Alussa testataan tukeeko käyttäjän selain getUserMedia-metodia. Mikäli getUserMedia ei ole tuettu etuliitteen kanssa tai ilman, ohjautuu sovellus ehtolauseen else-lohkoon, jossa käyttäjälle tulostetaan virheilmoitus sekä sovelluksen suoritus lopetetaan. Aivan viimeisenä on QR-koodin lukijan funktio, joka palauttaa koodissa olleen datan ja asettaa sen koodinsisalto-muuttujan arvoksi, mutta tähän palataan myöhemmin tarkemmin.

Mikäli selain tukee getUserMedia-metodia, siirtyy sovellus if-lauseen sisälle. GetUserMedia-metodin syntaksi on seuraavanlainen:

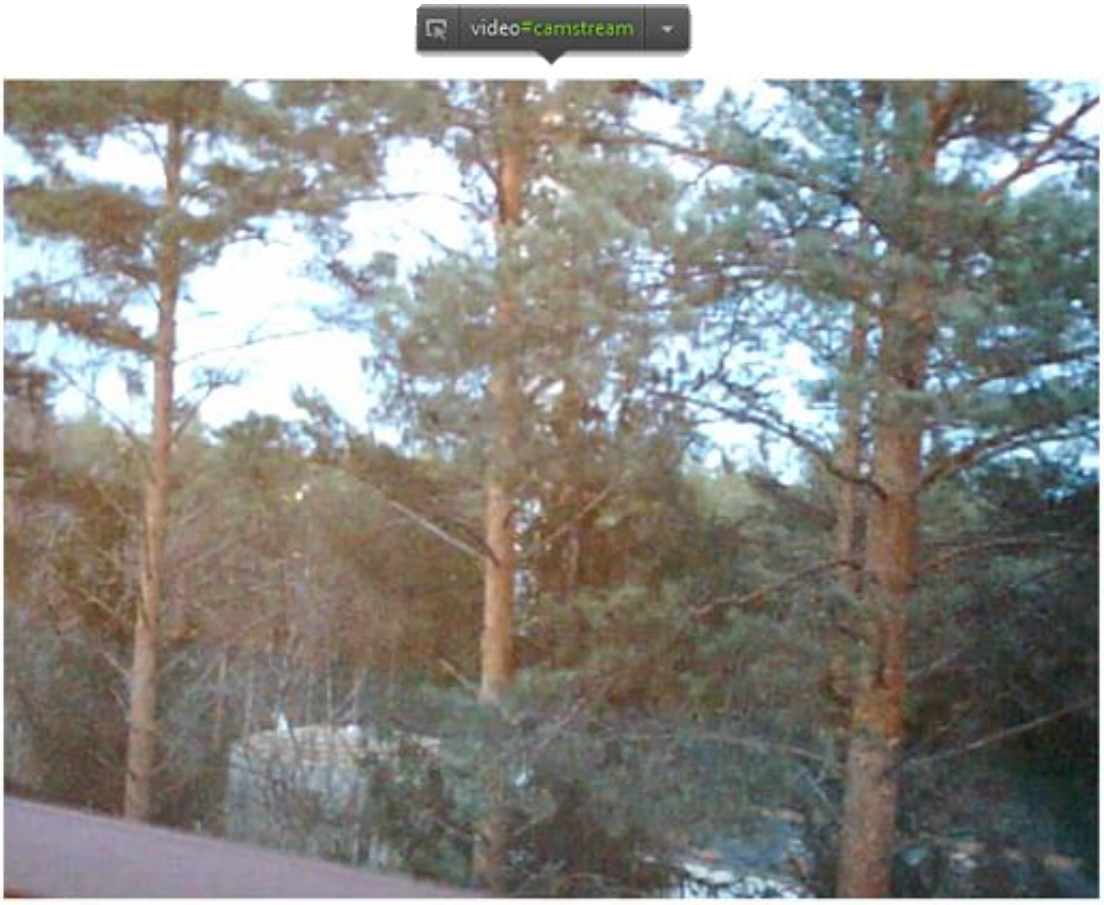
- navigator.getUserMedia (constraints, successCallback, errorCallback);

jossa argumentit ovat seuraavat:

- **constraints** on kahdesta totuusarvotyyppisestä avain-arvo -parista koostuva olio, jolla määrätään käytetäänkö laitteen kameraa vai mikrofonia vai molempia, esimerkiksi {video: true, audio: true} Näistä toisen tai molempien täytyy olla annettu, jotta syntaksi on validi. Mikäli avainta tai sen arvoa ei ole annettu, se on automaattisesti epätosi, false (Navigator.getUserMedia 2013).
- **successCallback** on funktio, joka ajetaan, jos laitteen video tai ääni on käytettävissä, toisin sanoen kun käyttäjä on antanut sovellukselle luvan käyttää laitettaan, ja jonka parametrina on localMediaStream-olio, joka sisältää laitteesta tulevan datan. Tämä voidaan asettaa esimerkiksi video-elementin lähdetiedostoksi. Tämä on pakollinen parametri.
- **errorCallback** on funktio, joka ajetaan virheen sattuessa tai kun käyttäjä ei anna sovellukselle lupaa käyttää laitettaan. Tämä on pakollinen argumentti Firefox-selaimessa. Sen parametrina on virhekohtainen virhekoodi. Virhekoodit ovat:
    - **PERMISSION_DENIED.** Käyttäjä ei antanut lupaa käyttää laitteensa kameraa tai mikrofonia.
    - **NOT_SUPPORTED_ERROR.** Selain ei tue constraints-oliossa määriteltyä rajoitetta.
    - **MANDATORY_UNSATISFIED_ERROR.** Constraints-oliossa määritellyn tyyppistä mediaraitaa ei löydy (Navigator.getUserMedia 2013).

Tässä työssä yllä mainitut funktiot on kirjoitettu suoraan getUserMedia-metodin sisään. Työn successCallback-funktion alussa tarkistetaan, onko selain Mozilla Firefox. Mikäli näin on, siirrytään ehtolohkon sisälle, jossa kamerasta tuleva kuva asetetaan video-elementin lähteeksi käyttämällä mozSrcObject-ominaisuutta. Tällä on mahdollista asettaa localMediaStream-olio suoraan video-elementin lähteeksi ilman että siitä tarvitsee luoda erikseen merkkijono createObjectURL-metodilla.

Else-lohko hoitaa localMediaStreamin käsittelyn muille selaimille. Ensin luodaan muuttuja ”url”, jonka arvoksi asetetaan joko window.URL tai window.webkitURL mikäli kyseessä on webkit-pohjainen selain kuten Google Chrome tai Apple Safari. Video-elementin src-ominaisuudelle ei suoraan pysty antamaan localMediaStream-oliota lähteeksi, koska se on blob-tyyppiä, vaan se täytyy ensin muuttaa merkkijonoksi käyttämällä url.createObjectURL-metodia. Nämä tulevat olemaan standardisoituja tulevaisuudessa (Taking webcam photos 2013).



KUVA 21. Web-kameran kuva selaimessa

Näiden ehtolohkojen jälkeen ohjelma siirtyy seuraavaan vaiheeseen, jossa kameran videokuvan sisältävään video-elementtiin liitetään tapahtumakäsittelijä, joka kuuntelee ”loadeddata”-tapahtumaa. Tämä tapahtuma tapahtuu, kun selain on ladannut video-elementtiin liitetyn videon datan. Tämä on kuunneltavista tapahtumista luotettavin, koska esimerkiksi Chrome ei tee mitään loadedmetadata-tapahtumassa, eikä Firefox sisällä siinä kameran kuvan kokoa (Cross-browser camera capture… 2013).

Tapahtumakäsittelijän funktiossa käynnistetään ohjelmallisesti video, jotta käyttäjä näkee varsinaisen kuvan. Tämän voi hoitaa myös asettamalla video-elementtiin attribuutin autoplay, tai XHTML-syntaksin mukaisesti autoplay=”autoplay”. Tämä voidaan tehdä myös JavaScriptissä asettamalla video-elementin autoplay-ominaisuus todeksi; ont.camstream.autoplay = true;. Tämän jälkeen selaimen ikkunassa näkyy laitteen kameran, tässä tapauksessa web-kameran, kuva.

```
initializeCanvas: function(){
    // Kameran videokuvan prosessointia varten
    ont.canvas = document.getElementById("canvas");
    ont.canvas.width = 200;
    ont.canvas.height = 200;
    ont.ctx = ont.canvas.getContext('2d');

    // Käynnistettävän videon prosessointia varten
    ont.tempCanvas = document.createElement("canvas");
    ont.tempCtx = ont.tempCanvas.getContext("2d");

    ont.videoCanvas = document.getElementById("videocanvas");
    ont.videoCtx = ont.videoCanvas.getContext('2d');

    ont.loop(); // Käynnistetään looppi
}, // initializeCanvas
```

KUVA 22. InitializeCanvas-metodi

Kameran kuvan sisältävän videosoittimen käynnistämisen jälkeen sovellus käynnistää initializeCanvas-metodin. Tämä metodi hoitaa eri canvas-elementtien luonnin ja määrittämisen. Ensimmäinen canvas-elementti, ont.canvas, on canvas-elementti, jota käytetään kameran videokuvan prosessointiin. Sovellus valitsee oikean elementin dokumentista id-attribuutin avulla käyttämällä document.getElementById-metodia. Tämän jälkeen canvas-elementille annetaan leveys ja korkeus. Tämän canvas-elementin ei tarvitse olla iso, joten annan arvoiksi 200 pikseliä. Lopuksi canvas-elementin 2d-konteksti, jolle itse piirto tapahtuu, asetetaan ont.ctx-muuttujaan.

Seuraavaksi metodi luo lennossa uuden canvas-elementin, jonka 2d-konteksti asetetaan niin ikään tempCtx-muuttujaan. VideoCanvas-muuttujalle annetaan arvoksi canvas-elementti, jolle chroma key-käsittelyn jälkeinen video tullaan asettamaan. TempCanvaksen tai videoCanvaksen kokoja ei määritellä vielä tässä vaiheessa, koska ne riippuvat toistettavan videon koosta, jota sovellus ei tässä vaiheessa vielä tiedä. Metodin lopuksi käynnistetään loop-niminen metodi.

```
loop: function(){
    if(!ont.koodinsisalto)
    {
        ont.streamCameraToCanvas();
        // Selaimet, jotka tukevat requestAnimationFramea
        if(window.requestAnimationFrame) {
            // Asetetaan toistotapahtuma muuttujaan,
            // jotta sen voi tarvittaessa sammuttaa
            ont.rafl = window.requestAnimationFrame(ont.loop);
        }
        else { // requestAnimationFramea tukemattomia selaimia varten
            ont.toisto = setTimeout(function(){
                ont.loop();
            },1000/60); // 60 fps
        }
    }
```

KUVA 23. Loop-metodin alku

Loop-metodi vastaa kameran kuvan prosessoinnin pyörittämisestä. Metodin alussa testataan if-lauseen ehdolla, onko koodinsisalto-muuttujalla arvoa. Koska tämä muuttuja on tyhjä aluksi, ohjelma siirtyy alkuasetuksilla if-lohkon sisään toistorakenteisiin. Lohkon sisällä ensimmäiseksi käynnistetään streamCameraToCanvas-metodi, joka nimensä mukaan piirtää kamerasta tulevaa kuvaa canvas-elementille. Tämän jälkeen testataan tukeeko selain requestAnimationFrame-metodia.

RequestAnimationFrame on uusi tapa luoda animaatioita JavaScriptilla. Tähän asti kehittäjät ovat käyttäneet erilaisia toistorakenteita, kuten setTimeout-metodia. Vaikka setTimeout toimiikin sinänsä hyvin, ei se ota huomioon muuta selainta. Mikäli setTimeout on määrätty toimimaan esimerkiksi 100 millisekunnin välein, se toistaa täsmälleen niin usein. Tämä aiheuttaa ongelmia silloin, kun käytettävä laite ei pysy toiston mukana. Mikäli laite ei kykene piirtämään sivua uudestaan yhtä nopeasti kuin toistorakenne vaatii, tapahtuu ns. frame skippingiä, osa piirrettävistä kuvista jää piirtämättä ja tämä näkyy käyttäjälle animaation nykimisenä.

RequestAnimationFrame eroaa siinä, että se antaa selaimelle vallan päättää kuinka usein uusi kuva piirretään, toisin sanoen seuraava kuva piirretään, kun selain kykenee siihen, ei jonkun tietyn ennalta määrätyn ajan välein. SetTimeout ei myöskään pysähdy, vaikka animoitava kohde olisi näytön ulkopuolella tai selaimen ei-aktiivisessa välilehdessä. Tämä lisää turhaan prosessorin ja/tai näytönohjaimen rasitusta, joka johtaa suurempaan virrankulutukseen, asia, joka on hyvin ikävä erityisesti mobiililaitteil-

la. RequestAnimationFrame toistaa animaatioita tyypillisesti 60 fps:n (frames per second) nopeudella, mutta tämä riippuu täysin käytettävästä laitteesta (requestAnimationFrame 2013; Window.requestAnimationFrame() 2013).

Mikäli selain tukee requestAnimationFrame-metodia, sovellus käyttää sitä toistamaan streamCameraToCanvas-metodia. RequestAnimationFrame-metodia tukemattomia selaimia varten olen laittanut setTimeout-metodin. Sen nopeudeksi olen karkeasti laittanut 60 fps, mutta sen lisäksi voisi tehdä loputtomia optimointeja, kuten laskea kuinka kauan selaimelta menee seuraavan kuvan piirtämiseen ja sen kautta määritellä toistonopeus. Ilman toistorakennetta streamCameraToCanvas-metodia kutsuttaisiin vain kerran, jolloin ainoastaan kameran videokuvan ensimmäinen kuva (englanniksi frame) piirtyisi canvas-elementille.

```
streamCameraToCanvas: function(){
    try {
        ont.ctx.drawImage(ont.camstream, 0, 0, 200, 200, 0, 0, 200, 200);
        if(ont.lueKoodi == true){
            // Lähetetään canvas-elementin sisältö decoderille
            qrcode.decode(ont.canvas.toDataURL());
        }
    } catch(exception) {
        console.log(exception);
    }
}, // streamCameraToCanvas
```

KUVA 24. StreamCameraToCanvas-metodi

StreamCameraToCanvas-metodin toiminnallisuus kiedotaan try-catch-rakenteen sisään, jolloin poikkeustilanteessa poikkeus kirjataan selaimen konsoliin. Kameran kuvan piirto canvas-elementille tapahtuu käyttämällä canvas-elementin piirtokontekstin drawImage-metodia. Yksinkertaisimmillaan tämä metodi tarvitsee argumenteiksi vain piirrettävän kuvan sekä piirron aloituspisteen canvas-elementin x- ja y-koordinaatteina. Työn tapauksessa annetaan kuitenkin vielä lisäargumentteina piirrettävän kuvan leveys ja korkeus kuvapisteinä sekä leikattavan kuvan aloituspisteet ja koko.

Käytin aluksi ont.camstream-muuttujan clientWidth- ja clientHeight-ominaisuuksia, mutta eri selaimet antavat niillä eri arvoja. Esimerkiksi Firefox antaa 300x150, mutta Chrome taas videokuvan oikean koon, tässä tapauksessa 640x480. Kovakoodatuilla

arvoilla saan kaikilla selaimilla samanlaisen tuloksen. Tämä riittää hyvin QR-koodien lukemiseen.

Mikäli käyttäjä käskee sovellusta lukemaan QR-koodin klikkaamalla selaimen ikkunaa, jolloin lueKoodi-muuttuja asetetaan todeksi, lähettää sovellus sen hetkisen canvas-elementin sisällön QR-koodin lukijalle käyttämällä toDataURL-metodia, joka muuntaa kuvan base64-koodatuksi merkkijonoksi. Tämä merkkijono on muotoa ”….”. Vakioasetuksilla kuvasta tulee PNG-muotoinen, mutta tyypin voi halutessaan muuttaa. (HTMLCanvasElement 2013). Kun QR-koodin lukija on lukenut koodin, se palauttaa koodin sisällön initialize-metodin lopussa olevalla qrcode.callback-funktiolla, jossa koodin sisältö asetetaan koodinsisalto-muuttujan arvoksi.

```
    else
    {
        var tukeeVarinaa = "vibrate" in navigator;
        if(tukeeVarinaa){navigator.vibrate(500);}

        ont.camstream.style.border = "5px solid green";
        setTimeout(function(){
            ont.camstream.style.border = "none";
        },2000);

        ont.kaynnistaVideo.style.display = "block";
        ont.video = document.getElementById("video");
        if(ont.video.src == null || ont.video.src == "") {
            ont.video.src = ont.koodinsisalto;
        }
        else {
            ont.tiedostoArr = ont.video.src.split("/");
            ont.tiedosto = ont.tiedostoArr[ont.tiedostoArr.length-1];
            if(ont.koodinsisalto != ont.tiedosto){
                ont.video.src = ont.koodinsisalto;
            }
        }
        ont.kaynnistaVideo.addEventListener("click", ont.videoHandler);
    }
}, // loop
```

KUVA 25. Loop-metodin loppuosa

Kuvassa 25 on esitetty loop-metodin loppuosan toiminnallisuus. Sovellus siirtyy tähän osaan, mikäli aikaisemmin tarkistetulla koodinsisalto-muuttujalla on jokin arvo. Mikäli sillä on arvo, tarkoittaa se, että QR-koodin lukija on löytänyt ja tulkinnut QR-koodin sisällön. Kun koodin sisältö on luettu, näyttää sovellus kameran videokuvan ympärillä kahden sekunnin ajan vihreän reunuksen merkkinä käyttäjälle. Lisäksi käytetään HTML5:n mukana tullutta laiterajapintaa, joka värisyttää laitetta 500 millisekunnin

ajan käyttämällä sen värinämoottoria. On kuitenkin hyvä huomata, että selain voi tukea värinää, vaikka laite, jolla sitä käytetään, ei tukisikaan. Tällöin värinää ei yksinkertaisesti tapahdu.

Näiden jälkeen video-muuttujaan asetetaan viittaus video-elementtiin ja asetetaan videon käynnistävä painonappi näkyväksi. Näiden jälkeen tulevissa ehtolauseissa tutkitaan ladataanko soittimeen uusi koodista tuleva tiedostonimi vai käytetäänkö vanhaa. Ensimmäinen if-lause tarkistaa onko soittimen src-ominaisuuden arvo tyhjä; mikäli on, asettaa sovellus soittimen lähdetiedostoksi QR-koodin sisältävän tiedoston. Mikäli soittimella on jo jokin lähdetiedosto, sovellus hakee kyseessä olevan tiedostonimen pätkimällä tiedoston polun ”/”-merkistä ja ottamalla siitä saadun taulukon viimeisimmän arvon. Tällä varmistetaan se, että kyseinen ratkaisu toimii minkä pituisella polulla hyvänsä. Tämän jälkeen verrataan olemassa olevaa lähdetiedostoa QR-koodista tulevaan tiedostonimeen. Mikäli ne eivät täsmää, asetetaan QR-koodin tiedosto lähdetiedostoksi, muussa tapauksessa soitin käyttää vanhaa lähdetiedostoa.

Tällainen tarkistus on hyvä tehdä, koska mikäli soittimelle asetetaan joka kerta uusi lähdetiedosto vaikkei se vaihtuisikaan, aiheuttaa se pidemmässä käytössä huomattavan muistinkulutuksen ja mahdollisesti selaimen muistivuodon (englanniksi memory leak). Omissa testeissäni jo kolmannen saman lähdetiedoston uudelleenasettamisen jälkeen videon toistaminen muuttui aivan käyttökelvottomaksi, eikä käytettävän videon koko ollut kuin reilu 1 megatavu.

Näiden jälkeen painonappiin liitetään tapahtumakäsittelijä, joka kuuntelee sen ”click”-tapahtumaa. Kun käyttäjä painaa nappia, ajetaan videoHandler-metodi, jonka sisällä kutsutaan kuvan 26 setCanvasDimensions-metodia, jossa kuvassa 22 luoduille canvas-elementeille asetetaan mitat, koska nyt sovellus tietää toistettavan videon koon.

```
videoHandler: function(){
    ont.kaynnistaVideo.style.display = "none";
    ont.setCanvasDimensions();
    ont.video.play();
    ont.convertVideo();
}, // videoHandler

setCanvasDimensions: function(){
    ont.tempCanvas.width = ont.video.videoWidth;
    ont.tempCanvas.height = ont.video.videoHeight;

    ont.videoCanvas.width = ont.video.videoWidth;
    ont.videoCanvas.height = ont.video.videoHeight;
}, // setCanvasDimensions
```

KUVA 26. VideoHandler ja setCanvasDimensions-metodit

Näiden toimenpiteiden jälkeen asetetaan painettu nappi näkymättömäksi, käynnistetään kyseessä oleva video ont.video.play()-komennolla ja tämän jälkeen kutsutaan convertVideo-metodia. Metodi ottaa videon jokaisesta kuvasta jokaisen kuvapisteen väriarvot ja vertaa niitä tietyn värin raja-arvoihin, joka halutaan asettaa läpinäkyväksi, tätä kutsutaan myös chroma keying-efektiksi. Tällä tavalla tietty väri saadaan poistettua videosta.

Kuvassa 27 on convertVideo-metodin ensimmäinen puolisko. Alussa luodaan väliaikaiset muuttujat, joiden arvoiksi asetetaan aiemmin luotujen canvas-elementtien sekä video-elementin mitat. Elementtien mittoja voisi käyttää suoraankin ilman erillisiä muuttujia, mutta koska niiden arvoja käytetään niin usein, voidaan muuttujiin asettamalla saada huomattava vaikutus suorituskykyyn (Faster Canvas… 2011).

```
convertVideo: function(){
    var tempW = ont.tempCanvas.width, tempH = ont.tempCanvas.height,
        vCanvasW = ont.videoCanvas.width, vCanvasH = ont.videoCanvas.height,
        videoW = ont.video.videoWidth, videoH = ont.video.videoHeight;

    ont.tempCtx.clearRect(0, 0, tempW, tempH);
    ont.videoCtx.clearRect(0, 0, vCanvasW, vCanvasH);
    ont.tempCtx.drawImage(ont.video,0,0, videoW, videoH);

    if(ont.video.ended){
        ont.koodinsisalto = false;
        ont.lueKoodi = false;
        ont.loop();
        ont.tempCtx.clearRect(0, 0, tempW, tempH);
        ont.videoCtx.clearRect(0, 0, vCanvasW, vCanvasH);
        if(window.cancelAnimationFrame) {
            window.cancelAnimationFrame(ont.vRafl);
        } else {
            clearTimeout(ont.vToisto);
        }
        return;
    }
```

KUVA 27. ConvertVideo-metodin alku

Ennen jokaista kuvan uudelleenpiirtoa molempien canvas-elementtien piirtokonteksti tyhjennetään. Tämän voi tehdä kahdella tapaa; joko palauttamalla canvas-elementin leveys ”ont.tempCanvas.width = ont.tempCanvas.width”, tai yllä olevassa kuvassa esitetyllä clearRect-metodilla. Näistä kuvassa käytetty vaihtoehto on nopeampi ja parempi tapa (HTML5 Canvas Optimization…2012). Video piirretään tilapäiselle canvas-elementille samalla tavalla kuin kameran kuva piirrettiin sille varatulle canvas-elementille. Tässä tapauksessa ei kuitenkaan haluta leikata kuvaa, vaan piirtää se kokonaisena. Tällöin argumenteiksi riittävät piirrettävä kuva, piirron aloituspiste sekä piirrettävän kuvan leveys ja korkeus.

Seuraava ehtolause tutkii onko video päättynyt. Mikäli video on päättynyt, asetetaan koodinsisalto-muuttuja epätodeksi, jotta uuden koodin lukeminen onnistuu ja käynnistetään loop-metodi. Myös tässä vaiheessa videon prosessointi- ja esityscanvas-elementtien kontekstit tyhjennetään. Lisäksi nollataan toistotapahtuma.

```
    var kuvadata = ont.tempCtx.getImageData(0,0, videoW, videoH);
    var kdPituus = kuvadata.data.length;

    for(var i = 0; i < kdPituus; i+=4) {
        r = kuvadata.data[i + 0]; // Red channel
        g = kuvadata.data[i + 1]; // Green channel
        b = kuvadata.data[i + 2]; // Blue channel

        if(r > 100 && g > 100 && b < 50) {
            kuvadata.data[i + 3] = 0; // Alpha channel
        }
    }
    ont.videoCtx.putImageData(kuvadata,0,0);

    if(window.requestAnimationFrame) {
        ont.vRafl = window.requestAnimationFrame(ont.convertVideo);
    }
    else {
        ont.vToisto = setTimeout(function(){
            ont.convertVideo();
        },33);
    }
    return;
} // convertVideo
```

KUVA 28. ConvertVideo-metodin loppuosa

ConvertVideo-metodin loppuosa sisältää ehkäpä työn mielenkiintoisimman osan. Tämä osa on omiaan esittelemään canvas-elementin mahdollisia käyttökohteita ja sitä, miten canvas- ja video-elementit toimivat yhdessä. Alussa kuvadata-nimiseen muuttujaan otetaan sen hetkisen piirrettävän kuvan datan sisältävä ImageData-olio canvas-elementin piirtokontekstin getImageData-metodilla. Tämä metodi ottaa argumenteiksi aloituspisteen canvas-elementin x- ja y-koordinaatteina, sekä halutun kuvan, tässä tapauksessa videon, koon. Tämän lisäksi muuttujaan kdPituus otetaan kuvadata-muuttujan data-ominaisuuden pituus kokonaislukuna. Data-ominaisuus sisältää getImageData-metodissa määritetyn nelikulmion sisältämän pikselidatan yksiulotteisessa taulukossa (GetImageData method 2013).

Seuraavassa for-silmukassa analysoidaan edellä mainittu pikselidata. Data-ominaisuus sisältää 4 eri arvoa jokaista kuvapistettä kohti; ne ovat RGBA-arvot, punainen (**R**ed), vihreä (**G**reen), sininen (**B**lue) ja läpinäkyvyys (**A**lpha). Näistä R-, G-, ja B-arvot otetaan talteen muuttujiin, jotta niitä vastaan voidaan verrata. Ehtolauseessa valitaan väri, joka halutaan poistaa, tässä tapauksessa se on kuvassa 29 näkyvä tummahkon keltainen. Käytän tässä työssä Mozillan tarjoamaa videota demonstroimaan sovelluksen

toimintaperiaatetta, koska työn varsinaiseen käyttötarkoitukseen tarkoitettuja videoita ei vielä ole.

KUVA 29. Video, jonka tausta tullaan poistamaan (Manipulating video… 2012)

Mikäli datan sisältämän kuvapisteen väriarvot osuvat ehtolauseen raameihin, muutetaan kyseisen kuvapisteen alpha-kanavan arvo nollaksi, joka tarkoittaa täysin läpinäkyvää; tässä käytetyn videon tapauksessa kaikkien kuvapisteiden alpha-kanava on alkuarvoltaan 255, joka tarkoittaa täysin läpinäkymätöntä. Tällä tavalla saadaan haluttu väri poistettua videosta. Värien manipuloinnin jälkeen saatu uusi pikselidata piirretään videoCanvas-elementille käyttämällä putImageData-metodia. Tällä on sama periaate kuin drawImage-metodilla, mutta joka piirrettävän kuvan sijaan ottaa piirrettävän kuvan pikselidatan. Lopuksi vielä käynnistetään toistorakenne, jotta videon jokainen kuva prosessoidaan, muuten vain ensimmäinen kuva muutettaisiin. Myös tässä olen toistamiseen käyttänyt requestAnimationFramea, mikäli selain sitä tukee. Varakeinona käytän jälleen setTimeout-metodia, jolle olen toistonopeudeksi antanut karkean 30 fps:n nopeuden (1000 / 30). Useimmiten kameroilla kuvatut videot ovat joko 23,976 tai 25 fps:n nopeudella pyöriviä, mutta esimerkiksi pc-peleistä kuvatut videot voivat olla pakkaamattomina jopa yli 100 fps:n nopeuksisia (Choosing a Frame Rate 2013).

KUVA 30. Työn lopputulos

Kuvassa 30 on työn lopputulos. Kuten kuvasta voi nähdä, keltainen on huono taustakankaan väri tähän tarkoitukseen. Koska tämän tapauksen keltainen väri on kovin lähellä ihmisihon värisävyjä, tapahtuu helposti kuvassakin näkyviä virheitä, kuten kasvojen leikkautumista. Tämän takia onkin hyvä käyttää esimerkiksi kirkasta sinistä tai vihreää taustakangasta. On myös hyvä huomata, että mikäli halutaan toistaa useita videoita työn kaltaisella toteutuksella, jokaisella videolla täytyy olla samanvärinen taustakangas.

4.2 Kokeelliset ominaisuudet

Tein työn lomassa kameran lisäksi muita sensoreita hyödyntäviä toteutuksia, kuten laitteen gyroskooppia hyödyntävää erikoistehostetta videon esittämisessä. Tämä tarkoittaa käytännössä sitä, että kallisteltaessa laitetta, hyödynnetään siitä saatavaa dataa, jonka avulla videota käännellään hieman näytöllä. Itse kallistusefekti tehdään käyttä-

mällä CSS3:n transform-ominaisuutta. Ideana tässä minulla oli saada hieman ”scifi-mäistä”, hologramminäytön tapaista tyyliä ja lisätyn todellisuuden tuntua.

Transform on yksi CSS:n 3-version ominaisuuksista. Sillä voidaan nimensä mukaisesti muokata halutun HTML-elementin muotoja ja sijaintia esimerkiksi venyttämällä tai kääntelemällä. Tähän tarkoitukseen käytän transformin rotateY-ominaisuutta, joka kiertää haluttua elementtiä sen Y-akselin ympäri annetun astemäärän verran, joka tässä tapauksessa saadaan laitteen kallistelusta.

```
<!DOCTYPE html>
<html>
<head>
<script src="script.js"></script>
<title>Transform-kokeilu</title>
<style>
.wrapper {perspective: 1000px; width: 590px;}
.inner {
    transform: rotateY(20deg);
    transition: 500ms ease-out;
    box-shadow: -20px 0px 10px rgba(200, 200, 200, 0.7)
}
</style>
</head>
<body>
    <div class="wrapper">
      <div class="inner" id="innerdiv">
        <video width="100%" controls>
            <source src="big_buck_bunny_480p_stereo.ogg"/>
        </video>
      </div>
    </div>
</body>
</html>
```

KUVA 31. Transform-testin pohja

Kuvassa 31 on luotu kokeilua varten HTML-pohja, johon olen upottanut CSS-tyylimääritykset. Uloimmalle div-elementille olen antanut perspective-ominaisuudelle arvoksi 1000px. Tämä luo annetulle elementille syvyyden, jolloin kääntelemällä sen sisällä olevia elementtejä saadaan aikaiseksi 3D-efekti. Tässä olen antanut inner-luokan omaavalle elementille alkuarvoksi 20 asteen kallistuksen oikealle, jolloin myös sen sisällä oleva video-elementtikin kääntyy. Box-shadow-ominaisuus antaa hyvän varjoefektin elementille.

KUVA 32. Video kierrettynä rotateY-ominaisuudella

Kuten kuvasta 32 näkee, edellä mainituilla tekniikoilla saadaan aikaiseksi melko hyvä 3D-efekti. Tämäkin tekniikka on vielä melko uutta, joten esimerkiksi kuvassa näkyvät ns. jagged edges, rosoiset reunat videon ympärillä, jotka kaipaisivat reunanpehmennystä.

```
function init()
{
    var inner = document.getElementById("innerdiv"), kulma = "";
    if(window.DeviceOrientationEvent)
    {
        window.addEventListener('deviceorientation', function(event){
            kulma = limitAngle(Math.round(event.beta));
            inner.style.transform = "rotateY("+(kulma*-1)+"deg)";
            inner.style.boxShadow = (kulma*0.5)+"px 0 10px rgba(200,200,200,.7)";
        }, false);
    }
}
window.addEventListener("load", init, true);
```

KUVA 33. Kallisteluefektin dynaamisuus JavaScriptilla

Kallisteluefektin luonti tapahtuu kuuntelemalla window-olion ”deviceorientation”-tapahtumaa, jonka kautta saadaan orientaatiodata. Tässä tapauksessa käytän kallisteluun beta- eli X-akselin arvoa, jolloin oletan, että tätä käytetään puhelin vaakatasossa. Rajoitan kulman määrän -20 - +20-asteeseen limitAngle-funktiolla, jota kuvassa ei

kuitenkaan esitetä. Elementin kiertämisen lisäksi liikuttelemalla box-shadow-ominaisuuden sijaintia saadaan aikaiseksi dynaamisen varjon efekti.

Syy, miksi tämä jäi vain kokeelliseksi ominaisuudeksi, on yksinkertainen: laitetehon puute. Testasin tätä tekniikka ensin omalla PC:lläni, jossa on Intelin i5-2500k-prosessori ja AMD Radeon HD 6950-näytönohjain, käyttämällä kallistuksen sijaan liukusäädintä, jolloin haluttu efekti toimi ihan hyvin ja suorituskyky oli hyvä. Siirryttäessä tehokkaasta peli-PC:stä mobiililaitteeseen, huomasi heti kuinka paljon tämä viritelmä vaatii laitetehoa. Mobiililaitteena käytin Samsungin Galaxy S2-älypuhelinta, jossa on 1,2 GHz tuplaydin-prosessori. Vaikka se onkin suhteellisen tehokas mobiililaitteeksi, kiertämällä toistettavaa videota reaaliajassa dynaamisen varjon kanssa, sulavuus katoaa olemattomiin. Testasin tätä videon lisäksi pelkillä staattisilla kuvilla ilman dynaamista varjoa, jolloin suorituskyky oli ihan hyvä. Uskoisin kuitenkin, että tällaisen luominen WebGL:llä olisi parempi vaihtoehto, jolloin voisi saada paremman suorituskyvyn, mutta se on tämän työn tutkimusongelman ulkopuolella.

5 PÄÄTÄNTÖ

Selainohjelmointitekniikat ovat kehittyneet viime vuosina hurjaa vauhtia, ja kehitys ei näytä hidastuvan. Tämänkin työn aikana jouduin muutamaan otteeseen päivittämään lähteitäni, koska eräiden kohtien kirjoitushetkestä muutaman päivän päästä W3C oli päivittänyt niitä koskevien rajapintojen kehitysasteita, esimerkiksi Geolocation API päivittyi pari päivää siitä kirjoittamisen jälkeen suositusehdotuksesta suositukseksi. On hienoa seurata kuinka tähän asti yksinomaan natiivikoodien kautta saadut ominaisuudet ja rajapinnat siirtyvät vähitellen myös verkkosovelluksiin, joiden avulla voidaan rakentaa entistä paremmin laitetta hyödyntäviä verkkosovelluksia.

GetUserMedia mahdollistaa sen minkä Flash on tarjonnut jo vuosia, mutta ilman mitään asennettavia lisäosia. GetUserMedia on osa WebRTC-projektia, jonka PeerConnection-rajapinnan avulla getUserMedia:lla voidaan luoda täysin selainpohjainen reaaliaikainen videopuhelusovellus. Flashin käyttöä puoltavat kuitenkin sen laitetuki, joka kattaa lähes minkä tahansa mobiili- ja työpöytälaitteen, sekä yhtenäinen toiminnallisuus eri alustojen välillä, jollaista työssä käytetyillä tekniikoilla on vielä lähes mahdotonta saada. Koska tässä työssä käsitellyt tekniikat ovat vielä melko tuoreita, ovat nii-

den tuki ja toiminnallisuus hyvin pitkälti selainvalmistajien käsissä ennen kuin varsinaiset standardit saadaan aikaiseksi.

Työn teon aikana törmäsin useaankin ongelmaan tai rajoitteeseen, joita eri selaimet asettivat. Monet näistä olivat sellaisia, joissa selaimet antoivat erilaista dataa samoista vaiheista. Tämä johtuu joko selainmoottorien eroista tai puhtaasti siitä, kuinka eri valmistajat ovat toteuttaneet kyseiset toiminnot. Esimerkiksi Mozillan Firefox-selain ei tarjonnut videon oikeaa kokoa saman tapahtuman aikana kuin Googlen Chrome-selain, joka taas asetti rajoitteita erilaisille ulkoasullisille ja toiminnallisille seikoille. Myös virheidenkäsittelyissä on eroja. Siinä missä Firefox antaa ”selkokielisen” virheilmoituksen getUserMedia-metodin virheenkäsittelyfunktiossa, Chrome antaa vastaavassa tilanteessa virheen olio-muodossa, jonka ominaisuudesta löytyisi kyseessä oleva virheilmoitus. Tämä olisi mahdollista kiertää esimerkiksi tutkimalla, mikä selain on kyseessä ja sen mukaan käsitellä virheet, mutta en toteuttanut sellaista tähän työhön.

Työn aikana tutustuin erityisesti canvas-läheisen ohjelmoinnin hyviin käytäntöihin ja optimointiin työn kohdealustojen vuoksi ja siksi, koska työssä käytetään useita canvas-elementtejä ja reaaliaikaista kuvapisteiden manipulointia. Kuvapisteiden manipulointiin canvas-elementin avulla on useita eri keinoja, joista osoitteessa http://jsperf.com/canvas-pixel-manipulation/34 on yksi lista ja niiden suorituskykytestit. Testin tulosten mukaan käytän tässä työssä huonointa mahdollista vaihtoehtoa. Tutustuin testissä käytettyihin tekniikoihin syvemmin työn loppupuolella, mutta aika ja oman osaamiseni puute bittiläheisistä operaattoreista kohtaan tulivat esteeksi, joten tähän osa-alueeseen jäi vielä kehitettävää.

Tätä työtä on mahdollista jatkokehittää vielä paljon. Sovelluksen ulkoasu on vielä tässä vaiheessa hyvin karu ja minimaalinen, koska tämä on hyvin varhainen prototyyppi. Tällä hetkellä sovellukseen on kovakoodattu väri, joka taustakankaalla täytyy olla, mutta esimerkiksi lisäämällä mahdollisuus valita väri, joka poistetaan kuvasta koskettamalla videon kuvaa, saadaan sovellus tukemaan periaatteessa minkä väristä taustakangasta vain. Myös erilaisten sensoreiden hyödyntäminen lisäisi mahdollisia käyttökohteita ja käyttötapoja; esimerkiksi lisäämällä paikkatieto, voitaisiin käyttäjä ohjata lähimmälle infopisteelle tai muuhun kohteeseen ja gyroskooppia hyödyntäen

lisätä lisätyn todellisuuden mahdollisuuksia. Uskonkin, että tulevaisuudessa tulemme näkemään paljon enemmän natiivisovelluksien kaltaisia verkkosovelluksia.

LÄHTEET

Adobe Flash: Why does Flash drain laptop batteries so quickly? 2013. Quora. Quora.com. WWW-dokumentti http://www.quora.com/Adobe-Flash/Why-does-Flash-drain-laptop-batteries-so-quickly. Ei päivitystietoa. Luettu 12.10.2013.

A first peek at Opera 15 for Computers 2013. Opera Developer News. My.opera.com. http://my.opera.com/ODIN/blog/2013/05/28/a-first-peek-at-opera-15-for-computers. Päivitetty 28.5.2013. Luettu 7.10.2013.

A re-introduction to JavaScript 2013. Mozilla Developer Network. Developer.mozilla.org. https://developer.mozilla.org/en-US/docs/Web/JavaScript/A_re-introduction_to_JavaScript. Päivitetty 17.9.2013. Luettu 25.9.2013.

All Standards and Drafts 2013. World Wide Web Consortium. W3.org. WWW-dokumentti. www.w3.org/TR/. Päivitetty 24.10.2013. Luettu 27.10.2013.

Ambient Light Events 2013. World Wide Web Consortium. W3.org. WWW-dokumentti. http://www.w3.org/TR/ambient-light/. Päivitetty 30.9.2013. Luettu 27.10.2013.

Bidelman, Eric 2012. HTML5 Rocks. Html5rocks.com. WWW-dokumentti. http://www.html5rocks.com/en/tutorials/getusermedia/intro/. Päivitetty 29.7.2013. Luettu 5.10.2013.

Can I use Geolocation 2013. Caniuse.com. WWW-dokumentti. http://caniuse.com/geolocation. Päivitetty 21.10.2013. Luettu 22.10.2013.

Can I use getUserMedia/Stream API? 2013 Caniuse.com. WWW-dokumentti. http://caniuse.com/stream. Päivitetty 7.10.2013. Luettu 7.10.2013.

Can I use the HTML5 canvas element? 2013. Caniuse.com. WWW-dokumentti. http://caniuse.com/canvas. Päivitetty 24.9.2013. Luettu 26.9.2013.

Can I use the HTML5 video element? 2013 Caniuse.com. WWW-dokumentti. http://caniuse.com/video. Päivitetty 28.9.2013. Luettu 30.9.2013.

Canvas 2013. Mozilla Developer Network. Developer.mozilla.org. https://developer.mozilla.org/en-US/docs/HTML/Canvas. Päivitetty 16.8.2013. Luettu 25.9.2013.

Chinnathambi, Kirupa 2013. Accessing Your Webcam in HTML5. Kirupa.com. WWW-dokumentti. http://www.kirupa.com/html5/accessing_your_webcam_in_html5.htm. Päivitetty 7.2.2013. Luettu 14.9.2013.

Choosing a Frame Rate 2013. Final Cut Pro 7 User Manual. Documentation.appel.com. WWW-dokumentti. http://documentation.apple.com/en/finalcutpro/usermanual/index.html#chapter=D%26section=4%26tasks=true. Ei päivitystietoa. Luettu 31.10.2013.

Cross-browser camera capture with getUserMedia/WebRTC 2013. Mozilla Hacks. Hacks.mozilla.org. WWW-dokumentti. https://hacks.mozilla.org/2013/02/cross-browser-camera-capture-with-getusermediawebrtc/. Päivitetty 13.2.2013. Luettu 17.10.2013.

DeviceOrientation Event Specification 2011. World Wide Web Consortium. W3.org. WWW-dokumentti. http://www.w3.org/TR/orientation-event/. Päivitetty 1.12.2011. Luettu 26.10.2013.

Dybwad, Barb 2010. Vimeo Joins The HTML5 Party With New Player. Mashable.com. WWW-dokumentti. http://mashable.com/2010/01/21/vimeo-html5-player/. Päivitetty 21.1.2010. Luettu 26.9.2013.

Explorercanvas 2009. HTML5 Canvas for Internet Explorer. Code.google.com. WWW-dokumentti. http://code.google.com/p/explorercanvas/. Ei päivitystietoa. Luettu 26.9.2013.

Facts About W3C 2013. World Wide Web Consortium. W3.org. WWW-dokumentti. http://www.w3.org/Consortium/facts. Päivitetty 30.9.2013. Luettu 1.10.2013.

Faster Canvas Pixel Manipulation with Typed Arrays 2011. Mozilla Hacks. Hacks.mozilla.org. WWW-dokumentti. https://hacks.mozilla.org/2011/12/faster-canvas-pixel-manipulation-with-typed-arrays/. Päivitetty 1.12.2013. Luettu 18.10.2013.

FAQ 2013. WHATWG Wiki. wiki.whatwg.org. WWW-dokumentti. http://wiki.whatwg.org/wiki/FAQ. Päivitetty 8.10.2013. Luettu 10.10.2013.

Geolocation API Specification 2013. World Wide Web Consortium. W3.org. WWW-dokumentti. http://www.w3.org/TR/geolocation-API/. Päivitetty 24.10.2013. Luettu 26.10.2013.

Geolocation API Specification Level 2 2011. World Wide Web Consortium. W3.org. WWW-dokumentti. http://www.w3.org/TR/geolocation-API-v2/. Päivitetty 30.11.2011. Luettu 26.10.2013.

GetImageData method 2013. Internet Explorer Dev Center. Msdn.microsoft.com. WWW-dokumentti. http://msdn.microsoft.com/en-us/library/ie/ff975418%28v=vs.85%29.aspx. Päivitetty 12.10.2013. Luettu 18.10.2013.

H.264 video in Firefox for Android 2012. Hacks.mozilla.org. WWW-dokumentti. https://hacks.mozilla.org/2012/11/h264-video-in-firefox-for-android/. Päivitetty 29.11.2012. Luettu 1.10.2013.

Heyes, Richard 2008. HTML5 canvas - an introduction. RGraph.net. WWW-dokumentti. http://www.rgraph.net/html5-canvas#history. Ei päivitystietoa. Luettu 30.9.2013.

HTML 4.0 Specification 1997. World Wide Web Consortium. W3.org. WWW-dokumentti. http://www.w3.org/TR/REC-html40-971218/. Päivitetty 18.12.1997. Luettu 24.9.2013.

HTML5 Canvas Optimization: A Practical Example 2012. Tutsplus. Dev.tutplus.com. WWW-dokumentti. http://dev.tutsplus.com/tutorials/html5-canvas-optimization-a-practical-example--active-11893. Päivitetty 28.6.2012. Luettu 18.10.2013.

HTML Audio/Video DOM Reference 2013. W3schools.com. WWW-dokumentti. http://www.w3schools.com/tags/ref_av_dom.asp. Ei päivitystietoa. Luettu 30.9.2013.

HTMLCanvasElement 2013. Mozilla Developer Network. Developer.mozilla.org. WWW-dokumentti. https://developer.mozilla.org/en-US/docs/Web/API/HTMLCanvasElement. Päivitetty 18.9.2013. Luettu 15.10.2013.

HTML Media Capture 2013. World Wide Web Consortium. W3.org. WWW-dokumentti. http://www.w3.org/TR/html-media-capture/. Päivitetty 7.5.2013. Luettu 14.9.2013.

HTML Standard Tracker 2011. Html5.org. WWW-dokumentti. http://html5.org/tools/web-apps-tracker?from=5944&to=5945. Päivitetty 14.3.2011. Luettu 7.10.2013.

HTML5 Video 2013. W3schools.com. WWW-dokumentti. http://www.w3schools.com/html/html5_video.asp. Ei päivitystietoa. Luettu 30.9.2013.

iPhone 4 gyroscope X-rayed, like to be added to future iPad 2010. AppleInsider. Appleinsider.com. WWW-dokumentti. http://appleinsider.com/articles/10/06/30/iphone_4_gyroscope_x_rayed_likely_to_be_added_to_future_ipad.html. Päivitetty 30.6.2010. Luettu 20.10.2013.

JavaScript and HTML DOM Reference 2013. W3schools.com. WWW-dokumentti. http://www.w3schools.com/jsref/. Ei päivitystietoa. Luettu 30.9.2013.

JavaScript Language Resource 2013. Mozilla Developer Network. Developer.mozilla.org. WWW-dokumentti. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Language_Resources?redirectlocale=en-US&redirectslug=JavaScript%2FLanguage_Resources. Päivitetty 17.7.2013. Luettu 25.9.2013.

Javascript Lesson 9 2008. Prologic.com. WWW-dokumentti. http://www.proglogic.com/learn/javascript/lesson9.php. Ei päivitystietoa. Luettu 30.9.2013.

Korpela, Jukka K 2011. HTML5: Uudet ominaisuudet. Porvoo: Bookwell Oy.

LePage, Pete 2011. This End Up: Using Device Orientation. HTML5 Rocks. Html5rocks.com. WWW-dokumentti. http://www.html5rocks.com/en/tutorials/device/orientation/. Päivitetty 29.5.2011. Luettu 26.10.2013.

Malone, Simon 2010. What is Chroma Keying and how do green screens work? Virtualstudio.tv. WWW-dokumentti. http://www.virtualstudio.tv/blog/post/14-what-is-chroma-keying-and-how-do-green-screens-work. Päivitetty 20.8.2010. Luettu 26.9.2013.

Manipulating video using canvas 2012. Mozilla Developer Network. Developer.mozilla.org. WWW-dokumentti. https://developer.mozilla.org/en-US/docs/HTML/Manipulating_video_using_canvas. Päivitetty 16.12.2012. Luettu 2.11.2013.

Marshall, Gary 2011. HTML5: what is it? Techradar.com. WWW-dokumentti. http://www.techradar.com/news/internet/web/html5-what-is-it-1047393. Päivitetty 13.12.2011. Luettu 14.9.2013.

Media formats supported by the HTML audio and video elements 2013. Mozilla Developer Network. Developer.mozilla.org. WWW-dokumentti. https://developer.mozilla.org/en-US/docs/HTML/Supported_media_formats#Browser_compatibility. Päivitetty 20.9.2013. Luettu 30.9.2013.

Media Capture in mobile browsers 2013. Dev.Opera. Dev.opera.com. WWW-dokumentti. http://dev.opera.com/articles/view/media-capture-in-mobile-browsers/. Päivitetty 26.9.2013. Luettu 5.10.2013.

Native webcam support and orientation events - technology preview 2011. Opera Core Concerns. My.opera.com. WWW-dokumentti. http://my.opera.com/core/blog/2011/03/23/webcam-orientation-preview. Päivitetty 14.3.2011. Luettu 7.10.2013.

Navigator.getUserMedia 2013. Mozilla Developer Network. Developer.mozilla.org. WWW-dokumentti. https://developer.mozilla.org/en-US/docs/Web/API/Navigator.getUserMedia. Päivitetty 28.9.2013. Luettu 12.10.2013.

Object.prototype 2013. Mozilla Developer Network. Developer.mozilla.org. WWW-dokumentti. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/prototype#Examples. Päivitetty 24.8.2013. Luettu 4.10.2013.

Orientation and motion data explained 2013. Mozilla Developer Network. Developer.mozilla.org. WWW-dokumentti. https://developer.mozilla.org/en-US/docs/Web/Guide/API/DOM/Events/Orientation_and_motion_data_explained. Päivitetty 6.9.2013. Luettu 22.10.2013.

Plan 2014 2012. World Wide Web Consortium. W3.org. WWW-dokumentti. http://dev.w3.org/html5/decision-policy/html5-2014-plan.html. Päivitetty 1.10.2012. Luettu 24.9.2013.

Proximity 2013. Mozilla Developer Network. Developer.mozilla.org. WWW-dokumentti. https://developer.mozilla.org/en-US/docs/WebAPI/Proximity. Päivitetty 27.7.2013. Luettu 27.10.2013.

Proximity Events 2013. World Wide Web Consortium. W3.org. WWW-dokumentti. http://www.w3.org/TR/proximity/. Päivitetty 30.9.2013. Luettu 27.10.2013.

Raggett, Dave 1997. HTML 3.2 Reference Specification. W3.org. WWW-dokumentti. http://www.w3.org/TR/REC-html32. Päivitetty 5.4.1999. Luettu 24.9.2013.

requestAnimationFrame 2013. CreativeJS. Creativejs.com. WWW-dokumentti. http://creativejs.com/resources/requestanimationframe/. Ei päivitystietoa. Luettu 14.10.2013.

Silverman, Matt 2012. The History of HTML5. Mashable.com. WWW-dokumentti. http://mashable.com/2012/07/17/history-html5/. Päivitetty 17.7.2012. Luettu 24.9.2013.

Simple HTML5 video player with Flash fallback and custom controls 2011. Dev.opera.com. WWW-dokumentti. http://dev.opera.com/articles/view/simple-html5-video-flash-fallback-custom-controls/. Päivitetty 17.12.2011. Luettu 30.9.2013.

Taking webcam photos 2013. Mozilla Developer Network. Developer.mozilla.org. WWW-dokumentti. https://developer.mozilla.org/en-US/docs/WebRTC/taking_webcam_photos. Päivitetty 12.7.2013. Luettu 13.10.2013.

Teeple, Kim S 2012. Coding Vendor Prefixes with JavaScript. Developerdrive.com. WWW-dokumentti. http://www.developerdrive.com/2012/03/coding-vendor-prefixes-with-javascript/. Päivitetty 21.3.2012. Luettu 11.9.2013.

The hierarchy of JavaScript Objects 2006. Sislands.com. WWW-dokumentti. http://www.sislands.com/coin70/week1/dom.htm. Päivitetty 30.9.2006. Luettu 30.9.2013.

Uj-Jaman, Asad 2011. Sensors in Smartphones. Mobiledeviceinsight.com. WWW-dokumentti. http://mobiledeviceinsight.com/2011/12/sensors-in-smartphones/. Päivitetty 1.12.2011. Luettu 11.9.2013.

Unity scripting 2013. Unity3d.com. WWW-dokumentti. http://unity3d.com/unity/workflow/scripting. Luettu 25.9.2013.

Using JavaScript to control the HTML5 video player 2013. Internet Explorer Dev Center. Msdn.microsoft.com. http://msdn.microsoft.com/en-us/library/ie/hh924823%28v=vs.85%29.aspx. Päivitetty 30.8.2013. Luettu 30.9.2013

Video Conferencing with the HTML5 Device Element 2010. Ajaxian.com. WWW-dokumentti. http://ajaxian.com/archives/video-conferencing-with-the-html5-device-element. Päivitetty 20.9.2010. Luettu 7.10.2013.

Video on the web 2013. Diveintohtml5.info. WWW-dokumentti. http://diveintohtml5.info/video.html. Päivitetty 25.1.2013. Luettu 26.9.2013.

Vroom, Jeff 2012. Choosing Your Programming Language. WWW-dokumentti. http://blog.jvroom.com/2012/01/28/choosing-your-programming-language-the-inside-scoop/. Päivitetty 28.2.2012. Luettu 25.9.2013.

W3C Mission 2013. World Wide Web Consortium. W3.org. WWW-dokumentti. http://www.w3.org/Consortium/mission.html. Päivitetty 20.9.2013. Luettu 30.9.2013.

Web Browser Plugin Market Share 2012. Statowl.com. WWW-dokumentti. http://web.archive.org/web/20121220235707/http://statowl.com/plugin_overview.php. Ei päivitystietoa. Luettu 1.10.2013.

What is ECMAScript? 2012. Programmerinterview.com. WWW-dokumentti. http://www.programmerinterview.com/index.php/javascript/javascript-what-is-ecmascript/. Päivitetty 7.5.2012. Luettu 30.9.2013.

What's the difference between a compiled and interpreted language? 2013. Programmerinterview.com. WWW-dokumentti. http://www.programmerinterview.com/index.php/general-miscellaneous/whats-the-difference-between-a-compiled-and-an-interpreted-language/. Ei päivitystietoa. Luettu 30.9.2013.

Wijering, Jeroen 2013. The State of HTML5: Firefox Now Supports MP4! WWW-dokumentti. http://www.longtailvideo.com/blog/34576/the-state-of-html5-firefox-now-supports-mp4/. Päivitetty 12.8.2013. Luettu 30.9.2013

Window.requestAnimationFrame() 2013. Mozilla Developer Network. Developer.mozilla.org. WWW-dokumentti. https://developer.mozilla.org/en-US/docs/Web/API/window.requestAnimationFrame. Päivitetty 25.9.2013. Luettu 15.10.2013.

You Are Here (And So Is Everybody Else) 2013. Dive Into HTML5. Diveintohtml5.info. WWW-dokumentti. http://diveintohtml5.info/geolocation.html. Päivitetty 25.5.2013. Luettu 23.10.2013.