Otto Piispanen

WEBGL-SOVELLUSKEHITYS

Peliohjelmointi Three.js-apukirjaston avulla

Opinnäytetyö
Tietojenkäsittely

Huhtikuu 2013

MIKKELIN AMMATTIKORKEAKOULU
Mikkeli University of Applied Sciences

KUVAILULEHTI

MIKKELIN AMMATTIKORKEAKOULU Mikkeli University of Applied Sciences	**Opinnäytetyön päivämäärä** **13.5.2013**
Tekijä(t) Otto Piispanen	**Koulutusohjelma ja suuntautuminen** **Tietojenkäsittelyn koulutusohjelma**

Nimeke

WebGL-sovelluskehitys
Peliohjelmointi Three.js-apukirjaston avulla

Tiivistelmä

3D-grafiikka ja virtuaalinen todellisuus ovat olleet yhdessä yksi eniten tietokoneiden kehitystä eteenpäin vievistä voimista. 3D-grafiikan piirtäminen on ollu 1990-luvun alkuun asti hyvin hidasta, koska eri laitteille on täytynyt kirjoittaa omat grafiikkakirjastot tyhjästä. Tätä helpottamaan luotiin grafiikkarajapinta, joka olisi grafiikan piirtämisen standardi ja joka toimisi kaikilla laitteilla. OpenGL on tällainen grafiikkarajapinta, ja se onkin ollut laajassa käytössä tähän päivään asti. OpenGL vaatii kuitenkin ohjelmien uudelleenohjelmointia eri käyttöjärjestelmille, vaikka ne käyttäisivätkin samaa grafiikkarajapintaa.

WebGL on OpenGL-grafiikkarajapinta, joka on sulautettu Internet-selaimeen. WebGL vähentää entisestään ohjelmoinnin käännöstyötä eri alustoille, sillä WebGL-rajapintaa tukevia selaimia on kaikilla käyttöjärjestelmillä. Suosituimmista selaimista vain Internet Explorer ei tue WebGL-rajapintaa. Syynä voidaan pitää OpenGL-rajapinnan kilpailua Microsoftin DirectX-rajapinnan kanssa. Koska teknologia on uusi, esittelen sen historian ja perusteet sen toiminnasta tässä opinnäytetyössä.

Keskityn opinnäytetyössä peliohjelmointiin, koska se on monipuolisin tapa esitellä ja opiskella uutta teknologiaa. Grafiikan ohjelmointi voi olla erittäin hidasta. Tämän vuoksi on hyvä harkita apukirjaston käyttöä nopeuttamaan kehitysprosessia. Tässä opinnäytetyössä esittelen Three.js-apukirjaston perustoiminnot askel askeleelta. Tämän lisäksi kerron, kuinka oma pelinkehitysprosessini eteni ja mitä tuloksia sain aikaan.

Asiasanat (avainsanat)

WebGL, OpenGL, Three.js

Sivumäärä	**Kieli**	**URN**
43	Suomi	

Huomautus (huomautukset liitteistä)

Ohjaavan opettajan nimi	**Opinnäytetyön toimeksiantaja**
Jukka Selin	

DESCRIPTION

MIKKELIN AMMATTIKORKEAKOULU
Mikkeli University of Applied Sciences

Date of the bachelor's thesis

13 May 2013

Author(s)
Otto Piispanen

Degree programme and option
Business information technology

Name of the bachelor's thesis

Application development with WebGL
Game programming with Three.js

Abstract

The purpose of this thesis was to introduce a new technology called WebGL and to show how it could be used in software development. WebGL is based on the OpenGL graphics library. OpenGL has been the leading interface for multi-device 3D-applications since 1992. OpenGL still requires the programs to be reprogrammed to work on different operating systems. An application using WebGL will work on every operating system and this means that no reprogramming is required on different platforms.

This thesis focused on game programming with the WebGL application programming interface as it was the most versatile way to introduce and to learn a new technology. To make the application development faster, there were many Javascript libraries that used the WebGL API. From all of the options I chose the Three.js library because of its all-around usage and simplicity. A basic Three.js scene was created in this thesis.

The development process resulted in several game engines for different types of games. It appeared that the development could be very time consuming when the Three.js library was used. Developers should therefore consider other libraries dedicated to game engines if the goal is to make game development faster. Three.js is suitable for all kinds of applications, but requires more work on the functionalities.

Subject headings, (keywords)

WebGL, OpenGL, Three.js

Pages	**Language**	**URN**
43	Finnish	

Remarks, notes on appendices

Tutor	**Bachelor's thesis assigned by**
Jukka Selin	

SISÄLTÖ

1 JOHDANTO

Kuvitteellisen kolmiulotteisen maailman ja oman maailmamme kuvaaminen virtuaalisesti on ollut pitkään yksi IT-alan tutkituimpia ja kehitetyimpiä alueita. Tähän mennessä erilaiset tekniikat ovat olleet hyvin laiteriippuvaisia, vaikkakin esimerkiksi OpenGL-rajapinnan piirtotyökaluja hyödynnetään monissa laitteissa. Jokaiselle laitteelle on kuitenkin pitänyt kääntää sovelluksesta oma versionsa. Nyt OpenGL on tuotu Internetin maailmaan WebGL-tekniikan myötä. WebGL mahdollistaa kolmiuloitteisen maailman tehokkaan mallintamisen ilman selaimeen asennettavia lisäosia tavallisilla nettisivuilla. OpenGL-sovellukset vaativat aina käännöstyötä riippuen mille alustalle sovellusta tai peliä luodaan. WebGL-rajapinnan kanssa tätä ei tarvitse miettiä, koska sama sovellus toimii kaikilla käyttöjärjestelmillä kunhan käytössä on WebGL-rajapintaa tukeva selain.

Tämän opinnäytetyön tutkimusongelmana on WebGL-rajapinnan hyödyntäminen sovelluskehityksessä. Sovelluskehityksessä keskitytään peliohjelmointiin, koska se on monipuolinen lähtökohta tekniikan esittelyyn. Peliohjelmointi vaatii hyvin laajaa osaamista tietojenkäsittelystä, jota voidaan hyödyntää hyötysovelluksissa. Apuna sovelluskehityksessä voidaan käyttää useita erilaisia apukirjastoja, mutta tässä opinnäytetyössä käydään läpi sovelluskehitys Three.js-apukirjaston avulla. Opinnäytetyön toinen tarkoitus on olla yksi harvoista oppaista Three.js-apukirjaston käytön aloittamiseen. Opinnäytetyössä pyritään ohjeistamaan erilaisissa yleisissä ongelmatapauksissa, joihin minä ja monet muut ovat törmänneet. Aihe on itselleni suuri haaste, koska minulla ei ole aiempaa kokemusta esimerkiksi OpenGL-rajapinnasta. Aiemmissa opinnoissa olen käyttänyt XNA-pelinkehitysympäristöä sekä luonut yksinkertaisia pelejä muilla kielillä ja tekniikoilla.

Luvussa kaksi käyn läpi tekniikat, jotka ovat vahvasti läsnä WebGL-rajapinnan kehityksessä. Näitä ovat muunmuassa OpenGL-rajapinta grafiikan piirtämiseen sekä HTML5 kokonaisuudessaan kehitysympäristönä. Näiden lisäksi luvussa kaksi selvitetään avoimen OpenGL-rajapinnan ja suljetun DirectX-rajapinnan kilpailua ja sen vaikutusta WebGL-rajapinnan tukeen. WebGL-rajapinnasta käydään läpi sen hyödyt kehitysympäristönä sekä lyhyesti sen toimintaperiaate.

Erilaisia apuvälineitä ja apukirjastoja on paljon. Luvussa kolme esittelen välineitä ja apukirjastoja, joita olen itse käyttänyt WebGL-sovellusten rakentamiseen. On hyvä valita tarkoituksiinsa oikeat välineet, esimerkiksi tietyn tyyppisiin fysiikan mallinnuksiin tai peleihin on valmiita apukirjastoja. Apukirjastoista on tarjolla peleihin keskittyneitä pelimoottoreita ja yleisemmän tason apukirjastoja, kuten Three.js.

Neljännen luvun tarkoitus on olla pohjustusta peliohjelmointiin Three.js-apukirjastolla, mutta se on myös opas Three.js-apukirjaston perustoiminnoista ja ominaisuuksista. Luvussa esitellyt tekniikat soveltuvat esimerkiksi tiedon visualisointiin, mutta myös yksinkertaisten pelimekaniikkojen luontiin. Tärkeintä on ymmärtää, kuinka Three.js-apukirjastossa luodaan ja käsitellään objekteja.

Viidennessä luvussa esittelen Three.js-apukirjaston soveltamista peliohjelmointiin luomalla yksinkertaisen pelimoottorin. Pelimoottorilla tarkoitan lähes valmista peliä, josta puuttuu vain teema ja 3D-mallit. Sovellus on käytännössä muuten täysin toiminnallinen peli. Three.js-apukirjasto ei sisällä valmiiksi peleissä tarvittavia ominaisuuksia, kuten äänien käsittelyä, törmäyksen tunnistusta tai ohjainlaitteiden käsittelyä. Luvussa käsitelläänkin törmäyksen tunnistusta Three.js-apukirjaston kanssa ottaen huomioon web-ympäristön asettamat rajoitteet. Periaatteet ovat silti hyvin samankaltaisia kuin muissa ympäristöissä.

2 TEKNOLOGIAT

Tässä luvussa käsitellään WebGL-tekniikan taustoja. HTML5 käsittää isomman kokonaisuuden, johon myös WebGL kuuluu. OpenGL on WebGL-rajapinnan sukulainen, joka on jo saanut hyvän otteen markkinoilla. OpenGL-rajapintaa on hyödynnetty jo lähes 20 vuotta hyötysovelluksiin ja pelehin. WebGL-rajapinnan laitetukeen kuulunee vahvasti myös OpenGL-tekniikan kilpailu Microsoftin DirectX-tekniikan kanssa.

2.1 HTML5

Tässä luvussa käydään läpi HTML5:n tärkeimmät elementit tähän opinnäytetyöhön ja WebGL-rajapintaan liittyen. HTML5 tarkoittaa kokonaisuutena paljon muutakin kuin verkkosivujen merkkauskieltä. Se on dynaamisien verkkosivujen kehitysalusta, joka koostuu itse merkkauskielestä ja sen tyylityksestä, Javascript-ohjelmoinnista ja websocket-tekniikasta. HTML5 on tätä kirjoittaessa vielä kehitysvaiheessa, mutta sitä voidaan jo nykyaikaisilla selaimilla hyödyntää täyspainoisesti.

HTML5-merkkauskielen kehitys alkoi vuonna 2004 ja sen oli tarkoitus korvata HTML4 monipuolisempana kehitysalustana. Yksi alkuperäisistä tavoitteista oli tehdä HTML-kielestä monen eri alustan kehitysalusta. HTML5:n tarkoitus onkin tulevaisuudessa korvata selaimen lisäosana tarjotun Flash-alustan monipuoliset peli- ja multimediaominaisuudet. (The History of HTML5 2012.)

2.1.1 HTML-merkkauskieli

Vuonna 2004 perustettu ”WHAT” Working Group tai lyhyemmin WHATWG on Applen, Mozilla Foundationin ja Opera Softwaren työryhmä, joka lähti tutkimaan HTML5-kielen kehitystä. Vuonna 2006 World Wide Web Consortium (W3C) ilmoitti jättävänsä XHTML-kielen kehityksen ja liittyvänsä WHATWG:n mukaan. XHTML oli tavallaan sisarprojekti, jonka oppeja on HTML5-kielen kanssa hyödynnetty. (The History of HTML5 2012.)

Vuonna 2008 julkaistiin ensimmäinen versio HTML5-merkkauskielestä. Tässä vaiheessa HTML5 oli tässä vaiheessa jo paljon muutakin kuin pelkkä merkkauskieli, se olisi tulevaisuuden dynaamisen verkon hallitsija ja kehittyisi kokoajan eteenpäin. Tämän jälkeen useimmat selaimet alkoivat tukea uutta teknologiaa. Monet tunnetut verkkosivustot, kuten Youtube, alkoi tarjota HTML5 vaihtoehtoa Flashin sijaan. Steve Jobs ilmoitti vuonna 2010 hylkäävänsä Flashin ja tukevansa HTML5-teknologiaa Applen laitteissa. Tämän jälkeen HTML5-tekniikan suosio on kasvanut räjähdysmäisesti, ja se onkin jo ottanut asemansa markkinoilla, vaikka standardina se ei ole vielä valmis. (The History of HTML5 2012.)

HTML5 perustuu XHTML-kielen tavoin XML-merkkaukseen. Se tarkoittaa sitä, että verkkosivuston elementit merkataan dokumenttiin erilaisten tagien avulla. Tageilla on alkuelementti ja loppuelementti, joiden sisälle sisältö sijoitetaan. (HTML Introduction 2013.) HTML5-tekniikan uutuudet ovat multimedian puolella. Aiemmin HTML on tukenut valokuvia img-tagin avulla. Nyt samaan tapaan voidaan merkitä dokumenttiin videoita ja audiota. Näiden lisäksi tärkeimpänä on canvas-elementti, joka mahdollistaa grafiikan piirtämisen selainelementissä. (10 new HTML5 tags you need to know about 2012.)

2.1.2 Javascript

Javascript on yksi maailman suosituimmista ohjelmointikielistä. Se on kieli, jota selain tulkitsee ja ajaa. Javascript on myös yksi maailman väärinymmärretyimmistä kielistä ja sitä pidetään yleisimmin vain verkkosivustojen tapahtumien käsittelyyn tarkoitettuna kevyenä ohjelmointikielenä. Vuonna 1995 Netscapen insinöörin Brendan Eichin kehittämä kieli on kuitenkin nostamassa päätänsä ohjelmointikielien parrasvaloissa. Toisin kuin nimestä voisi päätellä, Javascript ei ole sukua Java-ohjelmointikielelle, vaan se perustuu ECMAScript-kieleen. Javascript on ottanut kuitenkin mallia syntaksiinsa Javasta ja C-kielestä. (A re-introduction to JavaScript 2013.)

Suurimmat erot Javascriptin ja esimerkiksi Javan ja C-kielen kanssa on se, ettei muuttujia tarvitse määritellä erikseen kokonaisluvuksi, liukuluvuksi tai merkkijonoksi. Selaimet pitävät huolen muistin käytöstä. Tämä helpottaa ja nopeuttaa Javascriptillä ohjelmointia, mutta ongelmia tehojen käytön kanssa esiintyy varsinkin kuormittavissa WebGL-sovelluksissa. Javascript on myös olio-ohjelmointiin perustuva kieli, mutta toisin kuin Java, Javascriptissa ei ole luokkia. Sen sijaan se käyttää olioprototyyppejä ja funktioita, joita voidaan käyttää olioina. (A re-introduction to JavaScript 2013.)

Javascriptin käyttö on muihin kieliin nähden hieman suoraviivaisempaa. HTML-dokumenttiin sitä voidaan kirjoittaa script-tagien sisään, vaikka vain yksittäisinä riveinä tai kokonaisina sivua ohjaavina olioina. Mitään main-funktiota tai funktioita ylipäänsä ei ajamiseen kuitenkaan tarvita. (Javascript How To 2013.) Kuitenkin Javascriptista saa paremmin ominaisuuksia irti, kun osaa hyödyntää funktioita ja kielen olio-ominaisuuksia. HTML5 ajattelun myötä verkkosivuistot lähentelevät

kokoajan enemmän perinteistä sovelluskehitystä, jossa Javascriptilla on entistä isompi rooli.

2.1.3 Canvas-elementti

Apple esitteli HTML5-kielen canvas-elementin WebKit-selainmoottoriin vuonna 2004. WebKit-moottoria käyttävät useimmat selaimet tietokoneilla ja mobiililaitteilla poislukien Microsoftin Internet Explorer. Applen idea canvasiin oli tuoda Mac OSX-käyttöjärjestelmän Quartz piirtorajapinta HTML ja Javascript -käyttöön, jotta erilaisten widgettien Applen Dashboardiin helpottuisi. Lopulta tämä piirtorajapinta päätyi myös Applen Safari-selaimeen, Mozillan Firefox-selaimeen ja lopulta kaikkiin WebKit-selainmoottoria käyttäviin selaimiin vuoteen 2010 mennessä. Internet Exploreriin canvas tuli hieman jälkijunassa vasta versioon 9.0 vuonna 2011. (HTML5 Canvas is a go! 2010.)

Alkuperäinen canvasin piirtorajapinta sai seurakseen WebGL-rajapinnan. Näitä kahta käytetään kutsumalla canvas-elementin kontekstia. Alkuperäinen rajapinta kutsutaan 2D-kontekstilla ja WebGL-rajapintaa kutsutaan webgl-kontekstilla. Kummallakin piirtorajapinnalla on oma syntaksi piirtämiseen. (HTML5 canvas - the basics 2009.) Suurin ero näiden rajapintojen välillä on se, että WebGL-rajapinta käyttää tietokoneen näytönohjainta piirtämisen apuna ja on sen takia paljon tehokkaampi piirtämisessä.

TAULUKKO 1 Canvas-elementin kontekstien tuki viidellä suosituimmalla selaimella (Compatibility tables for support of HTML5, CSS3, SVG and more in desktop and mobile browsers 2012)

	Internet Explorer	**Firefox**	**Chrome**	**Safari**	**Opera**
2D-konteksti	9.0 (2011)	2.0 (2006)	4.0 (2010)	3.1 (2008)	9.0 (2006)
WebGL	-	4.0 (2011)	9.0 (2010)	5.1 (2011)	12.0 (2012)

Taulukosta 1 voidaan nähdä canvas-elementin piirtorajapintojen tukea eri selaimilla. Taulukkoon on merkitty varhaisin selainversio, jolla piirtäminen onnistuu kyseisellä rajapinnalla. Tuki on jokaisella selaimella pysynyt samana niiden jälkeen. Version lisäksi taulukkoon on suluissa merkitty version julkaisuvuosi. Taulukosta voidaan huomata, että vaikka canvas-elementti syntyi Applen toimesta, ei se kuitenkaan virallisesti tullut Applen selaimeen ensimmäisenä. Chromella on ollut todella nopea

eteneminen versioissa. 2D-konteksti tuli Chromeen alkuvuonna 2010 ja myöhemmin samana vuonna lisättiin tuki WebGL-kontekstille.

2.1.4 WebSocket

HTML5 toi mukanaan Websocket-tekniikan korvaamaan hitaamman AJAX, eli Asynchronous Javascript And XML, -tekniikan. Näillä kahdella tekniikalla sovellus hakee palvelimelta tietoa sovellukseen ilman, että selaimen tarvitsee päivittää sivustoa uudelleen. AJAX hoitaa asian luomalla uuden yhteyden palvelimeen, lataa palvelimelta ohjelman ja vastaanottaa tiedot. Tämä tapahtuu joka kerta, kun ohjelman käyttäjä haluaa päivittää tietoja. Websocket kehitettiin suoristamaan mutkia ja mahdollistamaan nopeammat yhteydet. (Websocket vs Ajax 2011.)

Websocket käyttää HTTP-protokollan mukaista yhteyttä palvelimeen luodakseen kaksisuuntaisen yhteyden. Turvallisuusmalli tekniikalle on sama, mitä selaimet käyttävät muutenkin palvelinpyyntöihin ja nettisivujen lataamiseen. (RFC 6455 – The Websocket Protocol 2011.) HTTP-protokolla mahdollistaa samojen porttien käyttämisen palvelimella, kuin normaalissa nettisivujen lataamisessa, joten viestien kulku ei pysähdy palvelinten palomuureihin. Asiakasohjelman ei tarvitse huolehtia yhteyden säilymisestä, koska tekniikkaan kuuluu sisäänrakennettu yhteyskokeilu. Yhteyskokeilusta huolehtii selain ja palvelimelle rakennettu yhteensopiva ohjelma. Websocket toimii siis hyvin samalla tavalla, kuin esimerkiksi peleissä käytetyt palvelimet, mutta sen yhteyden käyttämä portti on varmasti aina auki eri palvelimilla. (Websockets 101 2012.)

Aiemmin mainittuun AJAX-tekniikkaan verrattuna on Websocketin etuna se, että Websocketin ja palvelimen välinen työskentely on kevyempää. Websocket käyttää vain yhtä yhteyttä palvelimeen, avaa palvelimelta ohjelman kerran ja vastaanottaa tietoja. Toisin kuin AJAX, Websocket ei tee operaatiota uudelleen jokaisella tietopyyntökerralla. Lisäksi Websocket-protokollalla palvelin lähettää automaattisesti uusimman tiedon selaimelle, joten sovelluksen ei tarvitse tutkia palvelinta uuden tiedon varalta. (Websocket API 2012.)

Websocket tuo HTTP-ympäristöön nopeaan tiedonsiirtoon perustuvan palvelintekniikan. Se mahdollistaa esimerkiksi moninpelien yhteydet.

Moninpelaaminen selaimessa ei ole uusi asia, mutta Websocketin käyttämä tapa helpottaa sovelluskehitystä selainympäristössä.

2.2 OpenGL

OpenGL eli Open Graphics Library on nimensä mukaisesti avoin grafiikkakirjasto ja ohjelmointirajapinta. OpenGL on kielestä ja alustasta riippumaton ja onkin sen vuoksi hyvin suosittu CAD-ohjelmistoissa ja videopeleissä. OpenGL tarjoaa ohjelmoijalle rajapinnan tietokoneen laitteiston hyödyntämiseen 2D ja 3D -grafiikan piirtämisessä. (The History of OpenGL 2012.)

2.2.1 Historia

Vielä 1980-luvulla 2D ja 3D -grafiikkaa hyödyntävien ohjelmiston kehitys oli hidasta ja kallista. Jokaisen ohjelmiston kohdalla täytyi erikseen kirjoittaa ajurit erilaisia tietokonekokoonpanoja varten. Monet eri ohjelmistoyritykset kehittivät omia ajureitaan, vaikka alustoina olivat samat tietokonekokoonpanot. (The History of OpenGL 2012.)

1990-luvulle tultaessa Silicon Graphics inc. (SGI) oli työasemien 3D-grafiikan markkinajohtaja. SGI käytti omaa IRIS GL-grafiikkakirjastoa, joka oli kehitetty SGI:n omia laitteistoja varten, eikä ollut käytettävissä yrityksen ulkopuolella. IRIS GL-kirjastoa pidettiin helppokäyttöisenä ja se tuki välitöntä kuvan renderöimistä. (The History of OpenGL 2012.)

SGI:n kilpailijat, kuten IBM, Sun Microsystems ja Hewlett-Packard alkoivat tuoda omia laitteistojaan markkinoille. Kilpailijoiden laitteet käyttivät PHIGS-ohjelmointirajapintaa 3D-grafiikan piirtämiseen. Ja vaikka IRIS GL-kirjastoa pidettiin parempana rajapintana PHIGS-kirjastoon verrattuna, alkoi SGI:n markkinaosuus laskea. Tämän vuoksi SGI päätti tehdä IRIS GL-kirjastosta kaikille avoimen rajapinnan, mutta koska se oli jo patentoitu ja lisensoitu, SGI ei voinut käyttää sitä suoraan. Tämän vuoksi SGI kehitti OpenGL-rajapinnan IRIS GL-kirjaston pohjalta ja teki siitä täysin avoimen ohjelmointirajapinnan. OpenGL valmistui ja hyväksyttiin ARB:n (Architectural Review Board) toimesta vuonna 1992. ARB:n perustajajäseniin kuuluivat SGI, Microsoft, IBM, DEC ja Intel. OpenGL-rajapinnan kehitys on sen

jälkeen perustunut näytönohjainvalmistajien ja esimerkiksi pelinkehittäjien tekemiin OpenGL lisäosiin ja vaatimuksiin. Itse OpenGL-rajapinnasta ei uusia versioita ilmesty kovin usein, mutta näytönohjainvalmistajat voivat lisäosilla luoda uusia ominaisuuksia rajapinnalle. (The History of OpenGL 2012.)

2.2.2 OpenGL ES

OpenGL ES on rojaltivapaa ja alustasta riippumaton 2D- ja 3D-grafiikkaan keskittynyt ohjelmointirajapinta. Se on karsittu versio tavallisesta OpenGL-rajapinnasta ja se on tarkoitettu sulautetuille järjestelmille, kuten pelikonsolit, älypuhelimet, kodinkoneet ja ajoneuvot. Sen ominaisuudet ovat tarkasti valikoituneet suorituskyvyn ja joustavuuden varmistamiseksi. OpenGL ES-rajapinnan voi sulauttaa suoraan järjestelmään tai pitää se erillisenä rajapintana. OpenGL ES 2.0 perustuu OpenGL 2.0 versioon ja tarjoaa täysin ohjelmoitavaa 3D-grafiikkaa. (OpenGL ES - The Standard for Embedded Accelerated 3D Graphics 2012.)

Älypuhelimissa OpenGL ES 2.0 törmää väistämättä peleissä, mutta esimerkiksi Googlen Chrome-selain tukee sitä natiivisti. Tämä tarkoittaa sitä, että Chromelle voidaan ohjelmoida natiiveja OpenGL ES 2.0 sovelluksia ja pelejä samalla tavalla kuin älypuhelimille. Chrome onkin itse ohjelmoitava alusta, joka on toisen alustan, käyttöjärjestelmän, päällä. (Native Client, 2012.)

Selaimissa yleisimmin törmää myös käsitteeseen WebGL, jonka avaan myöhemmin tässä opinnäytetyössä. WebGL perustuu kuitenkin läheisesti OpenGL ES 2.0-rajapinnan spesifikaatioon, joka tarkoittaa sitä, että sen ominaisuudet ovat hyvin samanlaisia. OpenGL ES 2.0-rajapinnan kanssa työskennelleet voivat tuntea olonsa tutuksi WebGL-rajapinnan parissa. (WebGL – OpenGL ES 2.0 for the Web 2012.)

2.2.3 DirectX ja kilpailu OpenGL-rajapinnan kanssa

Vuonna 1994 Microsoft oli kehittämässä Windows 95-käyttöjärjestelmäänsä ja oli huolissaan pelinkehittäjien pysyvän MS DOS-järjestelmässä. Monet pitivät DOS-järjestelmää Windows-käyttöjärjestelmää parempana alustana pelien kehitykseen, koska DOS tarjosi suoran yhteyden näytönohjaimeen, äänilaitteisiin ja ohjainlaitteisiin. Tätä varten Microsoft kehitti Windows-käyttöjärjestelmälle DirectX-

rajapinnan, jolla päästään suoraan käsiksi edellä mainittuihin tietokoneen osiin. Tämän jälkeen DirectX on yleistynyt Microsoftin käyttöjärjestelmissä ja esimerkiksi Xbox-pelikonsolissa. (The History of DirectX 2012.)

Wolfire on yksityinen pelinkehittäjä, joka on valinnut OpenGL-rajapinnan peliensä kehitykseen DirectX-rajapinnan sijaan. Wolfiren mukaan (2010) OpenGL on tuettu lähes kaikilla alustoilla paitsi Microsoftin Xbox-pelikonsolissa, kun taas DirectX-rajapinnan tuki jää vain Microsoftin laitteisiin ja alustoihin. Näihin OpenGL-rajapinnan tukemiin alustoihin kuuluvat Microsoftin Windowsin lisäksi, Applen tietokoneet ja puhelimet, Linux-tietokoneet sekä pelikonsoleista Sonyn PlayStation 3 ja kannettavat PlayStationit, Nintendon Wii ja DS. Windows on kuitenkin suosituin käyttöjärjestelmä ja on suosionsa ansiosta kuitenkin ollut jo pitkään se alusta, jolle pelinkehittäjät haluavat pelinsä tulevan. Myös tästä syystä DirectX on ollut myös suositumpi vaihtoehto OpenGL nähden, koska se tukee grafiikan lisäksi äänen ja ohjainlaitteiden ohjelmointia. DirectX-rajapinnan tuki näytönohjainvalmistajien keskuudessa myös hieman parempi OpenGL-rajapintaan verrattuna. OpenGL-rajapinnan suosio on uhka Microsoftille, koska se haluaisi pelien kehittäjien pysyvän Windows ja Xbox -ympäristöissä. Wolfire on myös alustariippumattomuuden lisäksi sitä mieltä, että OpenGL on DirectX-rajapintaa tehokkaampi grafiikan piirtämisessä ja on näin parempi vaihtoehto pelien kehittämiseen.

Microsoft ei tue Internet Explorer-selaimessaan WebGL-rajapintaa. Microsoftin mukaan WebGL on turvallisuusaukko, koska nettisivu voisi sillä hallita näytönohjainta. Haitallinen sivusto voisi syöttää näytönohjaimen ajurin kaatavaa koodia ja saada tietokoneen jähmettymään kokonaan. Toinen näkökulma tähän on se, että OpenGL-rajapinnan ja DirectX-rajapinnan kilpailun takia Microsoft ei halua tukea kilpailijan sisarprojektia. Tällä hetkellä Microsoft on ainoa selainvalmistaja, joka ei tue tai aio tukea WebGL-rajapintaa selaimessaan. (Internet Explorer Won't Be Supporting WebGL 2011.)

2.3 WebGL

Edellisessä luvussa esitelty OpenGL sai uuden elämän selaimissa WebGL-rajapinnan muodossa. WebGL on tätä kirjoittaessa saanut jo virallisen valmiin version, mutta se

ei ole vielä yleinen standardi. Tässä luvussa esitellään WebGL-rajapinnan kehityshistoria ja muutamia oikean maailman esimerkkejä sen hyödyntämisestä.

2.3.1 Historia

Ennen canvas-elementin tuloa selaimissa 3D-grafiikan piirtoon käytettiin muita teknologioita. Ne olivat selaimeen asennettavia lisäosia, jotka käyttivät tietokoneen näytönohjainta hyväkseen piirtämisessä. Toinen oli Java OpenGL (JOGL) ja toinen Flashin Stage3D. JOGL toimi selaimessa Java applettina, joka vaati selaimeen asennettavan Javan lisäosan. (De opkomst van WebGL 2013.)

Näiden pohjalta vuonna 2006 Mozillan työntekijä Vladimir Vukicevic aloitti kokeilut niin sanotulla Canvas3D-tekniikalla, joka perustui jo siihen mennessä julkaistuun canvas-elementtiin. Canvas-elementin lisäosaksi ensimmäisenä tehty moz-gles11-konteksti seurasi OpenGL ES 1.1-standardia lähes identtisesti. Tämän jälkeen toiseksi lisäosaksi kehitetty moz-glweb20 seuraa läheisesti OpenGL ES 2.0-standardia. Molemmat kontekstit ovat implementoitu suoraan käyttöjärjestelmän OpenGL-rajapinnan päälle, jonka takia koneelta vaaditaan vähintään OpenGL 1.5 version tukea. (Canvas 3D: GL power, web-style 2007.)

Myös Operan kehittäjät tekivät oman implementaationsa Canvas3D-tekniikasta. Mozilla ja OpenGL-teknologiaa kehittävä Khronos Group alkoivat yhdessä kehittää standardia nimeltä WebGL. Ensimmäinen virallinen versio WebGL-rajapinnasta julkaistiin maaliskuussa 2011. Lopputuloksena oli 3D-grafiikkaa selaimessa piirtävä elementti ilman mitään erikseen asennettavia lisäosia. (De opkomst van WebGL 2013.)

OpenGL oli standardisoitunut, suosittu ja monialustainen rajapinta grafiikan piirtämiseen. Näiden seikkojen vuoksi se ajautui myös WebGL-standardin pohjaksi. Lisäksi WebGL hyötyy jo OpenGL-rajapinnan kanssa työskennelleistä, koska heillä oli jo valmiiksi oikeanlaista kokemusta. Vukicevic ennusti jo vuonna 2007, että mobiililaitteet ovat Webin tulevaisuus ja WebGL-rajapinnan perustuminen mobiililaitteilla suosittuun OpenGL ES:n mahdollistaa WebGL-rajapinnan tulemisen myös mobiililaitteille. (Canvas 3D: GL power, web-style 2007.)

On kuitenkin huomioitava, että vaikka WebGL perustuu OpenGL-rajapintaan, itse piirtäminen tapahtuu alustasta riippuen eri tavoilla. Google Chrome ja Mozilla Firefox -selaimet käyttävät Windows-ympäristössä ANGLE-kääntäjää, joka kääntää WebGL-rajapinnan kutsut DirectX 9-rajapinnalle tuttuun muotoon. Muilla alustoilla ja selaimilla kutsut menevät joko OpenGL tai OpenGL ES-rajapintojen kautta. (ANGLE 2013.) Ratkaisu on hämmentävä, mutta jos OpenGL-rajapinnan tuki muuttuu Windows-alustalla, selviää WebGL muutoksesta ainakin edellä mainittujen selaimien kohdalla.

2.3.2 Miksi WebGL?

WebGL on uusi tekniikka ja sillä on kilpailijoita, jotka ovat olleet markkinoilla pidempään. Ei pelkästään Web-ympäristössä, vaan myös perinteisempien sovellusten maailmassa. Miksi pitäisi valita WebGL oman sovelluksen tai pelin graafisen ympäristön rakentamiseen? Eräs asiaan perehtynyt on Florian Boesch, joka blogikirjoituksessaan (Why you should use WebGL 2013) käy läpi WebGL-rajapinnan etuja kilpailijoihinsa nähden.

WebGL-rajapinnan kutsumiseen tarvitaan vähemmän työtä kilpailijoihin nähden. Ei ole tarvetta tehdä tiedostoja, lisätä kirjastoja tai kutsua erikseen apuvälineitä. WebGL-rajapinnan kutsumiseen tarvitaan kolme riviä: Canvas-elementin luonti, sen lisääminen dokumenttiin ja kontekstin määritteleminen WebGL-rajapinnaksi. Työn määrän vertailulla Boesch viittaa todennäköisesti Windows tai Linux -pohjaisiin sovelluksiin. Boesch huomauttaa, että selain tekee ohjelmoijan puolesta samat työt, mitä ohjelmoijan pitäisi tehdä toisessa ympäristössä. (Why you should use WebGL 2013.)

Grafiikkaohjelmoinnissa näytönohjaimien ajurien tuki ja kehittyminen on yksi teknologian kehityksen kulmakivistä. WebGL on yksi testatuimmista ja nopeimmin tukea saavista rajapinnoista nykyään. Tämä johtuu siitä, että sen testaaminen erilaisilla kokoonpanoilla on helppoa. Testatakseen WebGL-rajapinnan tukea ei tarvitse kuin laittaa osoiteriville testisivun osoite ja klikata testin aloittaminen. Muissa ympäristöissä tämä voisi vaatia ohjelmien ja ajureiden lataamista. Toisin sanoen testaaminen on lähempänä peruskäyttäjää ja testaamiseen on Khronos Groupin oma testausohjelmisto, joka on kaikille samanlainen. Tämä tuo testaamiseen isomman

laitekannan ja ongelmien raportointi on saumatonta. Boesch mukaan tämä kehittää ensisijaisesti WebGL-rajapintaa teknologiana, mutta myös toissijaisesti OpenGL ja DirectX-rajapintoja. Myös laitekehittäjät kuten Intel, AMD ja Nvidia ovat huomioineet WebGL-rajapinnan edut laitekehityksessään. Boesch on myös sitä mieltä, että WebGL-rajapinnan suosio kiihdyttää myös selainvalmistajien Javascript-moottorien nopeuskilpailua ja edistää kehitystä selaimien puolella. (Why you should use WebGL 2013.)

OpenGL-rajapinnan tavoin WebGL on laajennettavissa ilman suuria versiopäivityksiä. Tämä tarkoittaa sitä, että kehittäjä voi ottaa mukaan uusia ominaisuuksia lennosta. WebGL onkin ainoa 3D-teknologia Web-ympäristössä, joka voi hyödyntää uusia ominaisuuksia samantien. Nämä laajennukset voivat tulla OpenGL-rajapinnasta tutulla tavalla kehittäjiltä yhteisölle, mutta pääasiassa laajennukset tulevat WebGL-rajapintaan Khronos Groupin kautta. (Why you should use WebGL 2013.)

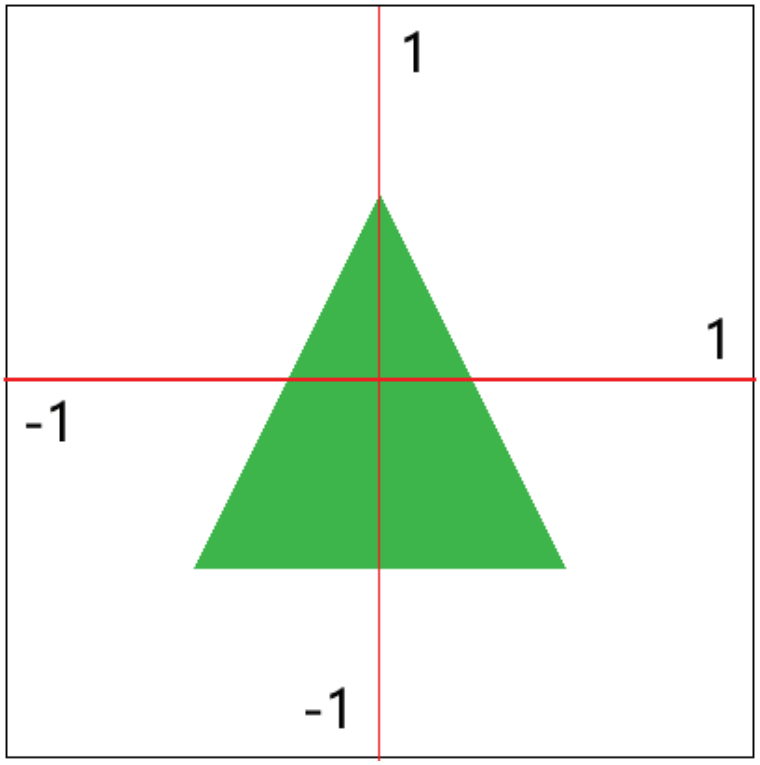
Mitä tämä kaikki tarkoittaa kaltaisilleni kehittäjille, jotka käyttävät apukirjastoja? Sovelluskehittäjien ei tarvitse välttämättä koskea ollenkaan raakaan WebGL-rajapinnan koodiin, mutta eri apukirjastojen kehittäjät voivat lisätä uusia ominaisuuksia lennosta sovelluskehittäjien käsiin. Boesch kertoo blogissaan WebGL-rajapinnan helppoudesta verrattuna muihin teknologioihin. On kuitenkin huomattava, että piirtämiseen ja objektien pyörittelyyn vaaditut logiikat pysyvät samoina teknologiasta riippumatta. WebGL-rajapinnan helppous perustuu aloittamisen helppoudesta ja työkalujen suoraviivaisuudesta. Helppous ja suoraviivaisuus onkin suuri etu WebGL-apukirjastojen kehittäjille ja rajapinnan pioneereille.

2.3.3 GLSL

GLSL eli OpenGL Shading Language on niin sanottu varjostinkieli. OpenGL Architecture Review Board (OpenGL ARB) kehitti GLSL-varjostinkielen, jotta kehittäjillä olisi enemmän työkaluja grafiikan piirtämiseen. C-kieleen perustuvan varjostinkielen avulla kehittäjien ei tarvitse ohjelmoida grafiikkaa laitekohtaisesti, koska OpenGL-rajapintaa tukeva näytönohjain tekee kaiken työn. Tämän vuoksi varjostinkieli toimii laitteistosta riippumatta ja antaa kehittäjille vapauden luoda. GLSL tuotiin alunperin lisäosana OpenGL versiossa 1.4, mutta siitä tehtiin osa ydintä

OpenGL 2.0-versiossa. Tätä myöten se on myös periytynyt WebGL-rajapinnan puolelle. (GLSL Explained 2013.)

Käytännössä kaikki piirtäminen tehdään näiden varjostimien kautta. Vertex-varjostin laskee pikselin paikan ruudulla ja fragment-varjostin laskee sen värin. Varjostimet laskevat kuinka WebGL-oliolle ilmaistut muodot lopulta piirretään ruudulle riippuen niille määritellyistä pinnoista, käännöksistä, valaistuksesta ja katselukulmasta. Monissa grafiikkarajapinnoissa varjostimien käyttö on vapaaehtoista, mutta WebGL vaatii varjostimien määrittelyä. Ilman varjostimia mitään ei piirretä ruudulle. (Parisi 2012, 7-8.)



KUVA 1. WebGL-koordinaatisto ja siihen piirretty kolmio

Tony Parisin (2012, 10) mukaan WebGL-sovelluksen anatomiaan kuuluu vähintään 8 osaa: canvas-elementin luominen, määritetään canvas-elementin konteksti WebGL-rajapinnaksi, määritellään viewport eli ikkuna WebGL-maailmaan, luodaan elementeille yksi tai useampi puskuri, määritellään yksi tai useampi matriisi elementtien sijoittelua ja pyörittämistä varten, luodaan yksi tai useampi varjostin piirtämisalgoritmia varten, yhdistetään elementtien data varjostimien kanssa ja lopuksi piirretään. Kuvassa 1 on piirretty kolmio täysin ilman apukirjastoja. Näinkin yksinkertaisen muodon toteutukseen tarvitaan jopa 53 rivia koodia. Kuvan päälle on

editoitu WebGL-canvasin käyttämä koordinaatisto piirtämiseen, jossa origo on aina keskellä ja reunat ovat arvoltaan 1 tai -1. Tämän koordinaatiston mukaan syötetään muodoille kulmien koordinaatit matriisissa, jonka mukaan WebGL-olio piirtää ne canvasille. Koordinaatisto onkin yksi suurimmista eroista 2D-kontekstin canvasiin, koska siinä origo on aina vasemmassa yläkulmassa ja suurimmat arvot riippuvat canvasille asetetusta koosta.

2.3.4 WebGL-sovellukset maailmalla

Selainvalmistaja Mozilla on yksi aktiivisimmista WebGL-tekniikan kehittäjistä. Mozilla julkaisi Epic Gamesin Unreal Engine 3-pelimoottorin käännettynä WebGL-tekniikalle. Pelimoottori mahdollistaa aiempaa paremman grafiikan ajamisen WebGL-tekniikalla tehokkaasti. Mozilla paransi projektinsa ansiosta myös oman javascript-tulkkinsa nopeutta. Unreal Engine 3 on ollut hyvin laajassa käytössä perinteisemmillä pelialustoilla, mutta Mozillan ansiosta nämä pelit voidaan tuoda selaimeen pelattaviksi. (Mozilla is Unlocking the Power of the Web as a Platform for Gaming, 2013.)

Tinkercad on selaimessa toimiva CAD-ohjelmisto, joka on suunnattu kaikelle kansalle. Asiakas voi selaimessaan suunnitella tai tuoda valmiin 3D-mallin ja tilata siitä 3D-tulostimella tulostetun kappaleen. Esimerkiksi itsesuunniteltuja silmälasikehyksiä tai kännykän kuoria. Tinkercad on toteutettu WebGL-rajapintaa käyttäen ja vaikka yhtiön päämaja sijaitsee Yhdysvalloissa, on sillä suomalaiset perustajajäsenet. (Tinkercad 2013.)

3 APUVÄLINEET

Tässä luvussa esittelen hyväksihavaittuja apuvälineitä WebGL-sovelluskehitysympäristöön. Erilaisiin apuvälineisiin kuuluvat käytettävät ohjelmat ja osittain valmiit ohjelmointiratkaisut. Sovelluskehitys ei vaadi muuta kuin WebGL-rajapintaa tukevan selaimen ja tekstieditorin, mutta tekstieditorinkin olisi hyvä ymmärtää HTML5-ympäristön eri kieliä. Tämän lisäksi voidaan käyttää apuna omaa palvelinta sekä 3D-mallinnusohjelmaa omien elementtien luontiin.

Pyörää ei kannata keksiä uudelleen, jos joku on sellaisen jo keksinyt. WebGL-rajapinnan työstäminen on erittäin aikaa vievää, jos siinä aina joutuu tekemään kaiken alusta. Tätä varten on luotu erilaisia ohjelmointikirjastoja eri tarkoituksiin ja eri kielillä. Kirjasto tarkoittaa sitä, että esimerkiksi Javascript-tiedosto sisältää valmiita funktioita ja olioita erilaisten perustoimintojen luomiseen. (Ohjelmointi 1 2010.) HTML5-maailmasta yksi tunnetuimmista apukirjastoista, jQuery, auttaa normaalien internetsivustojen käsittelyssä, mutta sen toiminnot eivät ylety WebGL-rajapinnan puolelle. WebGL varten on paljon erilaisia kirjastoja, joilla on erilaisia lähestymistapoja aiheeseen. Monet ovat hyvin pitkälti pelejä ja multimediaa varten luotuja, mutta yleiseen sovelluskehitykseen löytyy muunmuassa kirjasto nimeltään Three.js. Apukirjastot käsittävät valmiita ratkaisuja jopa erityyppisiin puolivalmiisiin peleihin, fysiikkamoottoreihin tai pelkästään pohjan 3D-ympäristölle.

3.1 Three.js

Three.js on apukirjasto grafiikan luomiseen WebGL-rajapinnalla. Tavallisen kuution luominen ilman apukirjastoja voi vaatia satoja rivejä Javascript- ja varjostinkoodia, mutta apukirjaston avulla siihen tarvitaan vain muutama. (Creating a scene 2013.) Three.js kehitys alkoi GitHub-palvelussa itseään mrdoobiksi kutsuvan miehen toimesta vuonna 2010. Se on avoimen lähdekoodin kirjasto, jonka kehittämisestä on tähän mennessä vastannut lähes 100 eri henkilöä. (Three.js contributors 2013.)

Myös dokumentaatio on yhteisön vastuulla, joten sen laatu on hyvin vaihtelevaa. Joistakin ominaisuuksista ei välttämättä löydä ollenkaan dokumentaatiota, Three.js-apukirjastolla tehtyjä esimerkkejä tai edes foorumikirjoituksia aiheesta. Tällöin yksi hyvä keino on ottaa mallia kokonaan muista kehitysympäristöistä ja soveltaa omaa osaamistaan sen mukaan.

Three.js on keskittynyt täysin grafiikan luomiseen 3D-objekteista varjostimiin ja erilaisiin tehosteisiin. Siihen on kuulunut myös lisäosina esimerkiksi näppäimistön käsittelyä ja törmäyksen tunnistusta. Törmäyksen tunnistus on kuitenkin poistettu Three.js-kirjastosta kokonaan ja jätetty ilman vaihtoehtoja (Removed Collision code 2013.) Tämä kannattaa ottaa huomioon varsinkin sellaisissa oppaissa, jotka käsittelevät törmäyksen tunnistusta ja ovat julkaistu ennen vuoden 2012 heinäkuuta.

Three.js-kirjaston uudemmissa versioissa ei välttämättä toimi kaikki toiminnot, joita oppaissa esitellään.

3.2 Physijs

Physijs on apukirjasto fysiikan mallinnukseen Three.js-apukirjaston kanssa. Se sisältää itsessään ammo.js-apukirjastoon perustuvan fysiikkamoottorin ja funktiot, jotka linkittävät objektit suoraan Three.js-apukirjaston luomiin elementteihin. Ammo.js on puolestaan yleisempi apukirjasto, joka on suora käännös alunperin c++-kielellä kirjoitetusta Bullet-fysiikkamoottorista. Bullet-fysiikkamoottorin oliot ja syntaksi on periytynyt suoraan myös Physijs-apukirjastoon. (Physijs 2013.) Tämän vuoksi itse Physijs-apukirjastosta ei ole omaa dokumentaatiota, joten apuna on syytä käyttää Bullet-fysiikkamoottorin dokumentaatiota.

Physijs-apukirjaston kanssa on otettava huomioon, että viralliset esimerkit käyttävät varsin vanhentunutta versiota (versio 51) Three.js-kirjastosta. Uudemmilla versioilla voivat tuloket olla avaamattomia, kuten objektien törmäykset eivät rekisteröidy ja objektit tippuvat oletetun lattian läpi tyhjyyteen. Physijs käyttää apunaan uusien selaimien webworker-ominaisuutta, joka jakaa Javascript käskyjä tausta-ajoon ja mahdollistaa moniydinprosessoreiden hyödyntämisen. Tästä voi sivuoireina tulla hidasta sovelluksen latautumista tai toimimattomuutta, jos selain ei tue webworkeria.

3.3 Muita kirjastoja

Three.js-kirjaston lisäksi on myös muita itsenäisiä apukirjastoja. GLGE-kirjasto on luotu nopeuttamaan WebGL-dokumentin luontia, jotta kehittäjä voi keskittyä itse sovelluskehitykseen. GLGE-kirjasto on helppo ja nopea käyttää ja siinä on erinomainen COLLADA-mallien tuki sekä helppokäyttöinen tuki fysiikoille. SceneJS-kirjasto tarjoaa JSON-tiedostomuotoon perustuvan grafiikkarajapinnan WebGL-grafiikan piirtämiseen. SceneJS erikoistuu suurien objektimäärien tehokkaaseen piirtämiseen. Objektit voivat olla valittavissa ja siirrettävissä, joka tekee kirjastosta mainion apuvälineen insinööreille tai vaikkapa käyttöön lääketieteen alalla korkeatasoisten 3D-mallien käsittelyyn. CubicVR on korkeatasoinen OpenGL 2.0- ja OpenGL ES-moottori, joka on käännetty Javascript-kielen ymmärtämään muotoon

C++-kielestä. Mozilla Labs on käyttänyt CubicVR-kirjastoa WebGL kokeilujen ja Gladius-pelimoottorin pohjana. (Parisi 2012, 138.)

3.4 Ohjelmat

WebGL-rajapinnalla kehittäminen ei vaadi erityisiä välineitä verrattuna muuhun web-ohjelmointiin. Tarjolla ei ole WebGL-rajapinnalle tai sitä hyödyntäville apukirjastoille tarkoitettuja kehitysympäristöjä. Käytän ohjelmointiin Notepad++-tekstieditoria sen keveyden ja kielisyntaksin korostuksen vuoksi. Testaamiseen käytän Googlen Chrome-selainta, mutta on hyvä tarkistaa sovelluksen toimivuus myös esimerkiksi Mozillan Firefox-selaimella.

Javascript ja WebGL eivät itsessään vaadi taustalle palvelinta, mutta palvelin on pidettävä taustalla päällä, jos ladataan kuvia tekstuureiksi tai 3D-malleja kansioista. Nykyaikaiset selaimet ovat estäneet Javascript-komennoilla lataamisen ulkopuoliselta palvelimelta ja tällaiseksi ne tulkitsevat tilanteen, jossa ei ole palvelinta ollenkaan. Palvelinta tarvitaan myös, jos tahdotaan käyttää websocket-ominaisuuksia hyväksi. Palvelimeksi valitsin ilmaisen Wamp Server-ohjelmiston.

3D-mallinnukseen käytän 3ds MAX-ohjelmistoa, koska se on tullut tutuksi aiemmissa opinnoissa ja 3ds MAX-ohjelmistolle on tarjolla kääntäjä Three.js-apukirjaston omalle JSON-formaatin malleille. SketchUp-ohjelmiston kääntämät COLLADA-formaatin mallit voivat toimia hyvin epämääräisesti varsinkin Three.js-apukirjaston kanssa. Three.js-apukirjaston kehittäjät tukevat mielellään Blender-ohjelmistoa kääntäjien ja muiden lisäosien osalta. He ovat kehittäneet myös oman online-editorin, jossa voi testata mallien toimivuutta, asettaa tekstuureja tai tehdä yksinkertainen 3D-ympäristö.

4 POHJAN RAKENTAMINEN

Tässä opinnäytetyössä käytetään Three.js-apukirjastoa WebGL-sovelluksen luontiin. Three.js-sovellukselle voidaan tehdä yleinen pohja, josta on hyvä lähteä rakentamaan omaa sovellusta. Erilaisia tapoja rakentaa sovelluksia on paljon, yhtä monta kuin on ohjelmoijia, mutta esittelemäni tapa on itse hyväksi havaittu ja esiintyy pääpiirteittäin myös muiden ohjelmoijien toimesta, jotka käyttävät Three.js-apukirjastoa. Myös

apukirjaston pääkehittäjä ohjaa käyttämään samankaltaista tapaa (Creating a scene 2013.) Tässä luvussa esitellään Three.js-sovelluksen pohja, Three.js-kirjaston 3D-objektit, materiaalit, valot ja kamera. Lisäksi tässä luvussa esitellään yksinkertaisen olion luonti javascriptilla sekä objektien liikuttaminen näppäimistön avulla.

Tässä opinnäytetyössä on hyödynnetty Three.js-apukirjaston versioita r53 ja r52. Onkin suositeltavaa testata tämän luvun esitttelemät metodit samalla versiolla, koska avoimeen lähdekoodiin perustuva Three.js-kirjaston olioiden ja funktioiden syntaksi voi vaihdella versioittain. Perusasioiden hallitseminen jollain versiolla auttaa uuteen versioon siirtymistä.

```
<!DOCTYPE html>
<html>
<head>
<!-- Head-tagien sisälle tulee dokumentin määrittelyt -->
<title>Three.js pohja</title>
<!-- Esitellään Three.js apukirjasto, joka tässä tapauksessa
     on kansiossa nimeltä js -->
<script src="js/three.js"></script>
</head>
<body>
<!-- Body-tagien sisälle tulee käyttäjälle näkyvä materiaali -->
<script>
/*  Selain ajaa tämän koodin dokumentin latauduttua.

    Tänne tulee itse sovelluksen koodi. */
</script>
</body>
</html>
```

KUVA 2. HTML5-standardin mukainen pohja

Kuvassa 2 esitellään pohja, jolle tuleva sovellus rakennetaan. HTML-merkkaus sallii sen, että three.js apukirjasto kopioitaisiin suoraan script-tagien sisälle. Se sallii myös sen, että itse rakennettu sovellus kirjoitetaan javascript-tiedostoon ja kerrotaan HTML-dokumentille vain sen sijainti. Kaikki seuraavaksi esitelty ohjelmointi on kirjoitettu body-tagien sisällä oleviin script-tageihin, jos muuta ei ohjata tekemään. Uusia script-tageja ei näin ollen tarvitse kirjoittaa.

4.1 Sovelluksen pohja

Sovellus koostuu yksinkertaisimmillaan kolmesta osasta: Yleisten muuttujien määrittelystä, asetuksien määrittelyfunktiosta sekä animaatiofunktiosta. Funktioiden nimeksi voidaan asettaa mitä hyvänsä, mutta asetusten määrittelyfunktion nimeksi on yleisesti käytetty init ja animaatiofunktion nimeksi animate. 3D-objektit kannattaa määritellä valmiiksi asetuksien määrittelyfunktiossa, koska se ajetaan vain kerran ja animaatiofunktio ajetaan 60 kertaa sekunnissa. Kuvassa 3 on esitelty sovelluksen tarvitsemat valmistelut aiemmin määriteltyyn script-tagiin.

```
<script>
    /* Yleisten muuttujien määrittely */
    var camera, scene, renderer;

    /* Nämä kaksi funktiokutsua aloittavat sovelluksen ajamisen */
    init();
    animate();

    function init() {
        /* Täällä määritellää sovelluksen asetukset */
    } // init

    function animate() {
        /* Sovelluksen toiminnallisuudet voidaan kirjoittaa tänne */
    } //animate
</script>
```

KUVA 3. Three.js-sovelluksen valmistelut

Three.js-sovellus tarvitsee vähintään kolme elementtiä: Scene, camera ja renderer. Nämä elementit ovat kaikki Three.js-apukirjaston olioita ja ne määritellään funktiossa init. Scene on olio, jolle lisätään kaikki 3D-objektit. Camera-olio on nimensä mukaisesti kamera, jonka kautta käyttäjä näkee scene-oliolle määritellyt objektit. Kameralle kerrotaan sen näkyvyyden leveys, kuvasuhde, lähimmän kamerassa näkyvän objektin etäisyys ja kaukaisimman kamerassa näkyvän objektin etäisyys. Oletuksena kamera sijaitsee scene-olion asettamalla koordinaatistolla keskellä ja se katsoo myös koordinaatiston keskelle, joten sitä on syytä siirtää heti. Muuten kamerassa ei välttämättä näy yhtään objektia. Renderer-olio piirtää kaiken scene-olion koordinaatistolla olevat objektit näkyviin. Renderer luo määritellyn kokoisen canvas-elementin HTML-dokumenttiin ja riippuen määrityksistä Three.js kykenee toimimaan 2D-kontekstilla tai WebGL-kontekstilla. WebGL-kontekstilla toimiva renderer on

kuitenkin huomattavasti tehokkaampi, joka tarkoittaa sitä, että sovellus toimii nopeammin. (Let's Make a 3D Game: Supporting Mobile 2011.) Renderer-olion luoma canvas-elementti on erikseen määriteltävä kuuluvaksi HTML-dokumenttiin.

```
function init() {
    var leveys = window.innerWidth;
    var korkeus = window.innerHeight;

    scene = new THREE.Scene();

    camera = new THREE.PerspectiveCamera( 75, leveys / korkeus, 1, 10000 );
    camera.position.set(0,500,-1000);

    renderer = new THREE.WebGLRenderer({antialias: true,clearColor: 0xf0f0f0});
    renderer.setSize( leveys, korkeus );

    document.body.appendChild( renderer.domElement );
} // init
```

KUVA 4. Init-funktion määrittelyt

Kuvasta 4 nähdään yksinkertaiset sovelluksen määrittelyt. Leveys ja korkeus voidaan ilmoittaa myös pikseleiden mukaan itse, mutta nyt sovellus ottaa tiedot käyttäjän selaimen koosta. Leveyttä ja korkeutta tarvitaan kuvasuhteen ja renderer-olion canvas-elementin koon määrittelyssä. Kameran näkyvyyden leveydeksi on asetettu 75 astetta, kuvasuhde leveydestä ja korkeudesta, lähin objekti yhden yksikön päähän ja kauimmainen objekti 10 000 yksikön päähän. Kameran paikkaa on myös muutettu korkeusakselilla eli y-akselilla 500 yksikköä ja syvyysakselilla eli z-akselilla 1000 yksikköä taaksepäin. Leveysakselilla eli x-akselilla arvo on 0 ja tarkoittaa, että kamera on sillä akselilla keskellä. Renderer-oliolle kerrotaan alustuksessa, että reunanpehmennys (antialias) halutaan käyttöön ja taustan väriksi (clearColor) asetetaan vaalean harmaa. Ilman taustavärin määrittelyä canvas on läpinäkyvä. Canvasin leveys ja korkeus on kerrottu rendererin setSize-funktiolla. Asetusten lopuksi canvas-elementti liitetään dokumenttiin. Canvas voidaan myös liittää esimerkiksi erilliseen HTML-elementtiin, kuten div-elementti ja tyylittää se pienemmäksi ja keskelle sivustoa.

```
function animate() {
    requestAnimationFrame( animate );

    /*  Sovelluksen toiminnallinen sisältö tulee tänne */

    renderer.render( scene, camera );
} //animate
```

KUVA 5. Animate-funktion määrittely

Animate-funktio tarvitsee vähimmillään kaksi asiaa. Toinen on uuden kuvan pyytäminen (requestAnimationFrame) ja toinen on kuvan piirtäminen renderer-olion avulla, joka käyttää scene ja camera -olioita. Yleisesti on ollut tapana, että kaikki toiminnallisuus tulee näiden kahden funktion väliin, kuten kuvasta 5 voidaan nähdä. RequestAnimationFrame on HTML5-tekniikan mukana tullut funktio, jolle on jokaiselle selaimelle oma syntaksinsa. (Window.requestAnimationFrame 2013.) Three.js-kirjaston oma, samanniminen, funktio vähentää kehittäjän työtä, koska se valitsee oikean syntaksin selaimen mukaan ja käyttää näin aina oikeaa HTML5-tekniikan funktiota selaimesta riippumatta. HTML5-tekniikan requestAnimationFrame-funktio pyytää selainta tyhjentämään ikkunan uutta piirtoa varten ja uusi piirto voi tapahtua maksimissaan 60 kertaa sekunnissa. (Window.requestAnimationFrame 2013.)

Näillä asetuksilla sovellus on jo toimiva, mutta se näyttää vain vaalean harmaata tyhjää ruutua, koska scene-oliolle ei ole määritelty yhtään objektia piirrettäväksi. Toimivuuden voi tarkistaa avaamalla selaimen Javascript konsolin ja jos siellä on Three.js merkintä, on sovellus toimiva. Näiden yleisten asetusten määrittämisen jälkeen kehittäjillä on vapaat kädet sovelluksen tulevaisuudesta ja siitä johtuen yleiset tavat loppuvat tähän. 3D-objektit ja niihin liittyvät osat voidaan määritellä omiin muuttujiin init-funktiossa, lisätä yleisesti määriteltyihin taulukoihin tai antaa olion määritellä ne. Tärkeintä on muistaa, että objekteihin päästään käsiksi animate-funktiossa vain, jos ne on määritelty yleisesti.

4.2 Perusmuodot

Seuraavaksi luodaan perusmuotoja, jotka ovat määritelty vain init-funktiossa. Näitä perusmuotoja (geometry) ovat esimerkiksi kuutio (cube), pallo (sphere) ja taso

(plane). Muotojen lisäksi luodaan materiaalit, jotka voivat olla pelkkiä värejä tai päällystetty kuvatekstuurilla.

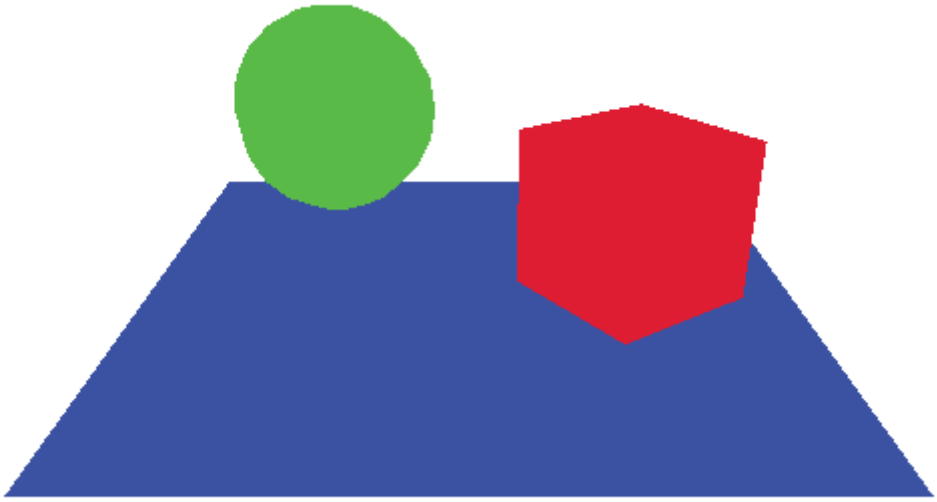
```
document.body.style.margin = "0px";
document.body.style.overflow = "hidden";

camera.lookAt(scene.position);

var lattia_materiaali = new THREE.MeshBasicMaterial( { color: 0x1B32E0,
                        wireframe: false,side:THREE.DoubleSide } );
var lattia_muoto = new THREE.PlaneGeometry(800,800);
var lattia = new THREE.Mesh(lattia_muoto,lattia_materiaali);
lattia.rotation.x = 90 * (Math.PI/180);
scene.add(lattia);

var laatikko_materiaali = new THREE.MeshBasicMaterial( { color: 0xE01B32 } );
var laatikko_muoto = new THREE.CubeGeometry(200,200,200);
var laatikko = new THREE.Mesh(laatikko_muoto,laatikko_materiaali);
laatikko.rotation.y = 35 * (Math.PI/180);
laatikko.position.x = -200;
laatikko.position.y = 100;
scene.add(laatikko);

var pallo_materiaali = new THREE.MeshBasicMaterial( { color: 0x32E01B } );
var pallo_muoto = new THREE.SphereGeometry(150,10,10);
var pallo = new THREE.Mesh(pallo_muoto,pallo_materiaali);
pallo.position.set(200,150,300);
scene.add(pallo);
```

KUVA 6. Tason, kuution ja pallon luonti init-funktiossa

Kuvasta 6 voidaan nähdä, kuinka eri muotoja luodaan ja kuinka niistä tehdään 3D-objekteja. Kuvasta käy ilmi myös, miten Javascriptin avulla dokumentin body-elementin marginaali nollataan ja ylimenevät elementit piiloitetaan. Tämä vähentää sivun turhaa venymistä ja vierityspalkkien ilmestymistä. Kameraobjekti on lukittu myös katsomaan scene-olion keskikohtaan.

Objektille luodaan materiaali ja muoto. Materiaalille voidaan antaa väri HEX-koodina, määrittää näytetäänkö objekti rautalankamallina (wireframe), määrittää näkyykö materiaali myös objektin toisella puolella (side: THREE.DoubleSide) sekä paljon muuta. Eri muodot täytyy määritellä hieman eri tavoilla. Tasolle määritellään leveys ja pituus, kuutiolle leveys, pituus ja korkeus ja pallolle säde. Pallolle on tässä tapauksessa myös määritelty segmenttien leveys ja korkeus, jotka eivät ole pakollisia määreitä. Objekti (mesh) luodaan lopulta yhdistämällä muoto ja materiaali. Objekti asettuu 3D-maailmaan scenen akselien keskelle oman keskikohtansa mukaan, joten palloa on syytä siirtää oman säteensä verran ylös ja kuutiota puolikkaan sivun verran ylös, jotta ne näkyvät maailmassa kokonaan. Kuvasta voidaan nähdä kaksi erilaista

tapaa liikuttaa objekteja maailmassa ja nämä tavat toimivat kaikille 3D-objekteille, myös kameralle. Objekteille voidaan myös ilmoittaa kiertoa (rotation). Kierto ilmoitetaan radiaaneina, mutta asteet saadaan radiaaneiksi kertomalla astemäärä piillä, joka on jaettu 180 asteella. Lattiaa on kierretty x-akselin mukaan 90 astetta ja kuutiota on kierretty y-akselin mukaan 35 astetta. Objekti tulee käyttöön vasta, kun se on lisätty scene-olioon add-funktion avulla. Objektin voi myös poistaa käyttäen remove-funktiota.



KUVA 7. Sovelluksen määrittelyjen tulos tähän mennessä

Tähän mennessä sovellus piirtää vain staattisen kuvan. Animate-funktiosta voitaisiin poistaa requestAnimationFrame-funktion kutsu, jos tarkoituksena on piirtää 3D-objekteja vain kerran, eikä niitä tarvitsisi liikuttaa. Ruudun päivitys olisi staattisessa tilanteessa turhaa resurssien kulutusta.

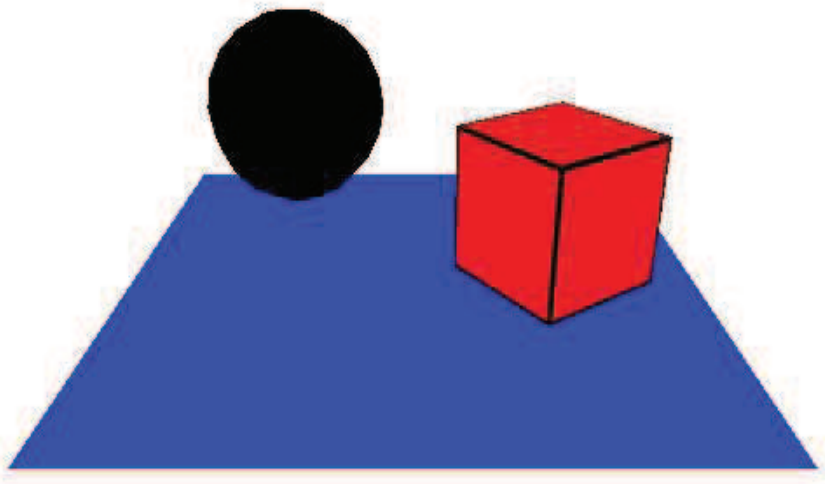
4.3 Tekstuurit ja valaistus

Objekteille voidaan määritellä tekstuuriksi myös kuva. Ensin on varmistettava, että tekstuuriksi käytettävä kuva on neliö, jonka sivut ovat luvun kaksi potensseja. Muuten tuloksena voi olla tekstuureiden venymistä tai toistumisen katkeamista. Yksinkertaisten objektien teksturointiin riittää normaali THREE.MeshBasicMaterial, mutta monipuolisempaan varjostukseen on tarjolla THREE.MeshPhongMaterial, joka perustuu Phong-varjostukseen. Phong-varjostuksen kohdalla on kuitenkin muistettava, että se on raskaampaa tietokoneelle.

```
var punainen = THREE.ImageUtils.loadTexture("pic/red.jpg");
punainen.wrapS = punainen.wrapT = THREE.RepeatWrapping;
punainen.repeat.set(1, 1);

var laatikko_materiaali = new THREE.MeshBasicMaterial( { map: punainen } );
var laatikko_muoto = new THREE.CubeGeometry(200,200,200);
var laatikko = new THREE.Mesh(laatikko_muoto,laatikko_materiaali);
laatikko.rotation.y = 35 * (Math.PI/180);
laatikko.position.x = -200;
laatikko.position.y = 100;
scene.add(laatikko);

var vihrea = THREE.ImageUtils.loadTexture("pic/green.jpg");
vihrea.wrapS = vihrea.wrapT = THREE.RepeatWrapping;
vihrea.repeat.set(2, 2);

var pallo_materiaali = new THREE.MeshPhongMaterial( { map: vihrea } );
var pallo_muoto = new THREE.SphereGeometry(150,10,10);
var pallo = new THREE.Mesh(pallo_muoto,pallo_materiaali);
pallo.position.set(200,150,300);
pallo.rotation.y = 75 * (Math.PI/180);
scene.add(pallo);
```

KUVA 8. Pallon ja laatikon kuvatekstuurien määrittely

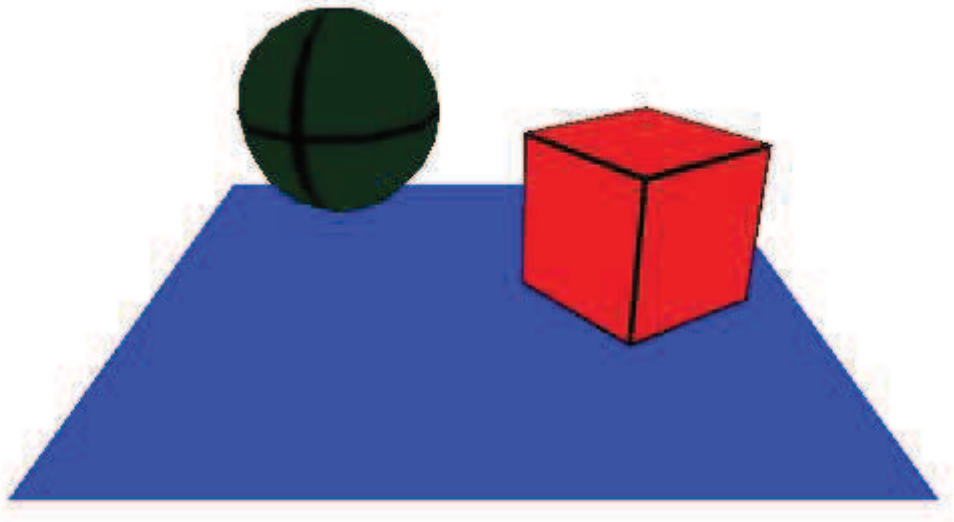
Pallolle ja laatikolle on määritelty tekstuureiksi kuvat. Kuvat ensin ladataan three.js-rajapinnan kuvatyökaluilla, määritellään tekstuuri koko objektin kietovaksi ja lopuksi määritellään kuinka monta kertaa se toistetaan objektissa. Aiemmin materiaalille oli määritelty väri, mutta sen sijaan sille on nyt määritelty kuvakartta eli map. Jos kuvan lisäksi määritellään color, muuttaa se myös tekstuuriksi käytetyn kuvan väriä. Pallolle on määritelty materiaaliksi Phong-varjostus, mutta ei mitään muita asetuksia. Tässä tapauksessa Phong-varjostuksen käyttäytyminen näytetään vain esimerkkinä.



KUVA 9. Laatikon tekstuuri näkyy, mutta pallon tekstuuri ei näy

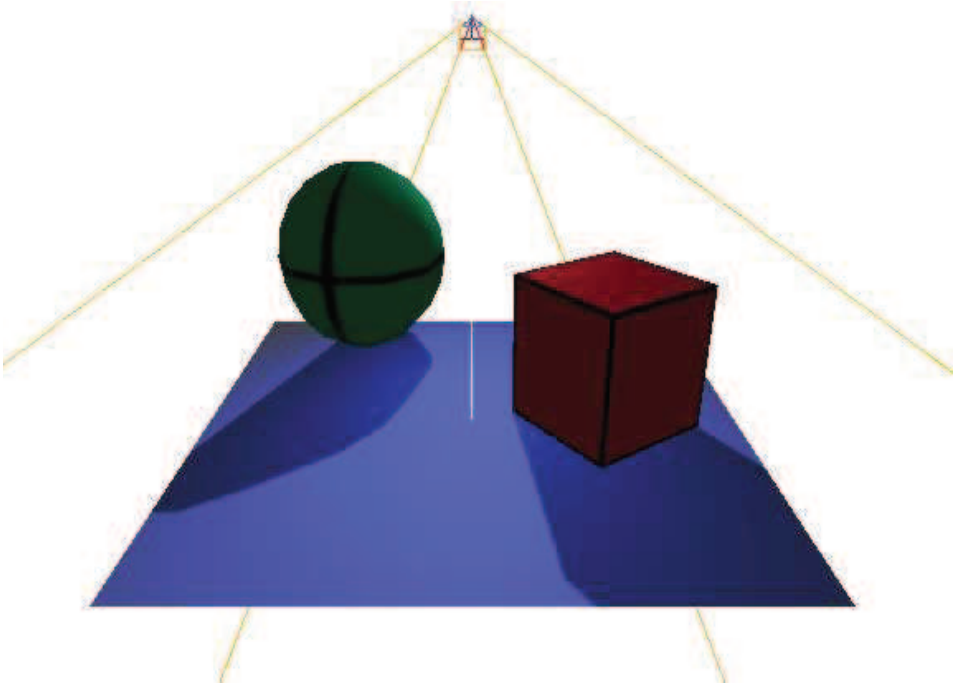
Kuvasta 9 voidaan huomata, ettei pallon Phong-varjostusta käyttävä tekstuuri näy. Phong-varjostus vaatii valoa, jota ei ole vielä dokumenttiin määritelty. Jos halutaan käyttää Phong-varjostuksen ominaisuuksia, on myös valo määriteltävä. Erilaisia valovaihtoehtoja on paljon, mutta yleisimmät ovat AmbientLight, SpotLight ja DirectionalLight. Ensimmäinen tuottaa valoa joka puolelle tasaisesti, toinen ja kolmas loistavat valoa tiettyyn kohteeseen, mutta käyttäytyvät hieman eri tavoilla. Seuraavaksi kuitenkin määritellään vain yleisvalo AmbientLight.

```
var light = new THREE.AmbientLight( 0x404040 );
scene.add( light );
```



KUVA 10. Yleisvalo paljastaa myös pallon kuvatekstuurin

Yleisvalo ei tuota objekteille varjoja. Varjon muodostusta varten on tehtävä esimerkiksi SpotLight-valo, jolle määritellään valon lähtösijainti, kohdepisteen sijainti ja lupa tehdä varjoja. SpotLight-olion määrittelyssä täytyy määrittää valon väri ja voimakkuus. Objekteille täytyy myös erikseen määrittää lupa ottaa vastaan varjoja ja lupa antaa tehdä varjoja. Renderer-oliolle on myös ilmoitettava, että varjokartta on otettava käyttöön piirtämisessä.



KUVA 11. SpotLight ja varjot

Kuvassa 11 on esimerkki SpotLight-oliosta, joka on määritelty visualisoimaan kuinka valo lähetetään. Kaikille materiaaleille on määritelty sama Phong-varjostusta käyttävä materiaali, sekä lupa lähettää ja vastaanottaa varjoja. SpotLight-olio toimii kamera-olion tavoin, koska sillä on aina sijaintipiste ja piste, johon se on kohdistettu. Kuvassa 12 nähdään, kuinka kaikki tämä tehdään koodissa. Renderer-oliolle on ensin määriteltävä varjokartan käyttöönotto, jotta vältytään mahdollisilta virhetilanteilta. SpotLight-oliolle, tässä tapauksessa muuttuja nimeltä light2, on määritelty .shadowCameraVisible arvoksi true. Tämä auttaa valon määrittelemistä visualisoimalla sen lähettämistä säteillä. Arvo on oletuksena false ja sen voi jättää kokonaan määrittelemättä dokumentissa. On mietittävä tarkasti mille objekteille annetaan lupa ottaa vastaan ja lähettää varjoja sovelluksen suorituskyvyn takia. Tässä tapauksessa lattian päällä olevat objektit saavat lähettää varjoja, mutta eivät saa ottaa niitä vastaan. Lattia saa ottaa varjot vastaan, mutta sen on turha lähettää varjoa.

```
renderer.shadowMapEnabled = true;

var light2  = new THREE.SpotLight( 0x8888FF, 2 );
light2.position.set(0,500,600);
light2.target.position.set( 0,2,0 );
light2.castShadow = true;
light2.shadowCameraVisible = true;
scene.add( light2 );

laatikko.castShadow = true;
laatikko.receiveShadow  = false;
pallo.castShadow = true;
pallo.receiveShadow  = false;
lattia.castShadow = false;
lattia.receiveShadow  = true;
```

KUVA 12. Valojen ja varjojen määrittelyt

Jos dokumenttiin tulee paljon objekteja, kannattaa harkita samankaltaisen objektien määrittelemistä taulukoihin ja hoitaa määrittelyt esimerkiksi for-silmukoiden sisällä. Toinen vaihtoehto on luoda omia olioita, joissa on valmiiksi kaikki tarvittavat määrittelyt tehty ja pelkkä olion luonti riittää. Olioissa on myös se etu, että voidaan monipuolistaa objektien ominaisuuksia esimerkiksi liikuttamiseen tarkoitetuilla funktioilla tai lisätä objektien sisälle tai rinnalle toisia 3D-objekteja tai 3D-malleja.

4.4 Olion luonti ja ohjaus näppäimistöllä

Olioilla voidaan helpottaa ohjelmoijan työtaakkaa varsinkin, jos on tarkoitus toistaa paljon objekteja ja käsitellä niitä yksilöinä. Javascript-kielen oliot ovat hyvin yksinkertaistettuja. Niitä ei määritellä muiden kielien tavoin luokan sisälle, mutta tapa on silti hyvin samankaltainen. Javascriptin tapauksessa oliosta luodaan tavallaan vain luokan rakentajafunktio ja siihen voidaan yhdistää muita irrallisia funktioita. Javascript-kielessä on monia erilaisia tapoja luoda olioita, mutta seuraavaksi esittelen tavan, jonka olen itse omaksunut.

```
<script src="js/three.js"></script>
<script src="js/THREEx.KeyboardState.js"></script>
```

```
/* Yleisten muuttujien määrittely */
var camera, scene, renderer
var laatikko; // oliolle varattu muuttuja
var clock = new THREE.Clock; // tulee three.js mukana
var keyboard = new THREEx.KeyboardState(); // erillinen kirjasto
```

KUVA 13. Valmistelut oliota ja ohjausta varten

Kuvassa 13 on esitelty tarvittavat valmistelut oliota ja olion luoman 3D-objektin ohjausta varten. Ensin on head-tagien sisälle lisätty erillinen apukirjasto, joka sisältää näppäimistön käsittelyn. Kirjasto on tarkoitettu lisäosaksi Three.js-apukirjastoon, mutta se toimii myös muussa ympäristössä. Normaalisti näppäimistön käsittelyn ohjelmointi voi tuoda ongelmia nopeissa pelitilanteissa, mutta tämä apukirjasto mahdollistaa esimerkiksi monen näppäimen käsittelyn yhtäaikaa. Tämän lisäksi yleisten muuttujien joukkoon on esitelty muuttuja ”laatikko”, jota käytetään oliota varten. Olion on syytä näkyä init-funktion ulkopuolellakin, joten se määritellään yleisissä muuttujissa. Tulevaisuutta varten esitellään myös Three.js mukana tuleva kello ja aiemmin esitellyn kirjaston näppäimistön käsittelijä. Kelloa voidaan käyttää apuna erilaisissa animaatioissa tai vaikkapa pelissä tapahtumien ajoituksessa.

Seuraavaksi muutetaan aiemmin esitelty laatikko olioksi. Yksinkertaisimmillaan kaikki laatikon määrittelyyn tarvittu koodi laitetaan funktion sisälle, joka tässä tapauksessa on nimetty Olioksi. Funktio muuttuu olioksi, kun korvataan var ilmaisut this.-muotoon. Tämän jälkeen kaikkia olion muuttujia ja funktioita pitää olion sisällä kutsua this.-määreellä ja olion ulkopuolella oliolle varatun muuttujan nimellä. Olion koodi kannattaa kirjoittaa ennen kaikkia muita tapahtumia tai kirjoittaa se kokonaan eri tiedostoon, jolloin se esitellään selaimelle dokumentin latautuessa.

```
function Olio(kuva,kuutio) {
this.texture = THREE.ImageUtils.loadTexture(kuva);
this.texture.wrapS = this.texture.wrapT = THREE.RepeatWrapping;
this.texture.repeat.set(1, 1);

this.elem_materiaali = new THREE.MeshPhongMaterial( { map: this.texture } );
this.elem_muoto = new THREE.CubeGeometry(kuutio,kuutio,kuutio);
this.elem = new THREE.Mesh(this.elem_muoto,this.elem_materiaali);
this.elem.position.y = kuutio/2;
this.elem.castShadow = true;
this.elem.receiveShadow  = false;
scene.add(this.elem);
this.liiku = move;
} //Olio
```

```
    laatikko = new Olio("pic/red.jpg",200);
} // init
```

KUVA 14. Olion määrittely ja kutsu init-funktiossa

Kuvasta 14 nähdään kuinka olioon kuuluvat funktiot määritellään oliolle samalla tavalla kuin tavallinen muuttuja eli ilman funktion sulkuja. Itse funktion tarvitsemat muuttujat määritellään normaalisti funktion määrittelyssä. Tässä tapauksessa oliolle annetaan kutsussa tekstuurikuvan sijainti ja kuution koko. Oliossa näitä ilmaistaan nimillä kuva ja kuutio ilman this.-määrettä.

Olio sisältää scene-olion add-funktion kutsun, joten oliota voidaan kutsua vasta scene-olion määrittelyn jälkeen. Tässä tapauksessa olio luodaan init-funktion viimeisenä työnä ja tulos on samanlainen kuin aiemmin ennen olion määrittelyä. Olion 3D-objekti on nimetty this.elem-määreellä ja siihen päästään olion funktioissa käsiksi sillä kutsulla. Olion ulkopuolella sitä kutsutaan tässä tapauksessa laatikko.elem-määreellä.

```
function move() {
    if ( keyboard.pressed("up") ) {
        this.elem.translateZ(10);
    }
    if ( keyboard.pressed("down") ) {
        this.elem.translateZ(-10);
    }
    if ( keyboard.pressed("left") ) {
        this.elem.rotation.y -= 5*(Math.PI / 180);
    }
    if ( keyboard.pressed("right") ) {
        this.elem.rotation.y += 5*(Math.PI / 180);
    }
} //move
```

```
function animate() {
    requestAnimationFrame( animate );

    laatikko.liiku();

    renderer.render( scene, camera );
} //animate
```

KUVA 15. Olion liikuttamisfunktio ja sen ajaminen animate-funktiossa

Funktio ”move” kuuluu oliolle ja kuvassa 15 nähdään this.elem-määreitä, jotka viittaavat olion 3D-objektiin. Three.js 3D-objekteilla on liikkumiseen ja animointiin käteviä funktioita, kuten translateZ, translateX ja translateY. Nämä siirtävät objektia sen omassa koordinaatistossaan riippumatta ulkopuolisista kierroista. Olion liikuttamisfunktio kiertää 5 astetta suuntaan tai toiseen ja siirtää kappaletta eteen tai taakse riippuen kierrosta.

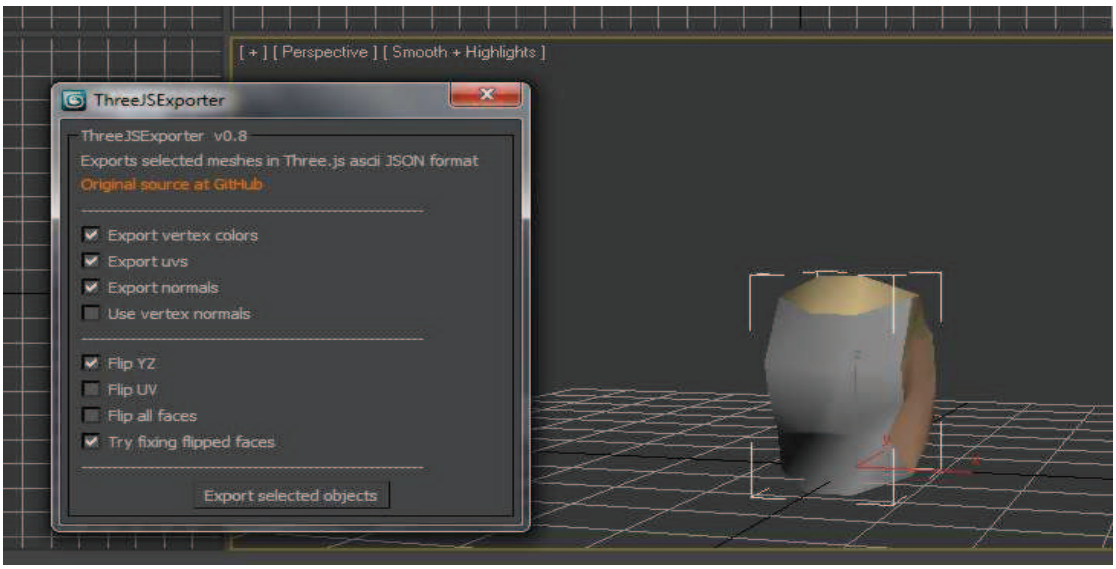
Kuvasta 15 voidaan nähdä myös, että olion liikuttamisfunktiossa on käytetty näppäimistön käsittelijää. Sen syntaksissa nuolinäppäimet voidaan ilmoittaa englanniksi kirjoitettuna ja normaalit kirjaimet pienellä kirjoitettuna. Käsittely toimii myös if-lausekkeessa ilmoitetuilla rinnakkaisehdoilla. Olion liikuttamisfunktiota kutsutaan animate-funktiossa. Tämän pitäisi riittää olion liikuttamiseen näppäimistön avulla.

4.5 3d-mallin tuominen

3D-mallien tuonti Three.js-projektiin voi olla melkoisen mutkikas työ. Itse olen tottunut käyttämään 3ds MAX-ohjelmaa mallintamiseen, mutta suurin osa Three.js-apukirjaston kehittäjistä käyttävät Blender-ohjelmaa. 3ds MAX-ohjelman tuki on nykyään hieman vanhentunut, sillä JSON loader-funktion syntaksi on ilmeisesti

muuttunut versioiden r52 ja r53 välissä. Tähän asti tässä luvussa on käytetty Three.js-apukirjaston versiota r53, mutta JSON loader-funktion syntaksimuutosten takia 3D-mallin tuomiseen joudutaan käyttämään vanhempaa r52-versiota. Aiempi koodi on tähän asti täysin taaksepäin yhteensopivaa, joten muuta lisätyötä muutos ei aiheuta. On hyvä huomioida tämän kaltaiset ongelmat Three.js-apukirjaston kanssa, jos tämän ohjeen tai muun Three.js-ohjeen ratkaisut eivät toimi. Kysyin ratkaisua ongelmaan 3ds MAX-ohjelmasta tuodusta mallista version r53 JSON loader-funktion kanssa Three.js-apufoorumilla ja itse pääkehittäjän mrdoobin vastaus oli vaihtaa ohjelmaa ja malliformaattia.

3D-mallin tuomiseen 3ds MAX-ohjelmasta tarvitaan JSON-formaattiin kääntävä 3ds MAX maxskript-tiedosto, jonka voi löytää joko vanhentuneen Three.js-apukirjaston GitHub-kansioista tai hakemalla nimellä ”ThreeJSExporter”. Maxscript-tiedostot toimivat niin, että 3ds MAX-ohjelmalla valitaan aktiiviseksi käännettävät mallit ja raahataan kansiosta maxscript-tiedosto 3ds MAX-työtilaan. Kääntäjä tekee mallista Three.js oman JSON-muotoisen formaatin mukaisen mallin. Vaihtoehtona voi kokeilla COLLADA- tai OBJ-formaatteja, mutta niiden kanssa voi tulla ongelmia tekstuureiden latauksen ja objektien esityksen kanssa. Google Warehouse-palvelusta voi esimerkiksi ladata ilmaisia SketchUp-ohjelmalla tehtyjä COLLADA-formaatin 3D-malleja, jotka Three.js-projektissa voivat tulla esille ilman tekstuureita ja kopioidut objektit eivät välttämättä näy ollenkaan. Ladatuissa 3D-malleissa ongelmiksi tulevat myös tekstuureina käytettyjen kuvien resoluutio ja objektien raskas polygonien määrä. Jos resoluutio ei ole numeron kaksi potenssi, se ei tule näkymään ollenkaan 3D-mallin kanssa.



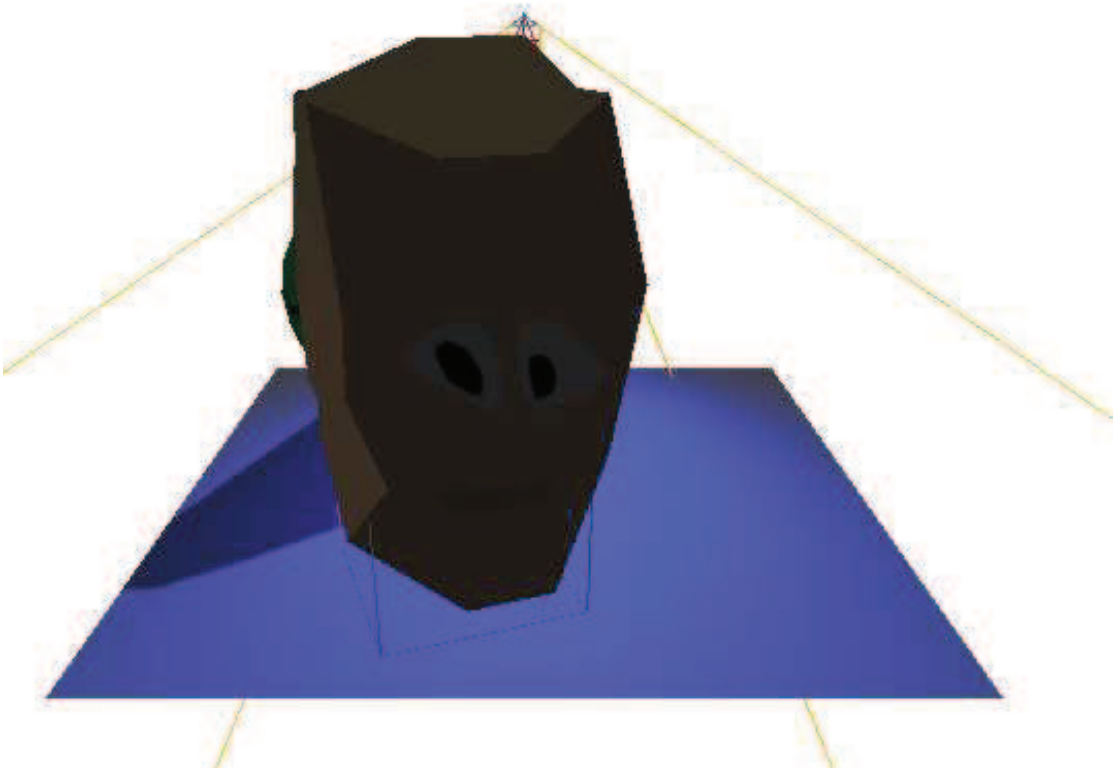
KUVA 16. ThreeJSExporter-kääntäjän suositellut asetukset

Kuvassa 16 nähdään, kuinka tiedosto aukaisee valintaruudun. Kyseisellä mallilla on kolme Phong-materiaalia, joista kahdelle on määritelty vain väri ja yhdellä on kuvatekstuuri. Materiaalina Phong on turvallinen käyttää, sillä se on tuettu myös Three.js-apukirjastossa. Materiaalit on määritelty mallille polygonien mukaan. Ponnahdusikkunasta valitaan halutut asetukset joiden mukaan 3D-malli käännetään JSON-formaattiin. ”Flip YZ”-valinta kannattaa olla päällä, koska Three.js ja 3ds MAX käyttävät eri suuntia ylöspäin suuntaaville vektoreille. ”Flip UV”-valinta kannattaa olla pois päältä, sillä muuten kuvatekstuurit tulevat näkyviin ylösalaisin. Lopuksi kannattaa varmuuden vuoksi laittaa merkki myös viimeiseen kohtaan, sillä jos mallissa on ongelmia, ne toivottavasti korjaantuvat kääntäessä. Koska Three.js-apukirjastolla rakennetaan web-dokumenttia, on otettava huomioon tiedostokoot ja mallien raskaus. Varsinkin pallo- ja sylinteriobjekteissa kannattaa pitää segmenttien määrä hyvin alhaisena, jos tehdään malleja joita piirretään ruudulle paljon kerralla.

```
laatikko = new Olio("pic/red.jpg",200,true);
var loader = new THREE.JSONLoader(),
    callback   = function( geometry ) {

    zmesh = new THREE.Mesh(geometry,new THREE.MeshFaceMaterial());
    zmesh.scale.set(15.5,15.5,15.5);

    laatikko.elem.add(zmesh);
    };
    loader.load("world/malli.js", callback);
```

KUVA 17. Mallin tuominen Three.js-projektiin

3D-malli tuodaan Three.js-projektiin sisäänrakennetun JSONLoader-funktion kautta. Eri malliformaateille on omat latausfunktiot, mutta ne eivät ole sisäänrakennettuja Three.js-apukirjastoon. Kuvassa 17 on jo luotu olio, jonka elementin sisälle tuodaan 3D-malli. Tämä elementti on se, jolle on jo aiemmin luotu näppäimistöohjaus. Kuten muidenkin objektien kanssa, myös 3D-mallin kanssa luodaan THREE.Mesh, jolle määritellään geometria ja materiaali. Tässä tapauksessa geometria tuodaan JSON-tiedostosta ja materiaali täytyy ilmoittaa, että se löytyy samasta tiedostosta. Tätä varten on funktio THREE.MeshFaceMaterial(). Materiaali voidaan myös tässä vaiheessa määritellä uudestaan.



KUVA 18. Näppäimistöohjattavan objektin sisällä on yksinkertainen 3D-malli ihmisen päästä

Kuvassa 18 on valmis malli Three.js-projektissa. Tässä luvussa esiteltyjen tapojen mukaan voidaan jo luoda erilaisia sovelluksia ja pelejä. 3D-mallien taso kannattaa pyrkiä pitämään yksinkertaisena, sillä sovelluksista tulee helposti liian raskaita niiden takia.

5 PELIN RAKENTAMINEN

Tämän opinnäytetyön esimerkkisovelluksena luodaan peli. Pelit vaativat monipuolista osaamista grafiikasta, käyttöliittymistä ja yleisestä tietojenkäsittelystä. Peli on mainio tapa esitellä teknologian mahdollisuuksia. Perinteisemmät sovellukset voisivat hyötyä paljon WebGL-rajapinnan tarjoamasta nopeudesta esimerkiksi tiedon visualisoinneissa ja simulaatioissa.

Pelin lajityypiksi valitsin perinteisen sivuttain kulkevan ammunnan. Esikuvana voidaan pitää esimerkiksi Nokian 2000-luvun alun kännyköistä tuttua Space Impact-peliä. Lisäksi pelityyppi on helposti lähestyttävä yksinkertaisuutensa vuoksi ja laajakuvaruutuihin sopii sivuttain kulkeva toiminta muita paremmin.

5.1 Perusvaatimukset

Tavoitteena on luoda yksinkertainen selaimessa toimiva peli, jota ohjataan hiiren tai näppäimistön avulla. Koska WebGL-teknologian tuki on rajallinen vanhemmilla selaimilla ja mobiililaitteilla, voidaan keskittyä täysin HTML5-ympäristön monipuolisiin ominaisuuksiin ja tietokoneilla tuttuihin ohjaustapoihin. Jos selain ei tue WebGL-rajapintaa, ei se tue todennäköisesti myöskään esimerkiksi websocket-tekniikkaa. Jos käyttäjän selain ei tue pelissä käytettäviä tekniikoita, on siitä näytettävä ilmoitus ja opastettava, kuinka peliä voi päästä pelaamaan. Tässä tapauksessa voidaan näyttää ilmoitus myös WebGL-rajapintaa tukeville mobiililaitteille, koska ohjaus hiirellä tai näppäimistöllä ei sovellu kosketusnäytöille.

Pelin täytyy olla helposti lähestyttävä ja nopeasti opittava. Pelin objektien animoinnista on käytävä ilmi, että peli etenee kohti tiettyä suuntaa, esimerkiksi oikealta vasemmalle. Lisäksi objektien 3D-malleista on tultava ilmi niiden mahdollinen hyöty tai haitta pelaajalle. Pelin vaikeustaso kohoaa pelin kuluessa temmon nostamisella ja maalien lisääntyvällä määrällä. Pelillä täytyy olla selkeä päämäärä ja loppu, joka määräytyy ajan ja pisteiden mukaan. Pelaajalle pitää tulla tunne, että hän voi vaikuttaa pisteiden kasautumiseen toiminnallaan, ja että pisteiden kerääminen on tärkeää.

Pelisovelluksen käytön etenemistä voidaan kuvata käyttötarinalla. Käyttäjä saapuu osoitteeseen, jossa peli sijaitsee, ja klikkaa ruutua lukitakseen hiiren pelialueeseen ja aloittaakseen pelin. Käyttäjä liikuttaa hiirtä ylös ja alas ohjatakseen hahmoaan, joka ampuu ammuksia kohti maaleja. Peli tallentaa käyttäjän tuloksen peliajan loputtua ja näyttää käyttäjälle muiden käyttäjien tulokset. Käyttäjälle annetaan mahdollisuus pelata uusi peli.

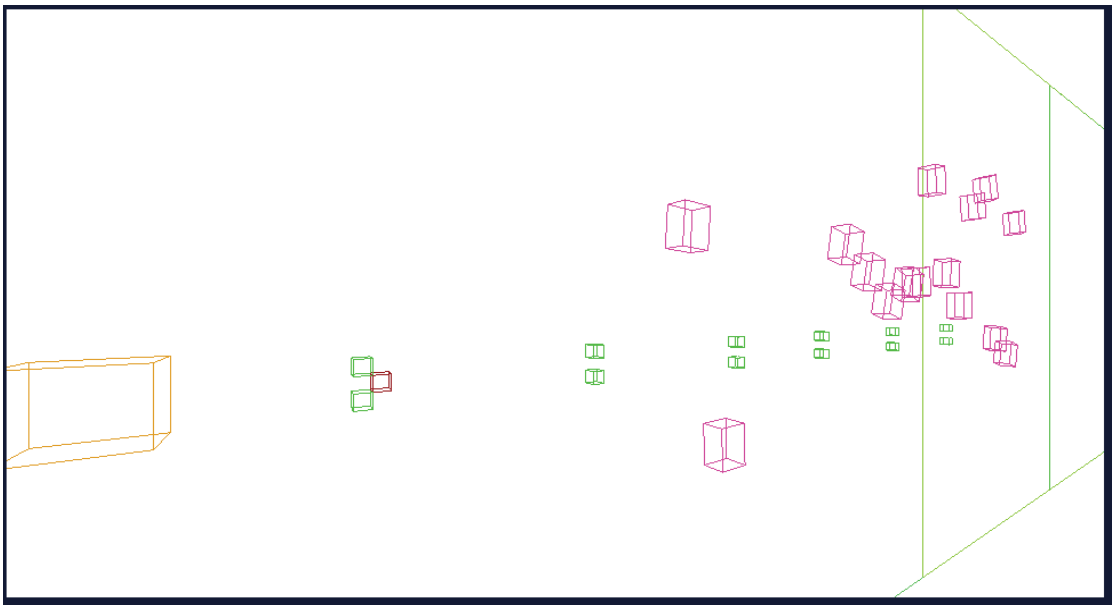
5.2 Pelimoottori

Three.js-apukirjasto on keskittynyt piirtämiseen. Tämän vuoksi kaikki ruudulla tapahtuva animointi ja muut tapahtumat on kehitettävä itse. Kehittäjän on siis hyvä miettiä muitakin vaihtoehtoja. Esimerkiksi Physijs-apukirjaston fysiikkamoottori tekee paljon töitä kehittäjän puolesta, jolloin kehittäjälle jää tehtäväksi vain ympäristöt ja ohjaus. Fysiikkamoottori ei kuitenkaan ole välttämättä hyvä ratkaisu, jos tarkoituksena on luoda maailma, joka ei vastaa oikean elämän luonnonlakeja. Internetissä on myös varmasti apukirjastoja, jotka käsittävät piirtämisen lisäksi ääniä ja vaikkapa websocket-rajapintaa. Toisaalta mitä enemmän on määritelty valmiiksi, sitä enemmän on myös rajoitteita.

Pelimoottorin perusajatuksena on luoda käytännössä valmis pelimekaniikka, joka pyörittää peliä eteenpäin. Pelimekaniikka käsittää pelin säännöt, jotka määrittävät pelin lajityypin ja tekevät sovelluksesta pelin kaltaisen kokemuksen. Peli voi toimia jopa lopullisen tuotteen tavoin, mutta siitä puuttuu maailman täydentävät 3D-mallit. Ilman 3D-malleja on helppo hioa pelin yksityiskohdat ja säännöt kohdilleen. Lisäksi pelimoottoria tehdessä on muistettava, että peli on samalla web-sivu, joten peli ei saa olla liian raskas. Tämä tuo rajoitteita 3D-malleihin, sekä muihin käytettäviin elementteihin esimerkiksi törmäyksen tunnistuksessa.

Pelimekaniikan valmistuessa sijoitetaan 3D-mallit laatikoiden sisään ja laatikot laitetaan näkymättömiksi. Laatikoiden koosta ei tarvitse välittää, koska niitä voidaan säätää 3D-mallien mukaan tai toisinpäin myös pelimekaniikan valmistumisen jälkeen. Laatikoiden mukaan toimii törmäyksen tunnistus, mutta se on ohjelmoitava mukautumaan muutoksiin. Kuvassa 19 nähdään pelaajan ohjaama oranssi laatikko, punainen tähtäin, vihreät ammukset ja ammuttavat kohteet magentana. Taustalla rullaa plane-objekteista koostuva taustaobjekti, johon voidaan myös kiinnittää 3D-malli tai

saumaton kuvatekstuuri. Tausta ei vaadi törmäyksen tunnistusta, joten elementti voi olla yksinkertainen. Valittavan teeman mukaan voidaan maailma muuttaa helposti. Kuvasta 19 nähdään, että kameran asettelulla voidaan saada erilainen vaikutelma maailmasta. Kamera ei ole perinteisesti suoraan sivulta kuvaava, vaan se on hieman pelaajan takana ja suunnattu kohti tulevia kohteita. Tällöin pelaaja jää kameran kuvan reunaan ja näyttäisi kuin se olisi suoraan pelaajan silmien edessä.



KUVA 19. Pelimoottori pyörittää pelin runkoa

Samankaltainen kehitystapa voisi olla jopa suositeltava, vaikka peliä tekisi jossain muussa kehitysympäristössä. Ei ole syytä kiirehtiä maailman suunnittelussa, jos peliä pyörittävä moottori ei ole kunnossa. Lisäksi 3D-mallien mukaan toimiva törmäyksen tunnistus voi olla monimutkainen toteuttaa ja hyvin raskas web-sivuilla pyörivälle sovellukselle.

5.3 Törmäyksen tunnistus

Three.js-apukirjasto ei sisällä minkäänlaista törmäyksen tunnistusjärjestelmää, joten se on kehittäjän tehtävä itse. Jokaiselta 3D-objektilta voidaan saada tietoja sen origon sijainnista, kulmapisteistä ja esimerkiksi seinistä. Näitä tietoja voitaisiin hyödyntää törmäyksen tunnistuksessa. Näistä pisteistä voitaisiin lähettää säde jokaiseen objektiin kerrallaan jokaisella piirtokierroksella ja tutkia, onko kohde tarpeeksi lähellä törmäykseen. Tässä tapauksessa käytän tapaa, jossa tutkitaan kohtaavatko ammuksen

3D-objektin seinä ammuttavan kohteen vastaavaan. Tutkimus käydään läpi jokaisella piirtokerralla.

Ammuttavien kohteiden 3D-objekti lisätään taulukkoon, jota käydään läpi törmäystunnistuksen prosessissa. Lisäämisen jälkeen ne ovat tavallisia 3D-objekteja ruudulla, jotka eivät tarvitse enempää huomiota, koska törmäyksen tunnistus tehdään ammuksissa. Ammus-oliot tutkivat osumista niin kauan kuin ne ovat ruudulla. Osumahetkellä saadaan taulukon avulla yhteys törmättävän kohteen olioon, jonka objektin poistamista kutsutaan. Samalla lisätään tai vähennetään pisteitä ja poistetaan myös ammuksen objekti.

Tällaiseen tunnistukseen on syytä käyttää yksinkertaista objektia, kuten kuutiota. Perinteisemmin törmäyksen tunnistukseen on käytetty sylintereitä ja palloja, mutta ne ovat tähän tapaan turhan raskaita. Tunnistukseen voisi toisaalta käyttää erillistä apukirjastoa, joka toisi mukaan oikeat fysiikat. Fysiikkamoottorin käyttöä täytyy tasapainoitella suorituskyvyn ja tarkoituksenmukaisuuden kannalta.

5.4 Jatkokehitys

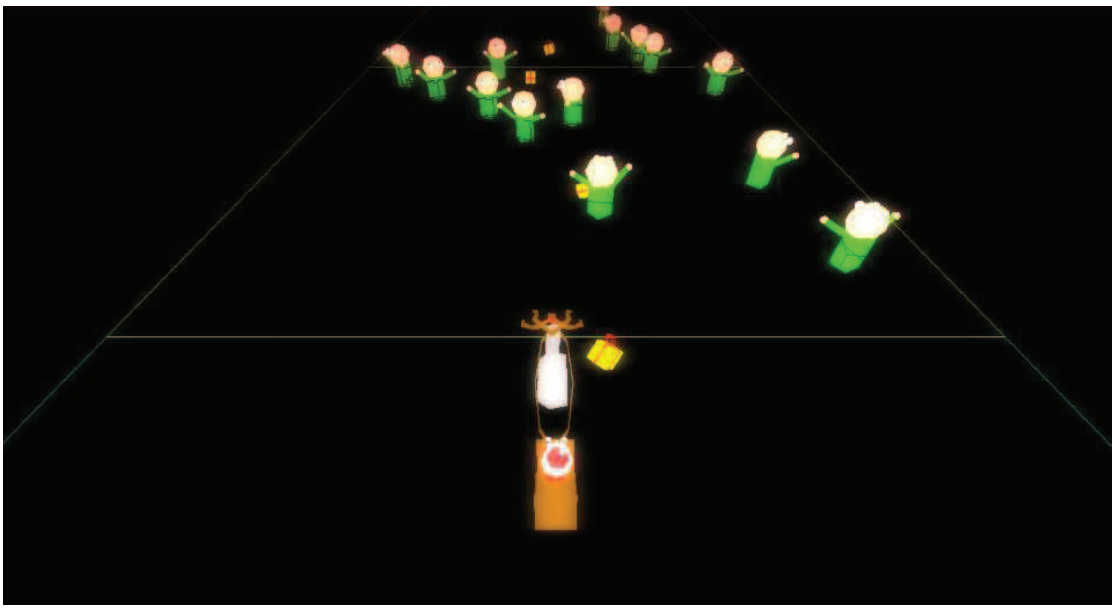
Pelimoottori ja pelin oliot ovat suunniteltu niin, että jatkokehitys ei vaatisi koskemista edes Three.js-apukirjastolla luotuihin elementteihin. Suunnitelmissa on lisätä moninpeli websocket-tekniikan avulla. Moninpelin toteutus olisi käytännössä hyvin yksinkertaista, mutta se vaatisi tietynlaisen palvelimen ja palvelimelle asennettuja ominaisuuksia. Oman palvelimeni rajoitteiden vuoksi jätin moninpelin lopulta pois tästä versiosta, koska halusin pelin olevan pelattavissa heti.

Pelin kehitysprosessin aikana itse Three.js-apukirjasto eteni monta versiota eteenpäin, joista uudemmilla versioilla koitui ongelmia. Seuraava projekti onkin muokata pelimoottori yhteensopivaksi uusimman Three.js-apukirjaston kanssa. Pelimoottori ei tässä versiossa sisällä ollenkaan äänien käsittelyä, mutta vaihtoehtoina on HTML5-tekniikan audio-tagin tai Webaudio-rajapinnan käyttäminen.

5.5 Muut pelimoottorit

Pelimoottorin kehitysprosessissa syntyi muutama sivuprojekti. Tein tutkimustyötä siitä, kuinka Three.js-apukirjasto taipuisi eri pelityyppeihin ilman muita apukirjastoja. Tuloksena syntyi kolme pelimoottoria, joiden kaikkien perustana on samanlainen logiikka törmäysten tunnistukseen. Tarkoituksena oli tehdä yksi esiteltävä pelimoottori, joka voisi olla hyvä kehityksen alusta. Tässä luvussa esitellyt pelimoottorit sisälsivät kuitenkin niin paljon ongelmia, että jouduin lopulta luopumaan niiden kehityksestä.

Ensimmäinen pelimoottori, jonka tein tätä opinnäytetyötä varten on ruudun ylälaidasta alalaitaan etenevä ammunta. Sen logiikka perustuu samalle pohjalle kuin tässä luvussa aiemmin esitelty pelimoottori, mutta sen toteutus on huomattavasti raskaampi. Törmäyksen tapahtuessa lasketaan pisteet ja hävitetään ammus ja ammuttu kohde. Tämä pelimoottori sai kuitenkin teeman ja 3D-mallit objektiensa sisälle. Lisäksi pelissä on käytetty Three.js-apukirjaston postprocessing-renderöintiä, joka korostaa objektien vaaleita värejä saaden ne hohtamaan. Tällaista tehostetta kutsutaan bloom-tehosteeksi ja sitä käytetään usein kaupallisissa peleissä. Tätä pelimoottoria tehdessä huomasin, kuinka pelkistä primitiivimuodoista tehdyt 3D-mallit voivat olla liian raskaita koneille, joissa ei ole erillistä näytönohjainta. Ratkaisuna oli vähentää 3Ds MAX-ohjelmistossa primitiivien segmenttejä reilusti. Bloom-tehosteen ansiosta objektien yksinkertaisuus ei pistä pahasti silmään, kuten kuvasta 20 voidaan todeta. Lisäksi pelimoottori käsitteli ääniä audio-tagin avulla. Äänet kuitenkin pätkivät tällä tavalla, jos niitä yritetään soittaa samaan aikaan useampi.

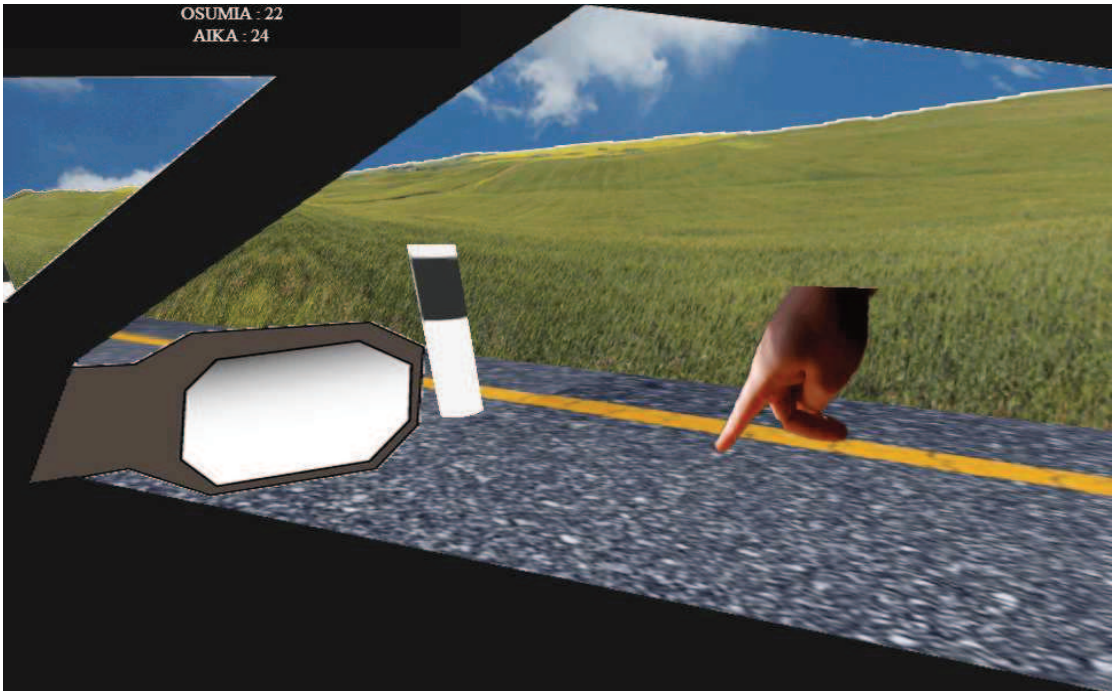


KUVA 20. Ensimmäinen pelimoottori päätyi jouluaiheiseksi peliksi

Toinen pelimoottori on tasoloikkapelin pohja. Olen aina halunnut tehdä oman tasoloikkapelin, joten päätin alkaa työstämään pelin pohjaa. Jälleen pelaaja liikuttaa käytännössä kuutioprimitiivejä, joiden sisälle voidaan asettaa animoitujakin 3D-malleja tai muita peliin tarvittavia elementtejä. Tavoite oli yksinkertaisesti tehdä notkeasti, esimerkiksi tietokannasta, saatujen tietojen mukaan rakennettava maailma ja pelaajaobjekti, joka reagoi tähän maailmaan odotetusti. En aluksi halunnut käyttää ensimmäisessä pelimoottorissa käytettyä törmäyksen tunnistusta, koska en osannut eritellä mistä suunnasta objekti törmää toiseen. Törmäyksen tunnistus perustui aluksi pelaajan koordinaattien ja maailman objektien koordinaattien vertailuun. Törmäys toimi täydellisesti vaakatasossa. Pelaajan on kuitenkin pystyttävä hyppäämään, joten oli tehtävä tunnistus pelaajan tullessa alaspäin ja koordinaattien vertailulla putoaminen pysähtyy. Lisäsin saman logiikan myös pelaajan osuessa yläpuolella olevaan objektiin. Logiikka sai pelaajan välillä jumiin ja välillä pelaaja pääsi objektien yläkulmasta sisälle. Hylkäsin tämän keinon turhauduttuani eri tapahtumiin, joiden olisi pitänyt tapahtua, mutta eivät millään tapahtuneet tai ne tapahtuivat väärin. Jätin tasoloikkapelin rauhaan ja keskityin ammuntapelin pelimoottorin uusimiseen.

Kolmas pelimoottori perustuu yhdelle peli-idealle. Minä ja monet muut ovat varmaankin lapsena autoreissuilla leikkineet juoksevansa sormillaan tienvierusta myöten ja hyppineet liikennemerkkien ylitse. Halusin tehdä tästä WebGL-pelin, joka kulkee ruudun vasemmasta reunasta oikealle. Pelimoottorissa on yksi taustaelementti, joka koostuu tiestä ja tienvieruksesta, jossa liikkuu lähimaisema. Tausta on yksinkertaisesti asetettu CSS-tyylityksillä kuvaksi taivaasta. Esteet ovat yksinkertaisia

liikennemerkkejä, jotka kulkevat eteenpäin samaa vauhtia tien kanssa. Päästessään ruudun ulkopuolelle oikeasta laidasta este siirtyy ruudun ulkopuolelle vasempaan laitaan satunnaiseen paikkaan, jotta esteisiin tulee vaihtelevuutta. Törmäyksen tunnistukseen käytetään samaa tapaa, kuin ensimmäisessä pelimoottorissa. Koska pelaajan objekti koskettaisi kaikilla kahdeksalla seinällään liikennemerkkeihin, täytyy kosketuksia tulkita. Ensimmäisen kosketuksen jälkeen saman objektin seinistä ei välitetä puoleentoista sekuntiin. Pelaajan objekti hyppii vain ylös ja alas ja pelaajan tehtävänä on ajoittaa hypyt, jotta kosketuksia esteisiin ei tule. Peliä varten kuvasin kännykkäkameralla hieman materiaalia sormien juoksusta ja editoin kuvankäsittelyohjelmalla pelkän käden ranteesta irti taustasta. Asetin pelaajan kuutio-objektin sisälle levyobjekteja (plane), joiden tekstuureina ovat animaation kuvat. Pelaajan oliolle on tätä varten käsittelijä, joka piilottaa ja näyttää oikean levyn animaation vaiheen mukaan. Lopuksi viimeistelin pelin lisäämällä HTML-elementin pelin päälle ja tälle taustakuvaksi läpinäkyvän tekstuurin, joka esittää auton etupenkin ikkunaa. Kuvasta 21 nähdään pelin lopputulos, joka kaipaisi vielä viimeistelyä.

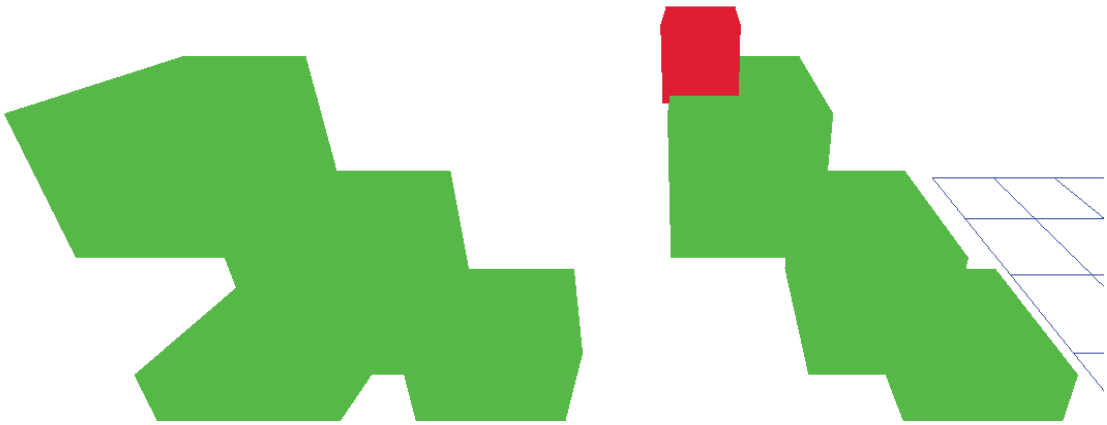


KUVA 21. Nostalginen peli-idea yhden päivän projektina

Tähän mennessä olen päätynyt aina käyttämään ensimmäisestä pelimoottorista tuttua törmäyksen tunnistusta, joten päätin ottaa selvää, kuinka sen eri osat toimivat. Sain selville, että Three.js-apukirjaston olio palauttaa json-objektina tietoja törmättävästä

seinästä. Seinän tiedoista selviää muunmuassa sen normaali, eli mihin suuntaan seinä osoittaa isäntäobjektistaan nähden.

Nostin tasoloikkapelin takaisin työn alle. Törmäyslogiikka on selkeä: Jos pelaaja törmää sen suunnan vastaiseen seinään, eli pelaaja liikkuu seinän normaalin vastaiseen suuntaan, pysäytetään pelaajan eteneminen. Nyt pelin logiikka on huomattavasti yksinkertaisempi myös koodin puolella aiempaan nähden, mutta ei vieläkään täysin aukoton. Törmäyksen kohde saattoi vaihtua sivuseinäksi pelaajan ollessa objektin päällä ja tällöin pelaaja putosi objektin läpi. Korjausten myötä pelaajan ominaisuudeksi tuli vahingossa ninjapeleistä tuttu seinään tarttuminen. Kuvasta 22 voidaan todeta, että pelaajan objekti on hieman taso-objektin sisällä. Tämä johtuu pelaajan objektin vauhdista, jota ei voi keskeyttää tarpeeksi ajoissa tällä tekniikalla.



KUVA 22. Tasoloikkapelin uusi versio hyödyntää muista pelimoottoreista tuttua törmäyksen tunnistusta

Pelimoottorin rakentaminen tyhjästä voi olla vaikeaa, jos ei ole aiempaa kokemusta. Ennen tämän opinnäytetyöprojektin aloittamista olen käyttänyt valmiita pelimoottoreita tai pelinkehitysympäristöjä. Three.js-apukirjastolla tekeminen onkin opettanut minulle paljon pelkästään kokeilujen kautta. En ehtinyt rakentaa 3D-malleja julkaisua varten, koska pelimoottorien rakentaminen oli niin hidasta.

6 PÄÄTÄNTÖ

WebGL-rajapintaa voidaan hyödyntää muillakin tavoilla kuin kolmiuloitteisissa maailmoissa tai esityksissä. WebGL-rajapinta mahdollistaa todella nopean kuvankäsittelyn still-kuville tai vaikka videoille. Tässä opinnäytetyössä keskityttiin tarkoituksella vain 3D-sovelluksiin, mutta tekniikkaa voidaan hyödyntää myös 2D-sovelluksissa. Tutkimusprosessissa en törmännyt kuitenkaan 2D-apukirjastoihin, joten aiheen läpikäyminen olisi vaatinut syvempää tuntemusta itse WebGL-rajapinnasta.

Opinnäytetyötä varten opiskelin pelkästään Three.js-apukirjaston käyttöä yli puoli vuotta. Osaamiseni kirjaston ympärillä on vielä hyvin suppea, sillä perustoimintojen soveltamiseen meni paljon aikaa. Yksi tärkeimmistä oppimisprosesseista oli huomata kirjaston edut ja rajoitteet. Opin paljon uutta asiaa 3D-maailman ohjelmoinnista, sillä Three.js-apukirjastosta puuttuvat monet ominaisuudet joihin olen tottunut muissa ympäristöissä. Tätä varten sain opetella rakentamaan näitä ominaisuuksia itse. Yksi tärkeimmistä peleihin liittyvistä ominaisuuksista on törmäyksen tunnistus. Jatkossa tulen harjoittelemaan lisää Three.js-apukirjaston ja mahdollisesti myös itse WebGL-rajapinnan soveltamista. WebGL-osaaminen antaa varmasti hyvät eväät perinteisemmillä pelinkehitysympäristöillä ohjelmoimiseen, mutta myös WebGL-tekniikalla voi nykyään työllistyä.

WebGL-rajapinnan tulevaisuus näyttää valoisalta. Erilaisia apukirjastoja löytyy moneen eri tarkoitukseen, selainvalmistajat kilpailevat WebGL/Javascript-tulkin nopeudella ja mobiililaitteissa WebGL-rajapinnan tuki on parantunut kokoajan. Monet alan asiantuntijat pitävät WebGL-rajapintaa suurena mahdollisuutena varsinkin mobiililaitteiden puolella tulevaisuudessa. Vielä julkaisematon mobiilikäyttöjärjestelmä Firefox OS tukee HTML5-sovelluksia, kuin ne olisivat natiivisovelluksia. Käyttöjärjestelmä tukee myös WebGL-rajapinnalla luotuja sovelluksia.

WebGL ei tule mullistamaan, kuinka selaamme internetiä, mutta se voi mullistaa kuinka yrityksissä työskennellään jatkossa. Raskaat ohjelmistot voivat siirtyä perinteisiltä pöytäkoneille asennettavilta versioilta selaimeen ja pilveen. Tämä mahdollistaa entistä notkeampaa ja tehokkaampaa työskentelyä. Insinööri- tai

arkkitehtitoimisto voisi esimerkiksi ladata työstämänsä mallin internetpalveluun, josta projektin päällikkö voi sitä esitellä tai tutkia vaikka tablettilaitteella. WebGL voi mullistaa myös ohjelmistokehityksen, sillä tekniikan ansiosta selaimesta ajettava ohjelmisto toimii eri käyttöjärjestelmillä ilman käännöstyötä. WebGL on joka tapauksessa suuri askel eri käyttöjärjestelmien yhteiseen kehitysalustaan, jonka mahdollisuudet ovat rajattomat.

LÄHTEET

10 new HTML5 tags you need to know about 2012. Justin James. WWW-dokumentti. http://www.techrepublic.com/blog/10things/10-new-html5-tags-you-need-to-know-about/3219. Päivitetty 13.4.2012. Luettu 11.1.2013.

ANGLE 2013. Google. WWW-dokumentti. http://code.google.com/p/angleproject/. Päivityksestä ei tietoja. Luettu 4.2.2013.

A re-introduction to JavaScript 2013. Mozilla Developer. WWW-dokumentti. https://developer.mozilla.org/en-US/docs/JavaScript/A_re-introduction_to_JavaScript. Päivitetty 8.1.2013. Luettu 14.1.2013.

Canvas 3D: GL power, web-style 2007. Vladimir Vukicevic. WWW-dokumentti. http://blog.vlad1.com/2007/11/26/canvas-3d-gl-power-web-style/. Päivitetty 26.11.2007. Luettu 14.1.2013.

Compatibility tables for support of HTML5, CSS3, SVG and more in desktop and mobile browsers 2012. Can I use. WWW-dokumentti. http://caniuse.com/#search=canvas. Päivitetty 3.12.2012. Luettu 14.1.2013.

Creating a scene 2013. Mrdoob. WWW-dokumentti. http://mrdoob.github.com/three.js/docs/55/#Manual/Introduction/Creating_a_scene. Päivityksestä ei tietoja. Luettu 29.1.2013.

De opkomst van WebGL 2013. Johan Dorland, Willem den Besten. WWW-dokumentti. http://www.students.science.uu.nl/~3685632/content/history.html. Päivityksestä ei tietoja. Luettu 14.1.2013.

GLSL Explained 2013. Everything Explained. WWW-dokumentti. http://everything.explained.at/GLSL/. Päivityksestä ei tietoja. Luettu 15.1.2013.

HTML Introduction 2013. W3C. WWW-dokumentti. http://www.w3schools.com/html/html_intro.asp. Päivityksestä ei tietoja. Luettu 11.1.2013.

HTML5 Canvas is a go 2010. Peter BeverLoo. WWW-dokumentti. http://peter.sh/tag/history/. Päivitetty 22.6.2010. Luettu 14.1.2013.

HTML5 canvas - the basics 2009. Opera Dev. WWW-dokumentti. http://dev.opera.com/articles/view/html-5-canvas-the-basics/. Päivitetty 8.1.2009. Luettu 14.1.2013.

Internet Explorer Won't Be Supporting WebGL 2011. Softpedia. WWW-dokumentti. http://news.softpedia.com/news/Internet-Explorer-Won-t-Be-Supporting-WebGL-206696.shtml. Päivitetty 7.6.2011. Luettu 18.12.2012.

Javascript How To 2013. W3Schools. WWW-dokumentti. http://www.w3schools.com/js/js_howto.asp. Päivityksestä ei tietoja. Luettu 14.1.2013.

Let’s Make a 3D Game: Supporting Mobile 2011. Learning Three.js. WWW-dokumentti. http://learningthreejs.com/blog/2011/12/28/let-s-make-a-3d-game-supporting-mobile/. Päivitetty 28.12.2011. Luettu 28.3.2013.

Mozilla is Unlocking the Power of the Web as a Platform for Gaming 2013. Mozilla. WWW-dokumentti. https://blog.mozilla.org/blog/2013/03/27/mozilla-is-unlocking-the-power-of-the-web-as-a-platform-for-gaming/. Päivitetty 27.3.2013. Luettu 28.3.2013.

Native Client 2012. Google Developers. WWW-dokumentti. https://developers.google.com/native-client/devguide/coding/3D-graphics. Päivitetty 5.12.2012. Luettu 18.12.2012.

Ohjelmointi 1 2010. Jyväskylän Yliopisto. WWW-dokumentti. http://kurssit.it.jyu.fi/ITKP102/moniste/html/moniste.html. Päivitetty 7.9.2010. Luettu 29.1.2013.

OpenGL ES - The Standard for Embedded Accelerated 3D Graphics 2012. Khronos Group. WWW-dokumentti. http://www.khronos.org/opengles/. Päivityksestä ei tietoja. Luettu 18.12.2012.

Parisi, 2012. WebGL: Up and Running. Sebastopol: O'Reilly Media.

Physijs 2013. Chandlerprall. WWW-dokumentti. http://chandlerprall.github.com/Physijs/. Päivityksestä ei tietoja. Luettu 22.2.2013.

Removed Collision code 2012. Mrdoob. WWW-dokumentti. https://github.com/mrdoob/three.js/commit/6ad7cbb47bd4a0e441c744030a8a0e2e3f2edfd2. Päivitetty tammikuussa 2012. Luettu 29.1.2013.

RFC 6455 - The WebSocket Protocol 2011. Faqs.org. WWW-dokumentti. http://www.faqs.org/rfcs/rfc6455.html. Päivitetty joulukuu 2011. Luettu 14.12.2012.

Tinkercad 2013. Tinkercad. WWW-dokumentti. https://tinkercad.com/. Päivityksestä ei tietoja. Luettu 9.3.2013.

The History of DirectX 2012. Coding Unit. WWW-dokumentti. http://www.codingunit.com/the-history-of-directx. Päivityksestä ei tietoja. Luettu 18.12.2012.

The History of HTML5 2012. Matt Silverman. WWW-dokumentti. http://mashable.com/2012/07/17/history-html5/. Päivitetty 17.7.2012. Luettu 11.1.2013.

The History of OpenGL 2012. Coding Unit. WWW-dokumentti. http://www.codingunit.com/the-history-of-opengl. Päivityksestä ei tietoja. Luettu 18.12.2012.

Three.js contributors 2013. Mrdoob. WWW-dokumentti. https://github.com/mrdoob/three.js/graphs/contributors. Päivitetty 27.1.2013. Luettu 29.1.2013.

WebGL - OpenGL ES 2.0 for the Web 2012. Khronos Group. WWW-dokumentti. http://www.khronos.org/webgl/. Päivityksestä ei tietoja. Luettu 18.12.2012.

Websockets 101 2012. Armin Ronacher. WWW-dokumentti. http://lucumr.pocoo.org/2012/9/24/websockets-101/. Päivitetty 24.9.2012. Luettu 14.12.2012.

Websocket API 2012. Blackberry. WWW-dokumentti. http://docs.blackberry.com/en/developers/deliverables/29271/Web_Sockets_support_1582781_11.jsp. Päivityksestä ei tietoja. Luettu 14.12.2012.

Websockets vs Ajax 2011. Geek's Drafts' blog. WWW-dokumentti. http://www.geeksdrafts.net/blog/2011/01/10/websockets-vs-ajax/. Päivitetty 10.1.2011. Luettu 14.12.2012.

Why you should use OpenGL and not DirectX 2010. Wolfire. WWW-dokumentti. http://blog.wolfire.com/2010/01/Why-you-should-use-OpenGL-and-not-DirectX. Päivitetty 8.1.2010. Luettu 18.12.2012.

Why you should use WebGL 2013. Florian Boesch. WWW-dokumentti. http://codeflow.org/entries/2013/feb/02/why-you-should-use-webgl/. Päivitetty 2.2.2013. Luettu 4.2.2013.

Window.requestAnimationFrame 2013. Mozilla Developer Network. WWW-dokumentti. https://developer.mozilla.org/en-US/docs/DOM/window.requestAnimationFrame. Päivitetty 2.1.2013. Luettu 3.1.2013.