WebGL-SOVELLUS

Joonas Vainionpää

Opinnäytetyö
Toukokuu 2015
Tietotekniikan koulutusohjelma
Sulautetut järjestelmät ja
elektroniikka

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Sulautetut järjestelmät ja elektroniikka

VAINIONPÄÄ, JOONAS
WebGL-sovellus

Opinnäytetyö 31 sivua
Toukokuu 2015

WebGL:n on JavaScriptiä käyttävä ohjelmointirajapinta, jonka avulla pystytään mallintamaan 3D- ja 2D-grafiikkaa mille tahansa WebGL-yhteensopivalle nettiselaimelle kuten Firefox ja Chrome. Grafiikan mallintamiseen ei tarvita ylimääräisiä liitännäisiä ja kaikki mallintaminen suoritetaan tietokoneen näytönohjaimen avulla. Grafiikan mallintamisessa käytettävät keskeiset teknologiat ovat HTML5-merkkauskieli ja JavaScript.

Tavoitteena oli tehdä reaaliaikainen graafinen audiovisualisointi, missä kuvataan kuutioiden avulla ääninäytteessä esiintyviä eri taajuuksia. Ääninäytettä analysoidaan Web Audio API:lla, jonka avulla äänestätiedostosta erotellaan kaikki siinä esiintyvät taajuudet. Jokaisen taajuuden arvo tallennetaan JavaScriptilla luotuun taulukkoon ja näitä arvoja käytetään kuutioiden muodon muuttamiseen reaaliajassa. Ääninäytteen analysointia varten on hyvä tietää digitaalisen signaalinkäsittelyn perusteet.

Yksinkertainen WebGL:n ja Web Audio API:n avulla tehty reaaliaikainen graafinen audiovisualisointi voidaan sisällyttää kaikkiin HTML5-versiolla tehtyihin nettisivuihin. Sen jokaista ominaisuutta voidaan helposti muokata toimimaan käyttäjän haluamalla tavalla. Ominaisuuksien muokkaaminen edellyttää hyvää JavaScrit-ohjelmointikielen osaamista, sekä hyvää kolmiulotteisten tapahtumien hahmotuskykyä.

Avainsanat: WebGL, Web Audio API, JavaScript, HTML5

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Computer Science and Engineering
Embedded Systems and Electronics

VAINIONPÄÄ, JOONAS
WebGL Application

Bachelor's thesis 31 pages
May 2015

WebGL is a JavaScript application programming interface for rendering 3D and 2D graphics for any compatible web browser like Firefox and Chrome. No additional plugins are required for the rendering because the rendering is executed by computers graphics processor unit. The main technologies that are used in the rendering process are HTML5 markup language and JavaScript.

The purpose was to build real-time graphic audio visualization where different sound frequencies are described using cubes. The sound sample is analyzed with Web Audio API which can separate all the frequencies that are present. Every frequency value is saved within JavaScript array and those values are used to manipulate the cubes shape in real-time. It is good to have a basic knowledge in digital signal processing when analyzing the sound sample.

A simple real-time graphic audio visualization done by using WebGL and Web Audio API can be implemented in any web site made by using HTML5. Every feature of the visualization can be easily changed in every way the user wants. Modifying the features will require good knowledge in JavaScript and good perception of three-dimensional events.

Key words: WebGL, Web Audio API, JavaScript, HTML5

SISÄLLYS

1 JOHDANTO

Tässä opinnäytetyössä esitellään WebGL-ohjelmointirajapinnan käyttöä graafisen audiovisualisoinnin avulla. Tarkoituksena on selvittää, miten graafisia kuvioita muodostetaan Three.js-apukirjaston avulla. Lisäksi tutustutaan keskeisiin tekniikoihin, jotka mahdollistavat dynaamisen nettisivujen kehittämisen, ja perehdytään digitaaliseen signaalinkäsittelyyn Web Audio API:n avulla.

Luvussa kaksi esitellään graafisen pohjan muodostamiseen vaadittavat tekniikat, kuten HTML-merkkauskieli ja JavaScript. Esitellään HTLM5-version ominaisuuksia ja selvitetään niiden käyttöä graafisessa mallinnuksessa. Digitaalisen signaalinkäsittelyn kappaleessa tarkastellaan äänen analysointiin liittyviä termejä kuten näytteenottotaajuus, Fourier-muunnos, aikataso ja taajuustaso. Luvun kaksi viimeisessä kappaleessa selvitetään WebGL:n taustoja ja tarkoitusta.

Luvussa kolme perehdytään äänen analysointiin ja esitellään Web Audio Apin käyttöä. Esitellään, miten JavaScriptillä muodostetaan audiokonteksti, ja mitä kaikkea se sisältää. Selvitetään, miten ääninäytteestä eritellään siinä esiintyvät eri taajuudet, ja esitetään miten näytteenottotaajuus vaikuttaa äänen analysointiin. Lopuksi esitetään, miltä näyttää taajuustason esitys nettiselaimella.

Neljännessä luvussa selvitetään, miten graafiset kuviot muodostetaan WebGL:n avulla, ja miten ääninäytteestä saaduilla taajuuden arvoilla muutetaan kuvoiden muotoa. Esitellään, miten Three.js-kirjastossa olevia kuvioiden muodostamiseen tarvittavia olioita käytetään. Tutkitaan, miten kolmiulotteinen kuvio modostetaan kärkipisteiden, särmien ja tahkojen avulla. Neljännen luvun viimeisesssä kappaleessa pohditaan graafisen mallinnuksen jatkokehitystä ja esitellään avaruuden mallintaminen uutena teoksena.

2 TEKGNOLOGIAT

Tässä luvussa tarkastellaan opinnäytetyön käsittelyssä käytettyjä tekniikoita ja selvitetään niiden taustoja. Tärkeimpinä tekijöinä opinnäytetyössä ovat HTML-merkkauskieli ja sen uudempi versio HTML5, jonka avulla saadaan opinnäytetyö esityskelpoiseen muotoon. Digitaalisesta signaalinkäsittelystä esitellään mahdollisimman yksinkertaisesti siinä käytettävät kaavat ja niiden merkitykset. Lisäksi esitellään JavaScript-komentosarjakielen käyttöä ja selvitetään miten se liittyy WebGL:ään.

2.1 HTML

Hypertekstimerkintäkieli (HTML, Hypertxt Markup Language) on standardoitu web-sivujen luontiin tarkoitettu merkintäkieli. HTML syntyi vuonna 1989 CERNissä, kun Tim Berners-Lee halusi kehittää tavan ylläpitää, esittää ja päivittää dokumentteja. Tarkoituksen oli myös, että dokumentit olisivat kaikkien saatavilla. Näin syntyi World Wide Web eli lyhyesti webbi tai WWW, joka on Internet-verkossa toimiva hajautettu hypertekstijärjestelmä. Vuonna 1991 CERNissä pystytetyn ensimmäisen WWW-palvelimen rinnalle suunniteltiin HTTP-protokolla (Hypertext Transfer Protocol), jota WWW-palvelimet ja selaimet käyttävät tiedonsiirrossa. Kuvassa 1 on esimerkki hypertekstimerkintäkielen käytöstä. (HTML Introduction 2015.)

```
<!DOCTYPE html>
<html>
    <head>
        <title>Sivun Otsikko</title>
    </head>
    <body>
        <h1>Ensimmäinen kappale</h1>
        <p>Ensimmäinen kappale.</p>
    </body>
</html>
```

KUVA 1. HTML-dokumentti index.html

HTML koostuu tunnisteista eli tägeistä, jotka ovat niin sanottuja avainsanoja sivun sisällön luomisessa. Tunnisteet muodostuvat pareista, aloitus- ja lopetustunniste, joita molempia ympäröi kulmasulut. Alku- ja lopetustunnisteen erottaa toisistaan lopetustunnisteessa oleva vinoviiva (kuva 2). Tunnisteet ja niiden välissä oleva sisältö muodostavat kokonaisuutena HTML-elementin. HTML-dokumentti rakentuu sisäkkäisistä elementeistä. Kuvan 1 esimerkki dokumentti sisältää kuusi elementtiä, joita ovat html, head, title, body, h1 ja p. Kuvassa 1 ensimmäisellä rivillä oleva DOCTYPE-määritelmä ei ole tunniste, vaan se kertoo selaimelle mitä HTML-versiota tulee käyttää. Kuvan 1 esimerkissä selaimen käyttöön määritellään HTML5-versio. (HTML Introduction 2015.)

```
<tunniste>sisältö</tunniste> //elementti
```

KUVA 2. HTML-elementti kokonaisuudessaan

Selaimen tehtävänä on lukea HTML-dokumentti ja näyttää sen sisältö (kuva 3). HTML-dokumentin tiedostonimen voi tallentaan haluamallaan nimellä, mutta kannattaa välttää isoja kirjaimia, välilyöntejä ja ääkkösiä. Tämä johtuu siitä, että osa palvelimista ei osaa lukea näitä merkkejä. Tiedostopääteen tulee olla .html tai .htm. Yleisesti käytetään tiedostonimeä index.html, koska palvelin antaa oletuksena tämän nimisen tiedoston selaimelle. Kuvassa 3 on selaimella avattu kuvan 1 HTML-dokumentti. Tunnisteet eivät näy selaimessa, vaan pelkästään elementin sisältö, ja tunnisteen rakenne määrittää sen, miten elementin sisältö näkyy selaimessa. Esimerkkinä kuvassa 3 ensimmäisen otsikon kirjasinkoko on suurempi, kuin ensimmäisen kappaleen. (HTML Introduction 2015.)

KUVA 3. HTML dokumentti avattu selaimella

Alkutunnisteeseen voidaan sisällyttää attribuutteja, joiden arvoilla voidaan muokata elementin sisältöä tarkemmin. Kuvassa 4 alkutunniste saa attribuuttina tyylin, joka määrittää sen, minkä värisenä elementti esitetään selaimessa. HTML:n alkuperäinen tarkoitus oli tyylitön tekstin esittämien, mutta hyvin nopeasti sivujen tekijät halusivat alkaa vaikuttaa dokumentin ulkoasuun. HTML-dokumentille kehiteltiin CSS-tiedosto, jonka avulla pystytään määrittelemään dokumentille useita tyyliohjeita. (HTML Introduction 2015.)

KUVA 4. Elementin tyylin muuttaminen attribuutilla

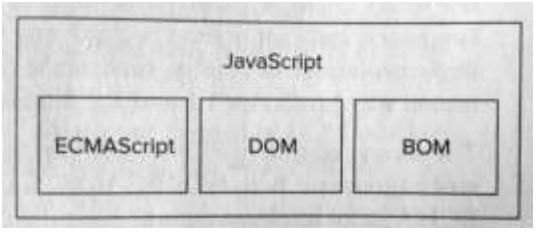
2.2 HTML5

HTML5 on uusi versio HTML-merkintäkielestä, joka julkaistiin vuonna 2014. HTML5-version saa käyttöön, kuten kuvan 1 ensimmäisellä rivillä on kirjoitettu. Vanhempien HTML-versioiden käyttöönottaminen vaatii DOCTYPE-määritelmään lisäämään ylimääräsiä attribuutteja, jotka tarkistavat HTML-dokumentin oikeinkirjoituksen ja muotoilun. Attribuuttien lisääminen on pakollista vanhemmissa HTML-versioissa. HTML5-versiossa muotoilu ja tarkistus määritellään yhdellä attribuutilla ja tässä opinnäytetyössä kaikki esimerkit käyttävät HTML5-versiota. HTML5-uudistus tuo mukanaan paljon uusia tunnisteita, joilla pystytään rakentamaan uusia elementtejä. HTML5-versiota käytetäänkin yleisnimenä nykyaikaisille web-tekniikoille, joihin kuuluuvat CSS:n uudet ominaisuudet ja erilaiset ohjelmointirajapinnat eli API:t. Yksi HTML5:den uutuuksista on canvas-elementti, joka toimii pohjana graafisille kuvioille ja teksteille. Elementti ei itsessään pysty kuvioita piirtämään, vaan ne on tehtävä skriptikielellä kuten JavaScript. (HTML(5) Tutorial 2015.)

2.3 JavaScript

Internetin alkuaikoina palvelinpuolen ohjelmointikielet, kuten Python, käsittelivät nettisivun jokaisen lomakekentän sisällön tarkistuksen. Kentän sisällöstä lähetettiin tieto palvelimelle, jossa se tarkistettiin, ettei kenttä ollut tyhjä ja että sen arvo oli oikea. Tiedon välitys palvelimelle ja takaisin saattoi kestää 30 sekuntia. Jos kentän sisältö ei ollut vaatimusten mukainen tai kenttä oli jätetty tyhjäksi, jouduttiin se täyttämään uudestaan ja tekemään uusi tarkistus palvelimella. Netscape oli tuohon aikaan teknologian kehittämisen kärkiyhtiöitä ja alkoi suunnitella asiakaspuolen komentosarjakieltä, joka yksinkertaistaisi nettisivujen prosessointia. (Zakas 2012.)

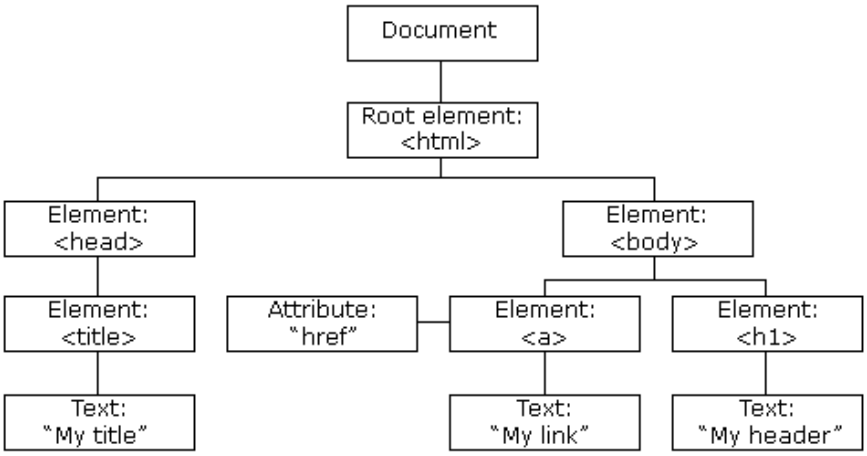
JavaScript ilmestyi ensimmäisen kerran vuonna 1995 Netscape Navigator 2:n yhteydessä. Brendan Eich, joka työskenteli Netscape yhtiössä tuohon aikaan, kehitteli komentosarjakieltä eli skriptikieltä nimeltä Mocha, jonka nimi muutettiin myöhemmin LiveScriptiksi. Nimi muutettiin lopullisesti JavaScriptiksi Netscapen liittouduttua Sun Microsystemsin kanssa, joka tapahtui juuri ennen Netscape Navigator 2:n julkaisua. JavaScript oli ensimmäinen skriptikieli, joka pystyi hoitamaan lomakekenttien tiedon tarkistuksen. (Zakas 2012.)



KUVA 5. JavaScriptin sisältämät kolme osaa (JavaScript for Web Developers, 2012)

JavaScript määriteltiin vuonna 1997 ECMA-262-standardin mukaiseksi, joka määrittää kaikki ECMAScriptit. ECMA-262-standardilla määritellään mm. ohjelmointikielen syntaksi, operaatiot ja oliot. JavaScriptiä ja ECMAScriptiä käytetään usein toistensa synonyymeinä, vaikka JavaScript on paljon muuta, mitä ECMA-262-standardi määrittelee (kuva 5). (Zakas 2012.)

JavaScriptillä pystytään muokkaamaan dokumenttioliomalleja (DOM, Document Object Models) ja selaimessa esiintyvien olioiden malleja (BOM, Browser Object Models). BOM:ia ei ole standardoitu mitenkään, joten selainvalmistajat saavat vapaasti päättää sen käyttöönotosta. DOM:ia voidaan ajatella seuraavanlaisen kuvan avulla (kuva 6). (Zakas 2012.)



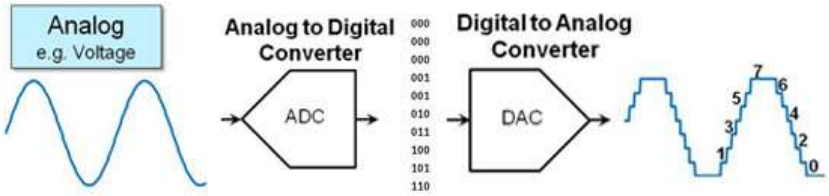
KUVA 6. DOM:in rakennelma on puumainen (W3Schools)

Selain muodostaa HTML-dokumentista DOM:in, kun se ladataan selaimelle, ja näin saadaan kuvan 6 mukainen puurakennelma, joka muodostuu olioista. Näin määritellään, kuinka dokumentissa olevat elementit välittävät tietoa toisille ja kuinka näihin elementteihin voidaan viitata. JavaScriptillä pystytään muokkaamaan dokumenttioliomalleja, minkä avulla sivuista saadaan dynaamisia. JavaScriptillä voidaan esimerkiksi muokata kaikkia elementtejä, mitä sivulla esiintyy, sekä niiden attribuutteja ja tyylejä. (Zakas 2012.)

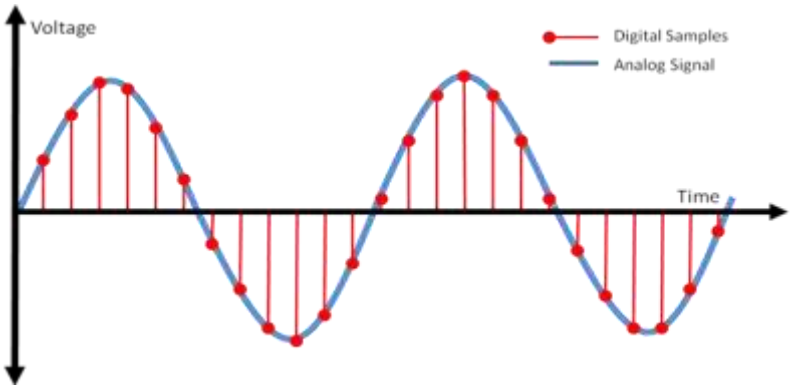
2.4 Digitaalinen signaalinkäsittely

Digitaalisessa signaalinkäsittelyssä analoginen signaali muunnetaan digitaalisiksi lukuarvoiksi A/D-muuntimella (kuva 7). Tämän jälkeen digitaalisessa muodossa olevaa signaalia voidaan muokata digitaalisella signaaliprosessorilla, mitä kutsutaan suodattamiseksi. Suodattamalla saadaan signaalista poistettua esimerkiksi kohina. Digitaalisilla suotimilla voidaan myös toteuttaa täysin lineaarivaiheinen suodin, mikä

tarkoittaa sitä, että kaikki signaalin sisältämät taajuudet viivästyvät yhtä paljon. Suodatuksen jälkeen signaali muutetaan takaisin analogiseksi signaaliksi D/A-muuntimella. (Huttunen 2014, 5.)



KUVA 7. Signaalin käsittelyä analogisesta digitaaliseksi (An Introduction to Electronic Measurement)



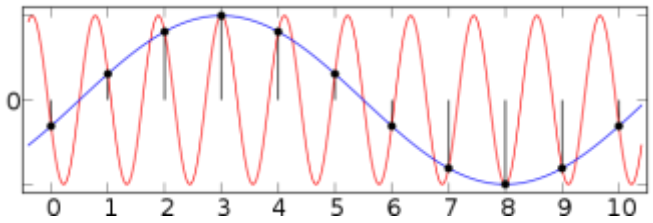
KUVA 8. Näytteistys (Signal Path Series)

Analoginen signaali on jatkuva-aikaista eli jokaiselle ajanhetkelle on olemassa sitä vastaava arvo. Kuten kuvassa 7 on esitetty, A/D-muuntimella saadaan analoginen signaali muutettua digitaalisiksi lukuarvoiksi eli näytteiksi. Tätä kutsutaan myös näytteistämiseksi, jossa jokaisen näytteen arvo on verrannollinen mitatun signaalin samalla ajanhetkellä olevaan arvoon (kuva 8). (Huttunen 2014, 5)

Se kuinka monta kertaa näytteitä halutaan ottaa signaalista sekunnissa on näytteenottotaajuus f_s , joka saadaan seuraavasta kaavasta

$$f_s = \frac{1}{T} \text{ , missä } T = \text{näytteiden väli sekunneissa} \quad (1)$$

Jos näytteenottotaajuus on liian pieni suurempitaajuiselle signaalille, tapahtuu laskostumista eli alinäytteistymistä. Laskostuminen on helpoin selittää kuvan 9 mukaisella kuviolla. Kuvassa punaisella piirrettystä signaalista otetaan näytteitä, jotka on kuvassa merkattu mustilla palloilla. Näistä näytteistä muodostetaan D/A-muuntimella uusi kuvassa sinisellä piirretty signaali. Sinisellä piirretty signaali ei vastaa kuvioltaan punaisella piirrettyä signaalia, koska näytteenottotaajuus on ollut liian alhainen ja on tapahtunut laskostuminen. (Huttunen 2014, 4.)



KUVA 9. Alinäytteistyminen (Aliaising)

Oikea näytteenottotaajuus saadaan määriteltyä Nyquistin teoreeman mukaisesti, joka on seuraavan kaavan mukainen

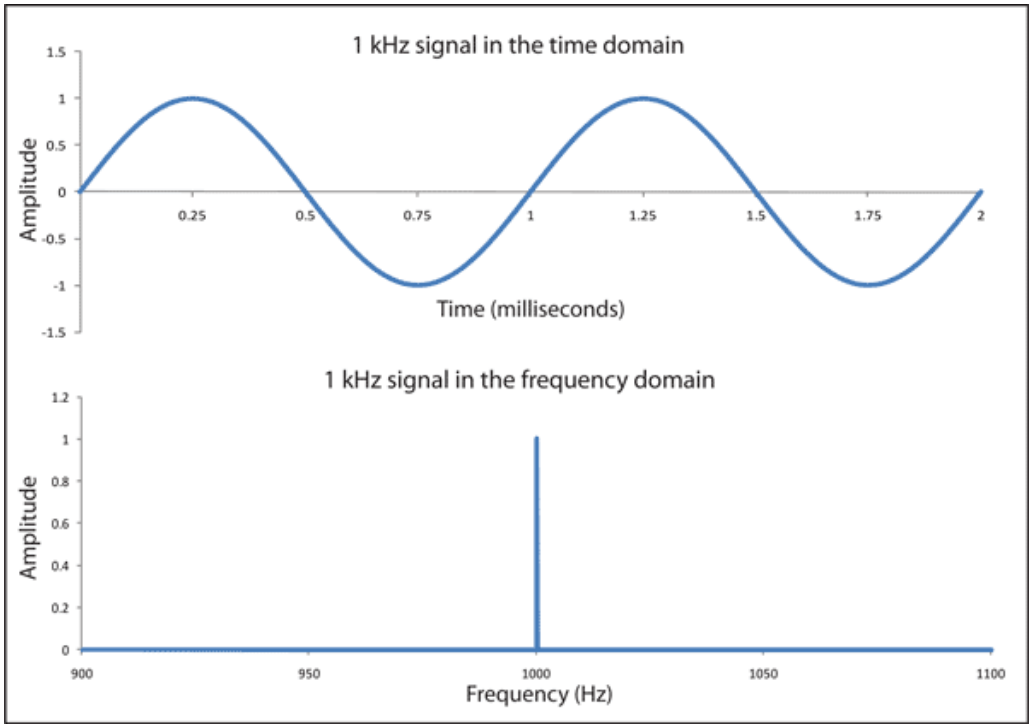
$$f_{max} = \frac{f_s}{2} <=> \ f_s \ = 2 * f_{max} \quad (2)$$

missä

$$f_{max} = \text{signaalin suurin taajuus}$$

Kaavan (2) mukaisesti näytteenottotaajuuden tulee olla vähintään kaksi kertaa niin suuri, kuin signaalissa esiintyvän korkeimman taajuuden suuruus. Korkeimmasta taajuudesta käytetään myös nimitystä Nyquistin taajuus. Ihmiskorva pystyy kuulemaan taajuuksia väliltä 20 Hz – 20 kHz, jolloin näytteenottotaajuuden tulee olla vähintään 40 kHz. Yleisesti käytettyjä näytteenottotaajuuksia musiikkiteollisuudessa ovat 44,1 kHz ja 48 kHz. Näillä taajuuksilla saadaan katettua kaikki ihmiskorvalle kuuluvat äänet. Näitä korkeammat näytteenottotaajuudet ovat musiikin kuuntelun kannalta tarpeettomia ja hyödyttömät näytteet käyttävät turhaan tallennustilaa. (Henderson. 2015.)

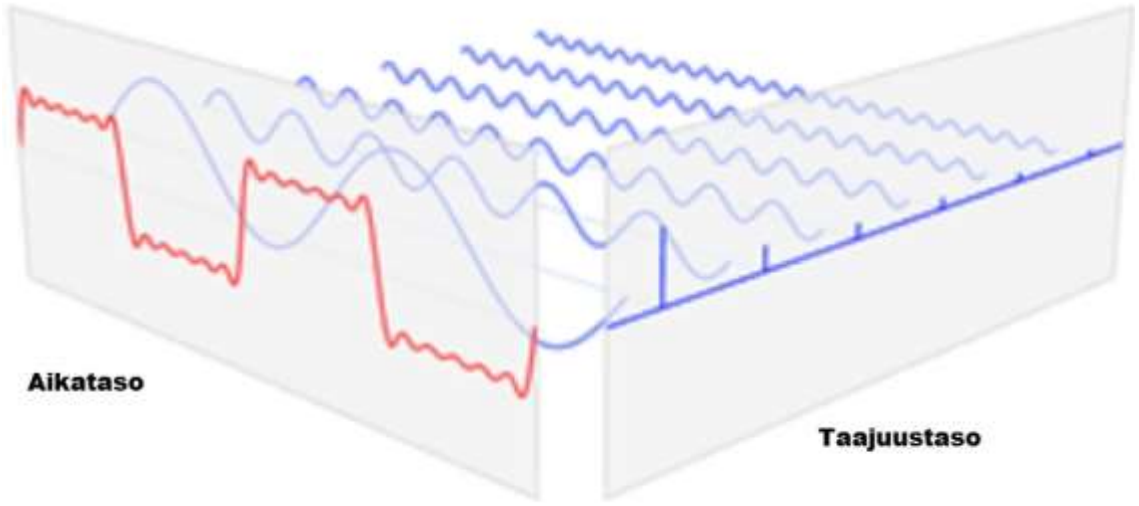
Kuten kuvista 8 ja 9 huomataan signaalit esitetään aikatasossa eli signaalin amplitudi ilmoitetaan ajan suhteen. Jotta signaalista saadaan tieto kuinka paljon se sisältää eri taajuuksia, on sille tehtävä Fourier-muunnos. Fourier'n muunnos ilmaisee taajuustasossa, kuinka paljon kyseistä taajuutta esiintyy signaalissa (kuva 10).



KUVA 10. Aikaesitys kuvassa ylhäällä ja taajuusesitys kuvassa alhaalla (Minelab Terminology)

Kuvan 10 esimerkissä on ylhäällä esitetty 1 kHz:n sinisignaali ajan suhteen ja alhaalla sama signaali, jolle on tehty Fourier-muunnos. Muunnostuloksena saadaan 1 kHz:n kohdalle palkki, joka kertoo kyseisen taajuuden amplitudiarvon. Fourier'n muunnos perustuu siihen, että kaikki jatkuvat riittävän säännölliset funktiot voidaan ilmaista sinimuotoisten funktioiden integraalina. (Huttunen 2014, 33.)

Digitaalisessa signaalinkäsittelyssä käytetään enimmäkseen diskreettiä Fourier-muunnosta, jolloin laskut saadaan matriisimuotoon ja ne ovat näin ollen äärellisiä ja helppoja. Diskreetissä Fourier'n muunnoksessa korvataan Fourier-muunnoksen integraali sinisignaalien summalausekkeella (kuva 11). Diskreetin Fourier-muunnoksen vaativuus kasvaa näytepisteiden määrän neliönä, joka tekee siitä todella vaivalloista. Fourier-muunnoksia voidaan laskea kuitenkin nopeamminkin niin sanotulla nopealla Fourier-muunnoksella (FFT, Fast Fourier Transform). FFT-muunnos perustuu Cooley-Tukey -algoritmiin, jossa annettu tehtävä muutetaan pienemmiksi osiksi, jolloin ratkaisu on huomattavasti helpompaa. Esimerkiksi 1024 näytteen laskeminen kestää diskreetillä Fourier-muunnoksella 100 kertaa kauemmin, kuin FFT-muunnoksella. (Huttunen 2014, 33.)

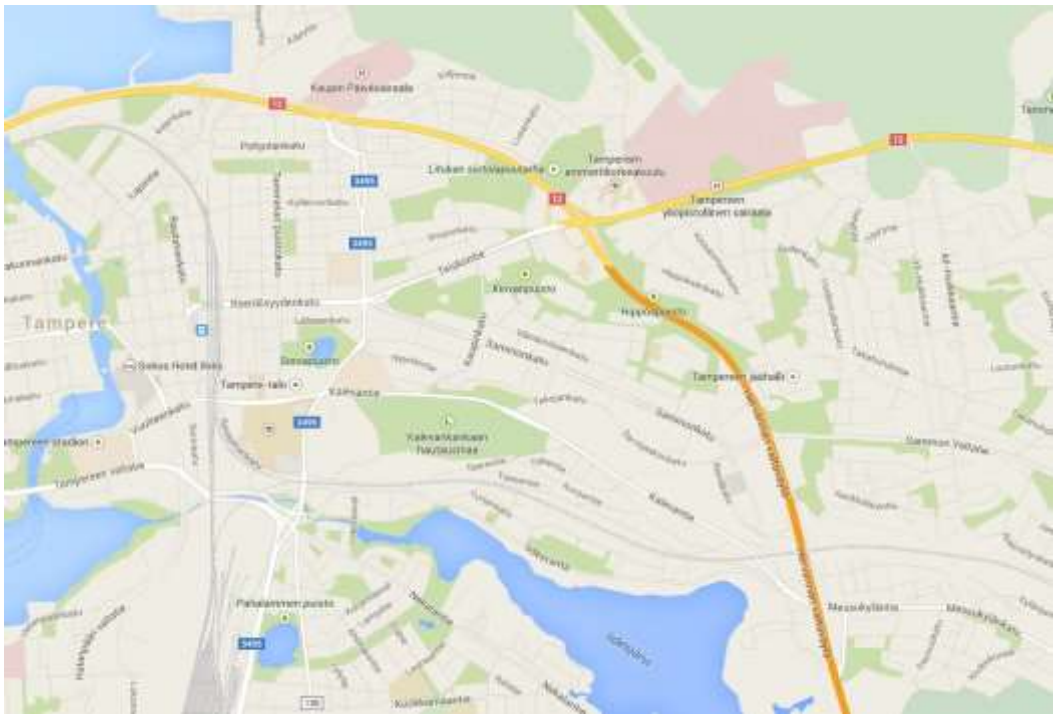


KUVA 11. Signaali kostuu sinisignaalien summasta (Kuva: Lucas Barbosa, 2013)

2.5 WebGL

WebGL (Web-based Graphics Library) tarkoittaa web-pohjaista grafiikkakirjastoa. Se on JavaScript-ohjelmointirajapinta, mitä käytetään interaktiivisen 3D- ja 2D-grafiikan mallintamiseen nettiselaimelle ilman ylimääräisiä lisäosia tai lisälaitteita. Grafiikka mallinnetaan HTML5 canvas-elementille käyttämällä tietokoneen näytönohjainta. Khronos Group niminen työryhmä julkaisi WebGL:n Firefox- ja Chrome-selaimille vuonna 2011 ja se vastaa edelleen WebGL:n päivityksistä. WebGL perustuu OpenGL ES 2.0 versioon, joka julkaistiin vuonna 2003, ja sillä mallinnettiin grafiikkaa muun muassa pelikonsoleille. (Zakas 2012, 571.)

OpenGL (Open Graphics Library) on laitteistoriippumaton ohjelmointirajapinta, mitä käytetään tuottamaan interaktiivista tietokonegrafiikkaa. Se on pohjana WebGL:lle, mutta sen laajuus on todella valtava, joten sitä ei käsitellä sen tarkemmin tässä kappaleessa. WebGL:llä voidaan luoda visuaalisia nettisivuja ja pelejä. Sillä voidaan myös luoda kuvankäsittelyyn tarkoitettuja sivuja, jolloin käyttäjän ei tarvitse ladata tietokoneelleen erikseen erillistä kuvankäsittelyä vaativia ohjelmia. Ehkä tunnetuimpana sivuna on Googlen omistama Google Maps, joka käyttää MapsGL-rajapintaa, mikä perustuu WebGL:ään (kuva 12). (OpenGL 2015.)

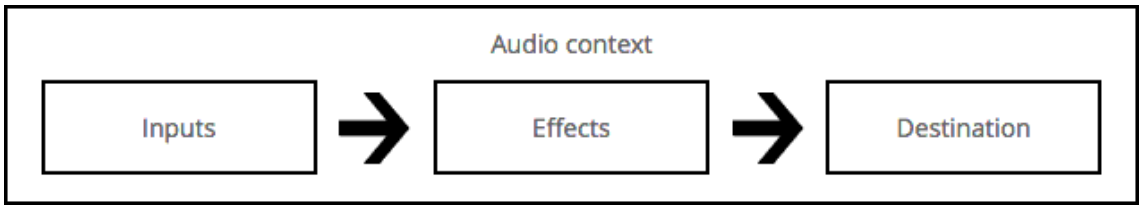


KUVA 12. Google Maps (Google Maps)

3 ÄÄNEN KÄSITTELY

Tässä luvussa esitellään ääninäytteen analysointia Web Audio API:lla. Esitellään miten audiokonteksti muodostetaan JavaScriptin avulla ja mitä audiokonteksti sisältää. Selvitetään miten näytteenottotaajuus vaikuttaa äänen analysointiin ja esitetään miltä näyttää taajuustason esitys nettiselaimella.

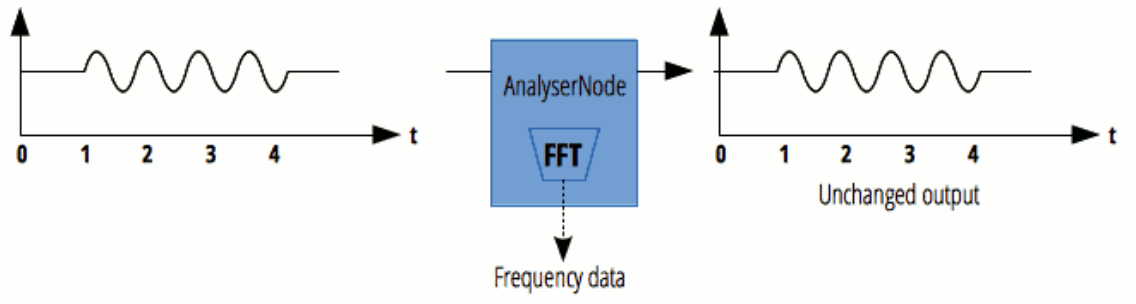
Web Audio API on JavaScriptille saatava korkeantason ohjelmointirajapinta, joka pystyy prosessoimaan ja syntetisoimaan ääntä, sekä luomaan dynaamisia ääniefektejä. Sillä pystytään myös suodattamaan ja vahvistamaan erilaisia ääniä, sekä sillä voidaan tehdä reaaliaikaisia analyyseja äänelle. API:ssa kaikki operaatiot, mitä äänelle halutaan tehdä, tapahtuvat audiokontekstin sisällä (kuva 13). (Web Audio API 2015.)



KUVA 13. Audiokontekstissa esiintyvät solmut (Basic concepts behind Web Audio API)

Audiokonteksti muodostuu kuvan 13 mukaisesti. Ensimmäisenä audiokontekstin sisälle määritellään lähde, mikä kertoo mistä haluttu ääni tulee. Ääni voidaan tehdä API:lla tai käyttämällä HTML5-audio-elementtiä, joka käyttää valmista ääninäytettä. Äänilähteenä voi myös toimia mikrofoni, joka on kytketty tietokoneeseen. Kun äänilähde on luotu, sille voidaan luoda erilaisia efektejä esimerkiksi kaikuefekti tai ääntä voidaan suodattaa. Tämän jälkeen valitaan äänelle sopiva ulostulokohde esimerkiksi tietokoneen kaiuttimet. Kaikki osat yhdistetään toisiinsa ja näitä osia kutsutaan nodeiksi eli solmuiksi. Nodeja voi olla lähes rajaton määrä esimerkiksi äänilähteitä voi olla useampia, samoin kuin efektejä tai suodattimia. (Web Audio API 2015.)

API:ssa on olemassa analysaattorisolmu (AnalyserNode), jonka avulla äänelle pystytään tekemään FFT-muunnos (kuva 14). Näin äänestä voidaan tallentaa haluttu tieto taulukkoon, ilman että signaalia muutetaan mitenkään. Tallennettua dataa käytetään grafiikan luonnissa, jonka avulla voidaan esittää signaalin taajuuden vaihtelut graafisesti. Web Audio API:n käyttö JavaScriptillä on kuvan 15 mukainen. (Web Audio API 2015.)



KUVA 14. AnalyserNoden käyttöä (Basic concepts behind Web Audio API).

```
<audio id="player" controls>
    <source src="toccata.mp3" type="audio/mp3" preload="auto">
     Your browser does not support the audio tag!
</audio>
function setupWebAudio() {
    // Get our <audio> element
    var audio = document.getElementById('player');
    // Create a new audio context (that allows us to do all the Web Audio stuff)
    var audioContext = new (window.AudioContext || window.webkitAudioContext)();
    // Create a new analyser (Global variable)
    analyser = audioContext.createAnalyser();
    // fftsize
    analyser.fftSize = 2048;
    // Create a new audio source from the <audio> element
    var source = audioContext.createMediaElementSource(audio);
    // Connect up the output from the audio source to the input of the analyser
    source.connect(analyser);
    // Connect up the audio output of the analyser to the audioContext destination i.e. the speakers
    analyser.connect(audioContext.destination);
    // Get the <audio> element started
    audio.play();
    console.log(audioContext.samplerate);
}
```

KUVA 15. Audiokontekstin tuottava JavaScript-aliohjelma

Kuvan 15 yläosassa on HTML5:llä luotu audio-elementti, jonka tunnusnimeksi on annettu player. Aloitustunnisteessa on myös sallittu ohjainten (controls) käyttö, joilla voidaan pysäyttää äänen toisto ja säätää äänenvoimakkuutta. Audio-elementin sisälle on laitettu source-elementti, jonka avulla määritellään, mikä äänitiedosto toistetaan. Tunnisteiden sisään on annettu attribuutteina tyyppi, millaisesta tiedostopäätteestä on kyse ja preload-tieto. Preload-attribuutilla selain lataa tiedoston valmiiksi, kun sivu avataan. Tämä on hyvä tapa isokokoisilla tiedostoilla, mutta ei välttämätön. Source-tunnisteiden jälkeen on kirjoitettu kuvassa näkyvä teksti, joka tulee näkyviin vanhoilla selaimilla ja ilmoittaa käyttäjälle, että selain ei tue audio-elementtiä.

Kuvan 15 alaosassa luodaan funktiotyyppinen aliohjelma JavaScriptillä, jonka avulla päästään käsiksi audio-elementtiin. Aluksi tehdään muuttuja, joka viittaa luotuun audio-elementtiin tunnusnimen kautta. Tämän jälkeen tehdään audiokonteksti, jotta päästään käsiksi Web Audio API:n käyttöön. Seuraavilla riveillä luodaan analysaattorisolmu ja tämän jälkeen asetetaan äänilähteeksi tehty audiomuuttuja, joka viittaa äänitiedostoon. Kaikki solmut yhdistetään keskenään, kuten kuvassa 13 esiteltiin. Funktion lopussa käynnistetään audio-elementti, jolloin ääni alkaa kuulumaan.

Analysaattorisolmussa annetaan myös FFT-ikkunan koolle arvo (fftsize), jonka avulla pystytään määrittämään kuinka tarkasti haluttuja taajuuksia kuvataan. Koska signaali sisältää suuren määrän eri taajuuksia, joudutaan FFT-ikkunan koolla rajaamaan aluetta sopivaan taajuustason esitysmuotoon. Suurella FFT-ikkunan koolla pystytään tarkemmin kuvaamaan taajuuksia, mutta tämä vaatii enemmän laskentatehoa, kuin pienemmällä ikkunan koolla (FFT Size 2012). Taajuustaso jaetaan sopivan kokoisiin taajuuskaistoihin, joita voidaan kutsua taajuuskomponenteiksi (frequency bin). Seuraavalla kaavalla määritellään taajuuskomponenttien lukumäärä

$$N = \frac{FFT}{2} \tag{4}$$

missä

$$N = \text{taajuuskomponenttien lukumäärä}$$

$$FFT = \text{ ikkunan koko}$$

Ja tästä pystytään määrittelemään yhden komponentin leveys Hertzeinä eli resoluutio

$$F_R = \frac{f_{max}}{N} \tag{5}$$

missä

$$F_R = \text{taajuusresoluutio}$$

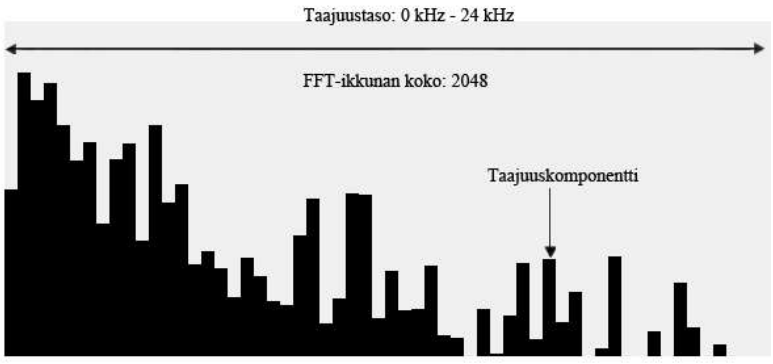
Sijoittamalla kaavaan (4) kuvassa 15 asetettu FFT-koon arvo komponenttien lukumääräksi tulee

$$N = \frac{FFT}{2} = \frac{2048}{2} = 1024$$

Web Audio API:ssa on oletusasetuksen näytteenottotaajuus f_s asetettu arvoon 48 kHz ja kaavan (2) mukaisesti saadaan maksimitaajuudeksi 24 kHz. Kaavalla (5) saadaan laskettua resoluutio eli erottelukyky taajuudelle

$$F_R = \frac{f_{max}}{N} = \frac{24\,000}{1024} \approx 23 \text{ Hz}$$

Näin saadaan erottelukyvyksi 23 Hertziä, joka kuvaa kuinka hyvin taajuuskomponentit voidaan erottaa toisistaan taajuustasossa (kuva 16). Web Audio API:ssa FFT-ikkunan koko voi maksimissaan olla 32768. Koska audioanalysaattori ei muuta signaalin laatua mitenkään, vaan signaalia ainoastaan analysoidaan, on ikkunan koko 2048 riittävän hyvä kuvaamaan taajuuksia. Näytteenottotaajuus saisi olla myös pienempi, mutta sen arvoa ei API:sta pysty muuttamaan. (The Frequency Domain 2011.)



KUVA 16. Web Audio API:lla tehty taajuustason esitys

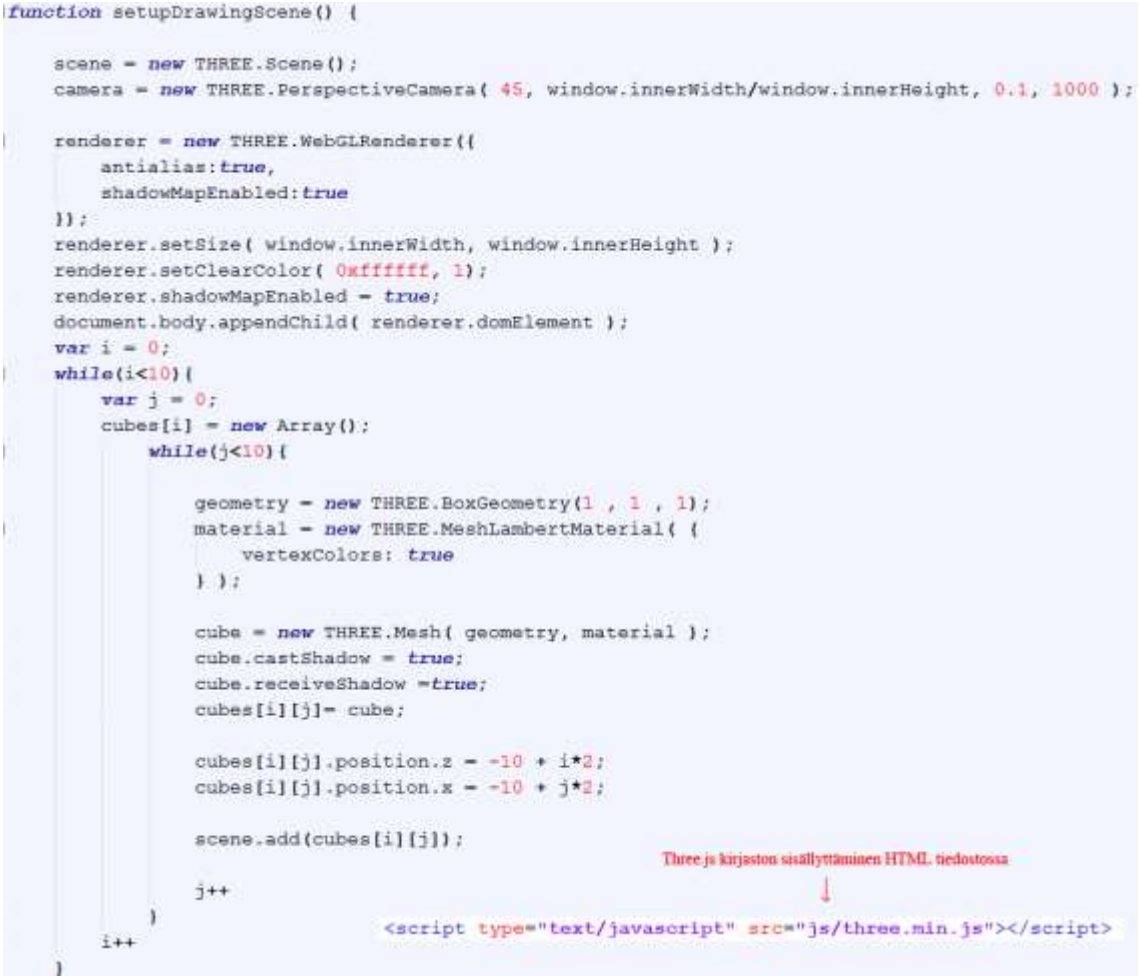
4 GRAFIIKAN LUONTI

Tässä luvussa esitellään miten äänen dataa käytetään grafiikan mallinnuksessa. Lisäksi tarkastellaan miltä lopullinen teos näyttää kokonaisuudessaan ja selvitetään kaikki siihen liittyvät vaiheet. Jatkokehitys osiossa tutkitaan miten teosta voidaan lähteä kehittämään eteenpäin ja mitä asioita valmiiseen teokseen olisi pitänyt tulla.

4.1 Three.js ja pohjan luonti

Three.js on WebGL-kirjasto, joka helpottaa grafiikan mallinnusta merkittävästi. Ilman apukirjastoa yhden kuution luomisesta voi hyvinkin muodostua tuhat riviä pitkä koodi. Lisäksi kirjaston käyttämisessä ei tarvitse tuntea WebGL:n ohjelmointia perusteellisesti, vaan riittää että tietää pääasiat. Three.js on MIT-lisenssin alaisuudessa, joka antaa käyttäjälle oikeudet vapaasti muokata, kopioida ja käyttää teosta omassa projektissa sillä ehdolla, että lisenssin teksti säilyy lähdekoodissa. Three.js-kirjastosta on olemassa oma dokumentointi, sekä paljon käyttäjien tekemiä esimerkkejä. (Three.js 2015.)

Jotta apukirjastoa voidaan käyttää, tulee se ladata koneelle ja ottaa käyttöön (kuva 17, oikealla alhaalla). Kuvassa 17 on esitelty miltä näyttää tässä työssä pohjan rakentamiseen käytetty JavaScript-koodi. Kolme ensimmäistä ja tärkeintä asiaa, mitä pitää muodostaa, ovat scene, camera ja renderer. Scene on näyttämö, jossa esiintyy mallinnettu kuvio. Camera eli kamera kuvaa mallinnettua kuviota. Sille annetaan parametreinä näkökenttä, kuvasuhde lähellä olevien kuvioiden piirtoetäisyys ja kaukana olevien kuvioiden piirtoetäisyys itseensä nähden. Viimeisenä muodostetaan renderer-elementti, joka tuottaa HTML5-canvas-elementin, mihin scene ja kamera kiinnitetään. Renderer-elementille annetaan useita parametrejä, jotka määrittelevät kuinka suuri osa selaimesta käytetään kuvion mallintamiseen, mikä on taustaväri ja käytetäänkö varjostuksia.



```
function setupDrawingScene() {

    scene = new THREE.Scene();
    camera = new THREE.PerspectiveCamera( 45, window.innerWidth/window.innerHeight, 0.1, 1000 );

    renderer = new THREE.WebGLRenderer({
        antialias:true,
        shadowMapEnabled:true
    });
    renderer.setSize( window.innerWidth, window.innerHeight );
    renderer.setClearColor( 0xffffff, 1);
    renderer.shadowMapEnabled = true;
    document.body.appendChild( renderer.domElement );
    var i = 0;
    while(i<10){
        var j = 0;
        cubes[i] = new Array();
            while(j<10){

                geometry = new THREE.BoxGeometry(1 , 1 , 1);
                material = new THREE.MeshLambertMaterial( {
                    vertexColors: true
                } );

                cube = new THREE.Mesh( geometry, material );
                cube.castShadow = true;
                cube.receiveShadow =true;
                cubes[i][j]= cube;

                cubes[i][j].position.z = -10 + i*2;
                cubes[i][j].position.x = -10 + j*2;

                scene.add(cubes[i][j]);

                j++
            }
        i++
    }
```

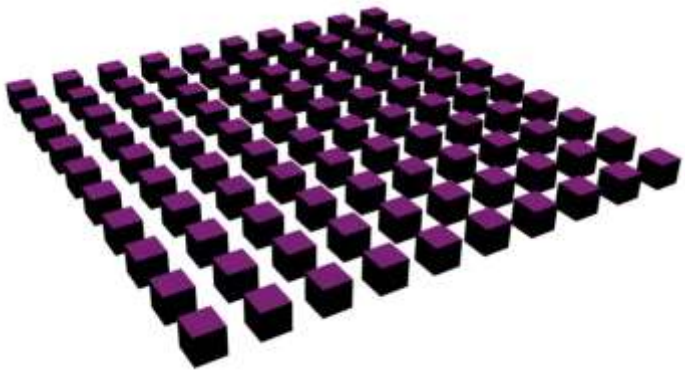
Three.js kirjaston sisällyttäminen HTML tiedostossa

```
<script type="text/javascript" src="js/three.min.js"></script>
```

KUVA 17. JavaScript-aliohjelma graafisen pohjan luomiselle

Tässä vaiheessa sivulla ei näy vielä mitään, koska kuvioita ei ole luotu. Kuviot tallennetaan kaksiulotteiseen taulukkoon, koska halutaan muodostaa 10x10-ruudukko kuutioista. Kuutiot muodostetaan kahdella sisäkkäisellä while-silmukalla ja sijoitetaan cube-nimiseeen taulukkoon. Kaksiulotteisen taulukon muodostaminen JavaScriptillä tapahtuu siten, että ensimmäisen taulukon sisälle muodostetaan toinen taulukko. Ensimmäinen taulukko on määritelty aliohjelman ulkopuolella globaalina muuttujana, jotta sitä voidaan lukea muista aliohjelmista. Yksittäisen kuution muodostaminen tapahtuu käyttämällä BoxGeometry-oliota ja tälle annetaan parametreinä leveys, pituus ja syvyys. Kuutiolle täytyy tämän jälkeen antaa materiaali mistä se rakennetaan luomalla Material–olio, joka määrittää kuution ulkonäön, värin, varjostuksen ja tekstuurin. Kuutio ja materiaali yhdistetään yhdeksi kokonaisuudeksi Mesh-oliolla.

Mesh-oliolle välitetään tiedot kuution leveydestä, pituudesta ja syvyydestä, minkä jälkeen kuutiolle asetetaan materiaali. Näin on muodostettu valmis kuutio, joka sijoitetaan cube-taulukkoon. Taulukossa olevalle kuutiolle annetaan lopuksi tieto mihin kohtaan näyttämöä se sijoitetaan. Aliohjelman lopuksi lisätään taulukko kaikkine tietoineen näyttämölle. Tässä vaiheessa ei olla vielä kutsuttu kuutioita mallintavaa ohjelmaa, joten sivulla ei vielä ole mitään näytettävää. Taulukossa on kuitenkin tieto kuutioista ja se voidaan esittää kuvalla 18, jotta saadaan käsitys mitä ollaan tehty.

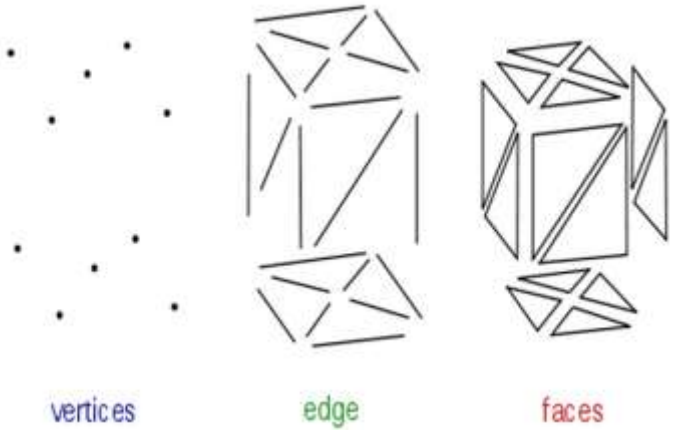


KUVA 18. 10x10-taulukko, joka sisältää kuutioiden tiedot

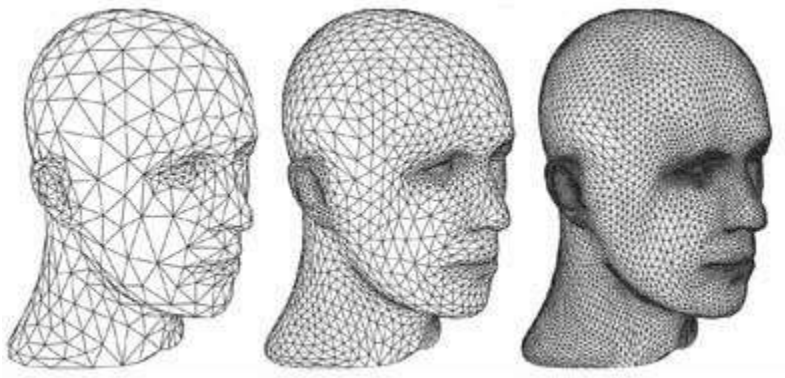
4.2 Mallinnus

Mesh-olio sisältää kaikki kuution oleelliset tiedot ja tätä oliota voidaan kutsua myös monikulmioverkoksi (Polygon Mesh). Monikulmioverkot ovat hyvin laaja osa-alue tietokonegrafiikassa ja geometrisessä mallinnuksessa, joten sen oppimiseen ja opettamiseen tarvitaan hyvät matemaattiset osaamiset ja kolmiulotteinen hahmotuskyky. Työtä varten joudutaan kuitenkin hieman perehtymään tähän osa-alueeseen, jotta saadaan hahmotettua se miten mallinnus tapahtuu. (About Polygon Meshes 2011.)

Mesh-olion sisällä kuutio näyttää kuvan 19 kaltaiselta. Kuutio rakentuu kärkipisteistä, särmistä ja tahkoista. Särmät yhdistyvät kärkipisteisiin ja näiden väliin jäävä alue muodostaa kolmion, jota kutsutaan tahkoksi. Kaikki tapahtuu koordinaatistossa yhdistelemällä pisteistä toisiinsa, joista syntyy pintoja (kuva 19).

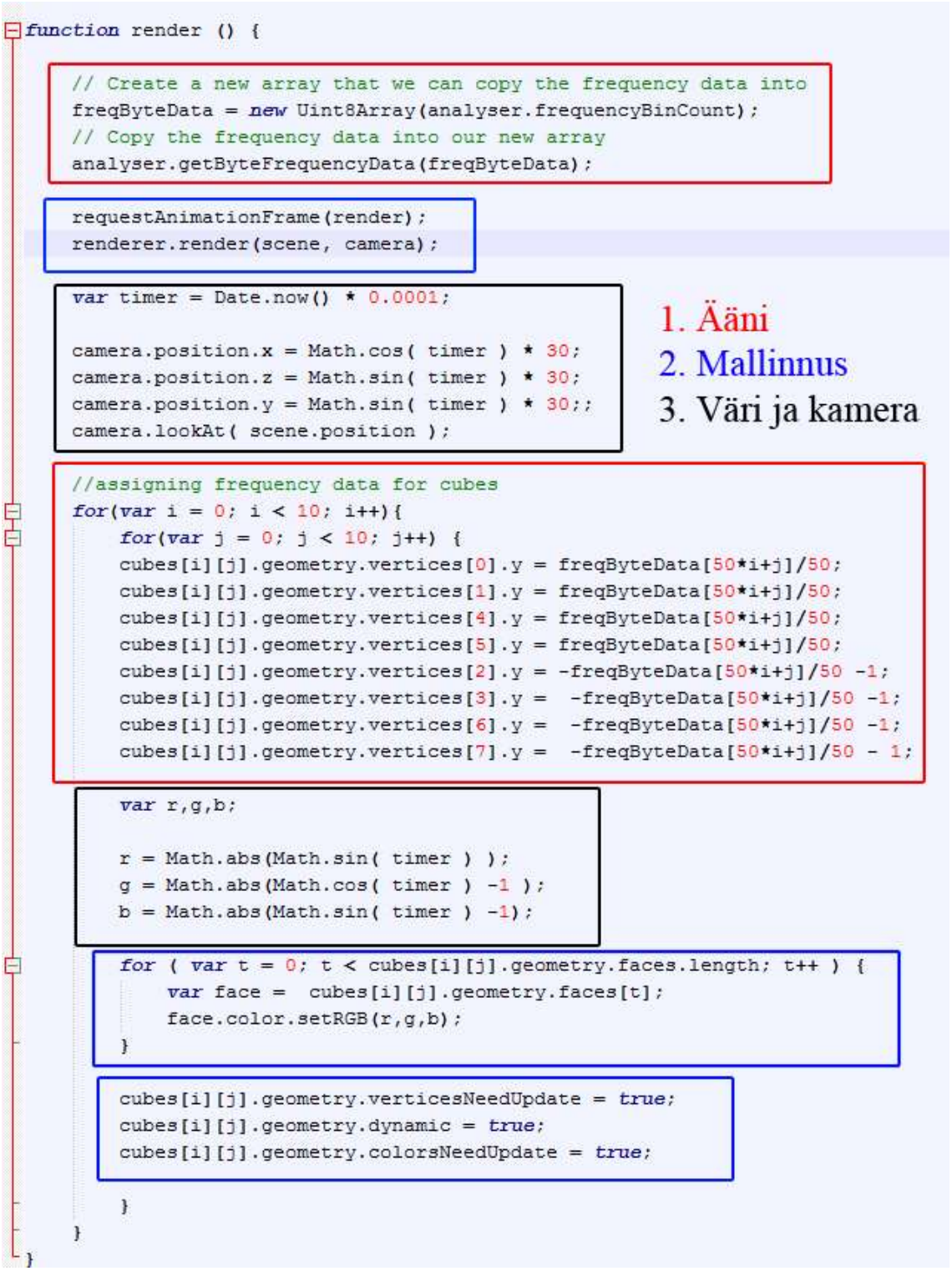


KUVA 19. Kuution rakenne kärkipisteinä, särminä ja tahkoina (Kuva: David Dorfman, 2011, muokattu)



KUVA 20. Monikulmioilla luotu ihmisen kasvot (PC Perspective)

Kun tiedetään miten kuutiot rakentuvat olioiden sisällä, tutkitaan mallinnuksen tuottavaa aliohjelmaa. Kuvassa 21 on esitetty koko mallinnuksen suorittava aliohjelma, johon on värikoodeilla merkattu mallinnukseen liittyvät eri osiot.




```
function render () {
    // Create a new array that we can copy the frequency data into
    freqByteData = new Uint8Array(analyser.frequencyBinCount);
    // Copy the frequency data into our new array
    analyser.getByteFrequencyData(freqByteData);

    requestAnimationFrame(render);
    renderer.render(scene, camera);

    var timer = Date.now() * 0.0001;

    camera.position.x = Math.cos( timer ) * 30;
    camera.position.z = Math.sin( timer ) * 30;
    camera.position.y = Math.sin( timer ) * 30;;
    camera.lookAt( scene.position );

    //assigning frequency data for cubes
    for(var i = 0; i < 10; i++){
        for(var j = 0; j < 10; j++) {
        cubes[i][j].geometry.vertices[0].y = freqByteData[50*i+j]/50;
        cubes[i][j].geometry.vertices[1].y = freqByteData[50*i+j]/50;
        cubes[i][j].geometry.vertices[4].y = freqByteData[50*i+j]/50;
        cubes[i][j].geometry.vertices[5].y = freqByteData[50*i+j]/50;
        cubes[i][j].geometry.vertices[2].y = -freqByteData[50*i+j]/50 -1;
        cubes[i][j].geometry.vertices[3].y =  -freqByteData[50*i+j]/50 -1;
        cubes[i][j].geometry.vertices[6].y =  -freqByteData[50*i+j]/50 -1;
        cubes[i][j].geometry.vertices[7].y =  -freqByteData[50*i+j]/50 - 1;

        var r,g,b;

        r = Math.abs(Math.sin( timer ) );
        g = Math.abs(Math.cos( timer ) -1 );
        b = Math.abs(Math.sin( timer ) -1);

        for ( var t = 0; t < cubes[i][j].geometry.faces.length; t++ ) {
            var face =  cubes[i][j].geometry.faces[t];
            face.color.setRGB(r,g,b);
        }

        cubes[i][j].geometry.verticesNeedUpdate = true;
        cubes[i][j].geometry.dynamic = true;
        cubes[i][j].geometry.colorsNeedUpdate = true;

        }
    }
}
```

KUVA 21. Mallinnuksen tuottava aliohjelma

Alussa ensimmäinen punaisella merkattu kuutio liittyy aikaisempaan kuvaan 14, jossa esiteltiin analysaattorisolmun käyttöä ja äänen tallentamista taulukkoon. Kyseinen taulukko tehdään ja siihen tallennetaan kaikki Fourier-muunnoksella saadut taajuuden arvot. Taulukointi tehdään mallinnusaliohjelmassa, koska seuraavassa sinisessä laatikossa oleva requestAnimationFrame on callback-kutsu mallinnusaliohjelmaan. Yksinkertaistettuna tämä tarkoittaa sitä, että taulukossa olevat arvot päivitetään oikealla hetkellä mallintajan kannalta eli ääni ja kuva ovat samassa tahdissa. Tällöin selain pystyy synkronisesti piirtämään oikeaa kuvaa oikeilla arvoilla ja prosessorin käyttöaste pienenee.

Ensimmäisessä mustalla rajatussa laatikossa säädetään kameran liikettä sini- ja kosinifunktioiden avulla. Tällöin saadaan kamera liikkumaan satelliitin kaltaisesti, mikä muodostaa pallomaisen liikeradan. Seuraavassa mustalla rajatussa laatikossa muodostetaan muuttuvat värit punainen, vihreä ja sininen. Näihinkin käytetään sini- ja kosinifunktioita, jotta saadaan värit vaihtumaan tasaisesti eri sävyjen välillä. Graafisen pohjan muodostavan aliohjelman (kuva 17) lopussa on lisätään valot, jotta saadaan kuutioiden väri näkymään kuten kuvassa 18. JavaScriptillä valojen lisääminen tapahtuu kuvan 22 mukaisesti.

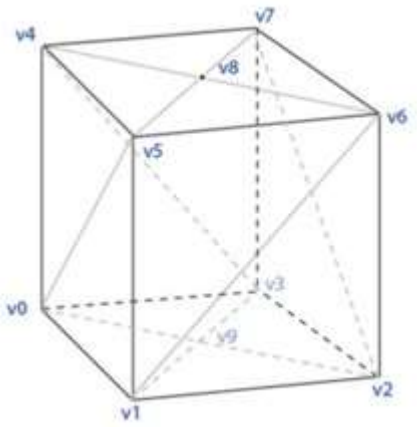
```
// Create lights
  // white spotlight shining from the side, casting shadow
var spotLight = new THREE.SpotLight( 0xffffff );
spotLight.position.set( 100, 1000, 100 );
spotLight.castShadow = true;
spotLight.shadowMapWidth = 1024;
spotLight.shadowMapHeight = 1024;
spotLight.shadowCameraNear = 500;
spotLight.shadowCameraFar = 4000;
spotLight.shadowCameraFov = 30;
scene.add( spotLight );

//same spotlight as above but under the boxes
var spotLight2 = new THREE.SpotLight( 0xffffff );
spotLight2 .position.set( -100, -1000, -100 );
spotLight2 .castShadow = true;
spotLight2 .shadowMapWidth = 1024;
spotLight2 .shadowMapHeight = 1024;
spotLight2 .shadowCameraNear = 500;
spotLight2 .shadowCameraFar = 4000;
spotLight2 .shadowCameraFov = 30;
scene.add( spotLight2 );
```

KUVA 22. Valojen sijoitus kuutioiden ylä- ja alapuolelle

Keskimmäisessä punaisella rajatussa alueessa (kuva 21) muodostetaan kahdella sisäkkäisellä for-silmukalla kuutioiden koon muuttaminen taajuuden mukaisesti. Tämä tapahtuu antamalla uudet koordinaatit kärkipisteille, jolloin kuution koko muuttuu ja se mallinnetaan uudestaan uusilla koordinaateilla. Kärkien liikuttamiseen käytettiin apuna kuvassa 23 olevaa kuviota. Kuvasta huomataan, että liikuttamalla pisteitä v0,v1,v4 ja v5 samanaikaisesti, kuutio pienenee tai suurenee vasemmalle. Samalla periaatteella tapahtuu kuution koon muuttaminen for-silmukassa. Kuutioita kasvatetaan molempiin suuntiin for-silmukassa, ylös ja alas, samalla taajuuden arvolla. Näin saadaan näyttävämpi vaikutelma.

Vertex List

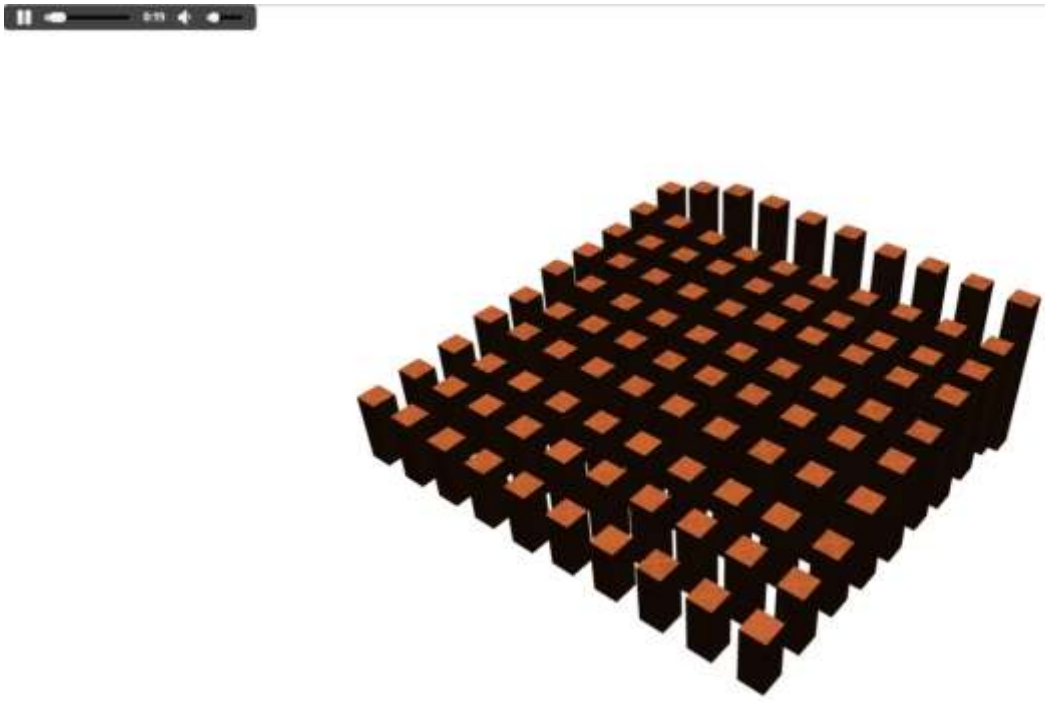
v0	0,0,0	v1 v5 v4 v3 v9
v1	1,0,0	v2 v6 v5 v0 v9
v2	1,1,0	v3 v7 v6 v1 v9
v3	0,1,0	v2 v6 v7 v4 v9
v4	0,0,1	v5 v0 v3 v7 v8
v5	1,0,1	v6 v1 v0 v4 v8
v6	1,1,1	v7 v2 v1 v5 v8
v7	0,1,1	v4 v3 v2 v6 v8
v8	.5,.5,1	v4 v5 v6 v7
v9	.5,.5,0	v0 v1 v2 v3



KUVA 23. Kärkipisteiden koordinaattien hahmottaminen (Kuva: David Dorfman, 2011)

Aliohjelman lopussa toinen sinisellä rajattu alue sijoittaa muodostetut värit kuution tahkoille, jolloin kuution jokainen pinta saa saman värin. Viimeisessä sinisellä rajatussa alueessa asetetaan kärkipisteiden päivitys päälle, jotta jokaisen piirtämisen jälkeen saadaan uudet koordinaatit pisteille. Lisäksi välitetään mallintajalle tieto, että kuutiot ovat dynaamisia, joka viittaa koordinaattien päivittämiseen ja lopuksi sallitaan värien päivittämien.

Kun selaimella avataan index.html tiedosto, joka sisältää teoksen, näyttää se kuvan 24 kaltaiselta. Vasemmassa ylänurkassa nähdään musiikkisoittimen ohjain ja kuvasta huomataan, kuinka kuutioiden koko vaihtelee. Ääninäyteen matalat taajuudet näkyvät kuvassa suurikokoisimpina kuutioina, koska ääninäyte sisältää enemmän matalia taajuuksia.



KUVA 24. Valmis audiovisualisointi

4.3 Jatkokehitys

Teos on rakennttu niin, että se on täysin muokattavissa. Kuvioiden muotoa ja materiaalia voidaan muokata eri käyttöön sopiviksi. Mallinnetusta kuviosta voidaan esimerkiksi tehdä interaktiivinen, jolloin käyttäjä pystyy muokkaamaan ympäristöä tai liikuttamaan mallinnettuja esineitä. Kameran vapaa liikuttaminen hiiren avulla on myös toteutettavissa oleva vaihtoehto.

Teosta on muokattu mallintamaan avaruudessa esiintyviä planeettoja kuvan 25 mukaisesti. Kuutiot korvattiin palloilla, jotka pyörivät oman akselinsa ympäri eri tahdissa. Planeettojen ympärille mallinnettiin 20 000 tähteä, jotka ovat sijoiteltu

satunnaisille paikoille planeettojen ympärille. Teos ei sisällä äänen käsittelyä, mutta se on mahdollista sisällyttää teokseen esimerkiksi tähtien vilkkumisella äänen tahtiin.

KUVA 25. Avaruuden mallinnus

5 POHDINTA

Opinnäytetyössä esitetty graafinen audiovisualisointi onnistui kokonaisuudessaan suunnitelmien mukaisesti. Teokseen oli tarkoitus tehdä HTML5-ohjelma, missä käyttäjä saa itse asettaa haluamansa kappaleen soimaan ja kuutiot liikkuvat tämän kyseisen kappaleen tahdissa. Tämä saatiin toimimaan, mutta Chrome-selaimen päivityksen jälkeen ohjelma lopetti toimintansa. Päivityksen yhteydessä korjattiin useita haavoittuvaisuuksia, joita selaimessa esiintyi. Yksi näistä ongelmista oli Cross-Origin, mikä salli selaimen hakea tietoa muista verkkotunnuksista. Tämä aiheutti sen, että teos lopetti toimintansa ilman palvelimen käyttöä.

Teoksen suunnittelun ja toteutuksen kannalta kuutiot olivat hyvä valinta toteuttaa taajuuden visualisointia. Opinnäytetyössä esitellään hyvin yksinkertaisesti Three.js-kirjaston käyttäminen, mikä antaa hyvät lähtökohdat grafiikan mallintamiseen WebGL:n avulla. WebGL on kuitenkin paljon laajempi kokonaisuus ja työssä tarkasteltiin vain hyvin pientä osa-aluetta. Työn lopussa esitetty avaruuden mallintamien kuvaa laajemmin WebGL:ää, koska siinä muodostetaan planeettojen varjostus paljon monimutkaisemmin. Varjostukseen käytetään GLSL-tekniikkaa (OpenGL Shading Language), joka on korkean tason varjostus kieli. Se muistuttaa syntaksiltaan C-ohjelmointikieltä.

Opinnäytetyössä äänen analysointiin käytetty Web Audio API on hyvin monipuolinen ja tehokas sovellus äänen käsittelyyn. Sen avulla voidaan rakentaa esimerkiksi digitaalinen syntetisaattori. Web Audio API:lla pystytään tuottamaan lähes kaikkia haluttuja ääniefektejä ja sen suodatinominaisuudet ovat erittäin vaikuttavia. Sen avulla pystytään muodostamaan kokonainen yhtye eri instrumenteilla. Soittimet pystytään mallintamaan WebGL:n avulla, jolloin tuloksena on hyvin näyttävän näköinen esitys.

LÄHTEET

About Polygon Meshes. 2011. Autodesk Softimage. Luettu 20.5.2015.
http://softimage.wiki.softimage.com/xsidocs/poly_basic_PolygonMeshes.htm

AMD ATI Radeon HD 2900 XT Review. 2007. PC Perspective. Luettu 25.4.2015.
http://www.pcper.com/

An Introduction to Electronic Measurement. 2011. Agilent. Luettu 15.4.2015.
http://www.agilent.com/labs/features/2011_101_elec.html

An Introduction to WebGL. 2011. Opera Developer Team. Luettu 15.4.2015.
https://dev.opera.com/articles/introduction-to-webgl-part-1/

FFT Size. 2012. AudioSculpt. Luettu 10.4.2015.
http://support.ircam.fr/docs/AudioSculpt/3.0/co/FFT%20Size.html

Google Maps. 2015. Google. Tulostettu 27.4.2015
http://maps.google.fi

Henderson, P. 2015. The Fundamentals of FFT-Based Audio Measurements in SmaartLive. Luettu 18.2.2015
http://www.rationalacoustics.com/files/FFT_Fundamentals.pdf

HTML Introduction. 2015. W3Schools Online Web Tutorials. Luettu 8.5.2015.
http://www.w3schools.com/html/html_intro.asp

HTML(5) Tutorial. 2015. W3Schools Online Web Tutorials. Luettu 8.5.2015.
http://www.w3schools.com/html/html_intro.asp

Huttunen, H. 2014. Signaalinkäsittelyn perusteet.Tampereen teknillinen yliopisto. Signaalinkäsittelyn laitos. Luettu 8.5.2015.
http://www.cs.tut.fi/kurssit/SGN-11000/SGN-11000.pdf

Minelab Terminology. Minelab. Luettu 20.4.2015.
http://www.minelab.com/emea/consumer/knowledge-base/terminology

OpenGL. 2015. The Khronos Group. Luettu 8.5.2015.
https://www.opengl.org/

Reah, I. 2013. Real-time analysis of streaming audio data with Web Audio API. Luettu 10.3.2015.
http://ianreah.com/2013/02/28/Real-time-analysis-of-streaming-audio-data-with-Web-Audio-API.html

requestAnimationFrame. 2014. The CreativeJS Team. Luettu 20.4.2015.
http://creativejs.com/resources/requestanimationframe/

Signal Path Series. 2014. Mixrevu. Luettu 15.4.2015.
http://www.mixrevu.com/?p=article&id=17

Smith, C. 2006. On Vertex-Vertex Systems and Their Use in Geometric and Biological Modelling. The University of Calgary. Luettu 10.4.2015. http://algorithmicbotany.org/papers/smithco.dis2006.pdf

Smus, B. 2013. Web Audio API. Luettu 28.4.2015. http://chimera.labs.oreilly.com/books/1234000001552

The Frequency Domain. 2011. Music and Computers. Luettu 10.5.2015 http://music.columbia.edu/cmc/musicandcomputers/chapter3/03_04.php

Three.js. 2015. Github.Päivitetty 21.4.2015. Luettu 28.4.2015. http://threejs.org/

Web Audio API. 2015. Mozilla Developer Network. Päivitetty 17.3.2015. Luettu 24.4.2015. https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API

WebGL-Audio-Visualization. 2014. Github. Luettu 10.3.2015. https://github.com/macobo/WebGL-Audio-Visualization

What is digital audio. 2014. Floss Manuals. Luettu 13.4.2015. http://en.flossmanuals.net/pure-data/introduction/what-is-digital-audio/

W3C Editor's Draft . Web Audio API. Päivitetty 3.4.2015. Luettu 23.4.2015. http://webaudio.github.io/web-audio-api/

Zakas, N. 2012. JavaScript for Web Developers. 3.painos. Indianapolis: John Wiley & Sons , Inc.