Jani Aaltonen

# Interaktiivinen 3D HTML5-selaimissa

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Mediatekniikan koulutusohjelma

Insinöörityö

7.5.2013

Metropolia

| | |
|---|---|
| Tekijä<br>Otsikko<br><br>Sivumäärä<br>Aika | Jani Aaltonen<br>Interaktiivinen 3D HTML5-selaimissa<br><br>51 sivua<br>7.5.2013 |
| Tutkinto | insinööri (AMK) |
| Koulutusohjelma | mediatekniikka |
| Suuntautumisvaihtoehto | digitaalinen media |
| Ohjaaja | yliopettaja Harri Airaksinen |


Insinöörityön tavoitteena oli tutkia Metropolia Ammattikorkeakoulun mahdollisuuksia tuottaa interaktiivista 3D:tä verkkoselaimiin WebGL:n avulla ja käyttäen ammattikorkeakoulun 3D-mallinnusohjelmaa. WebGL on ohjelmointirajapinta, jolla saadaan luotua 3D-grafiikkaa verkkoselaimeen ilman ylimääräisiä liitännäisiä. Insinöörityö tehtiin Metropolia Ammattikorkeakoululle, ja sen tuloksia käytetään sekä osana opetusta että mahdollisesti 3D-sisällön tuottamiseen ammattikorkeakoulua varten.

Työssä käytettiin kahta WebGL-ohjelmistokehystä, jotka molemmat olivat käyttäjäystävällisiä toteutuksia tuottaa WebGL:ää verkkoselaimiin. Niitä käyttäen luotiin opetusta varten 3D-peli ja lukuisia pieniä esimerkkituotoksia. Lisäksi luotiin prosesseista tarkat ohjeet, joita oppilaat seuraisivat opetuksessa. Molemmat toteutustavat toimivat yksinkertaisilla liitännäisillä oppilaitoksen 3D-mallinnusohjelman kanssa, joten sillä tehdyt 3D-mallinnukset ja -animaatiot saatiin kätevästi eteenpäin aina verkkosivuille asti.

Insinöörityössä tutustuttiin tarkemmin myös WebGL:ää tukevan 3D-ohjelman käyttöön sekä sen ja WebGL-ohjelmistokehyksien julkaisuasetuksiin. Niillä tuotiin onnistuneesti interaktiivista 3D:tä verkkoselaimiin.

Insinöörityön pohjalta Metropolia Ammattikorkeakoulu hankki opetuksen tueksi WebGL:ää tukevan 3D-ohjelman ja asensi 3D-mallinnusohjelmaan tarvittavat liitännäiset. Opetuksessa käytettiin onnistuneesti insinöörityön esimerkkituotoksia apuna tutustuttaessa WebGL:ään ja tuotettaessa omaa sisältöä sen avulla.

WebGL:n tulevaisuus WWW:n standardina 3D:n tuottamisessa verkkoselaimeen näyttää lupaavalta, mutta ongelmia on vielä ratkottavana. Niistä mahdollisesti suurin on saada kaikkien verkkoselaimien tuki WebGL:lle. WebGL tarjoaa kuitenkin kattavan valikoiman toimintoja ja mahdollisuuksia, joiden avulla se on vahvoilla.



| | |
|---|---|
| Avainsanat | WebGL, 3D, HTML5, CopperCube, X3DOM, animaatio, verkkoselain |

| | |
|---|---|
| Author<br>Title<br><br>Number of Pages<br>Date | Jani Aaltonen<br>Interactive 3D in HTML5 browsers<br><br>51 pages<br>7 May 2013 |
| Degree | Bachelor of Engineering |
| Degree Programme | Media Technology |
| Specialisation option | Digital Media |
| Instructor | Harri Airaksinen, Principal Lecturer |

The purpose of this thesis was to investigate the possibilities for Metropolia University of Applied Sciences to create interactive 3D to web browsers by using WebGL and the 3D modeling software used by Metropolia. WebGL is an application programming interface that allows 3D graphics to be brought into the web browser without requiring any additional plugins. This study was made for the Metropolia University of Applied Sciences and its results are being used as part of education and possibly to create 3D content for the University.

Two user-friendly WebGL frameworks were used in this thesis to produce WebGL to the web browsers. Using these frameworks, a 3D game and many small examples were created for educational purposes. Precise step by step instructions were also created for the students to demonstrate the processes for creating the game and examples. Both frameworks worked with simple plugins to the Metropolia's 3D modeling software, so the 3D models and animations created with it were easily brought into the web pages.

This thesis also took a closer look into how a specific 3D software with WebGL support works. Its publishing settings as well as those of the WebGL frameworks' were also studied. They were used to successfully export interactive 3D to the web browsers.

Based on this study, Metropolia University of Applied Sciences purchased the 3D software with WebGL support and installed the required plugins for its existing 3D modeling software. The examples created in this thesis were successfully used for educational purposes to help students get familiar with WebGL and to create their own content with it.

The future of WebGL as the web standard for creating 3D to the web browsers seems promising, although there are still problems to be solved. Possibly the most challenging one is the support for WebGL from all the web browsers. However, WebGL provides a rich variety of features and possibilities, which gives it an edge in the competition.

| | |
|---|---|
| Keywords | WebGL, 3D, HTML5, CopperCube, X3DOM, animation, web browser |

## Sisällys

## Lyhenteet

| | |
|---|---|
| HTML5 | Hypertext Markup Language -merkintäkielen kehitteillä oleva uusi versio. |
| WebGL | Web Graphics Library. Ohjelmointirajapinta, jolla saadaan luotua 3D-grafiikkaa verkkoselaimeen ilman ylimääräisiä liitännäisiä. |
| Canvas | Elementti, joka mahdollistaa grafiikan piirtämisen suoraan verkkoselaimeen erilaisten piirtokäskyjen avulla. |
| JavaScript | Komentosarjakieli, jonka avulla verkkosivuista saadaan dynaamisempia ja interaktiivisempia. |
| DOM | Document Object Model. Ohjelmointirajapinta, jonka avulla verkkosivuja voidaan muokata. |
| GLSL | OpenGL Shading Language. Ohjelmointikieli, jonka avulla grafiikkalaitteiden laskennat suoritetaan. |
| X3D | Extensible 3D Graphics. XML:ään pohjautuva tiedostotyyppi, jonka avulla esitetään 3D-grafiikkaa. |
| XML | Extensible Markup Language. Merkintäkieli, joka säilyttää luettavan tiedon merkityksen muun tiedon keskellä. |
| CopperLicht | WebGL-ohjelmistokehys, joka mahdollistaa WebGL:n käytön 3D-sovellusten tekemiseen ja julkaisemiseen. |
| X3DOM | Avoimen lähdekoodin WebGL-ohjelmistokehys, jonka avulla saadaan esitettyä interaktiivista 3D:tä suoraan HTML5-verkkoselaimessa. |

# 1 Johdanto

Internetin kehitys on jatkuva prosessi, ja yksi sen tämän hetken mielenkiintoisimmista tulevista uudistuksista on HTML5-merkintäkieli, joka lähenee valmista muotoaan. Sen mukana tulee lukuisia uusia ominaisuuksia, ja yksi mielenkiintoisimpia on 3D-grafiikan tulo suoraan mukaan verkkosivuille.

Tämän insinöörityön tavoitteena on tutkia WebGL-teknologiaa interaktiivisen 3D-grafiikan viemiseksi HTML5:tä tukevaan verkkoselaimeen. Työssä tutkitaan eri mahdollisuuksia tämän toteuttamiseksi ja sitä, onko WebGL varteenotettava vaihtoehto tähän tarkoitukseen.

Insinöörityön tilaaja on Metropolia Ammattikorkeakoulu, ja työn tuloksia käytetään osana aiheeseen liittyvän opintojakson opetusmateriaalia. Työn tarkoituksena on selvittää ammattikorkeakoululle sopiva tapa tuottaa interaktiivista 3D:tä verkkoselaimeen käyttäen oppilaitoksen tarjoamia ohjelmia. Metropolia Ammattikorkeakoulu haluaa esimerkkisovelluksen, jossa interaktiivista 3D:tä tuodaan selaimeen 3D-mallinnusohjelmasta. Tätä sovellusta käytetään esimerkkinä opiskelijoille, jotta he pääsevät kosketuksiin teknologian kanssa. Valmistettavan WebGL-sovelluksen päätavoitteet ovat hyvä toimivuus oppilaitoksen laitteilla ja ohjelmilla, yksinkertainen ja selkeä rakenne sekä käyttökelpoisuus oppilaitoksen opetusmateriaalin apuna.

Insinöörityöraportin loppuosassa analysoidaan WebGL:n tilannetta ja saatuja tuloksia sekä luodaan katsaus tulevaisuuteen.

# 2 HTML5-merkintäkieli ja WebGL-ohjelmointirajapinta

## 2.1 HTML5 ja sen canvas-elementti

HTML5 on yleisesti käytetty nimitys uusimmalle kehitteillä olevalle versiolle HTML-merkintäkielestä, ja sen tarkoituksena on tuoda lukuisia uudistuksia ja parannuksia jo olemassa olevaan kuvauskieleen [1, s. 13]. HTML5:n kehityksestä vastaavat World Wide Web Consortium eli W3C ja Web Hypertext Application Technology Working Group eli WHATWG [1, s. 18]. W3C on konsortio, jonka muodostavat useat alan yritykset sekä muut yhteisöt, ja ne kehittävät WWW:n standardeja ja suosituksia [2]. WHATWG puolestaan on Applen, Mozillan ja Operan luoma yhteisö, joka kehittää HTML-merkintäkieltä ja sen ohjelmointirajapintoja [3].

Alun perin WHATWG alkoi työstää kehiteltyä versiota HTML:stä vuonna 2004, mutta se muuntui vasta myöhemmin osaksi HTML5-käsitettä. Myös W3C tuli mukaan HTML5:n kehittämiseen vuonna 2007, mikä johti tiiviimpään kehitykseen ja tarkoitukseen pitää HTML:n ja selainten kehittäminen lähellä toisiaan. [1, s. 24.]

Yhtenä uutena mielenkiintoisena ominaisuutena HTML5 on tuomassa käyttöön canvas-elementin. Se mahdollistaa grafiikan piirtämiseen suoraan verkkoselaimeen erilaisten piirtokäskyjen avulla. Tätä varten sillä on 2D-grafiikan tekemiseen ohjelmointirajapinta, jossa käytetään JavaScriptiä käskyjen antamiseen. [4, s. 51.]

Canvas-elementin toiminta perustuu sen luomaan suorakulmaiseen piirtoalustaan, joka on yksinkertainen tyhjä alue. Alueen koko voidaan määritellä halutun kokoiseksi width- ja height-arvojen avulla. Alueelle piirretään antamalla JavaScript-komentoja. [1, s. 200.]

Piirtämiseen käytettävät JavaScript-komennot kohdistuvat kaikki canvas-elementtiin, joten piirtäminen aloitetaan hakemalla sivulle luotu canvas-elementti sen id:n nimellä. Näin saadaan käsittelyyn piirtämistä varten luotu alue. Lisäksi täytyy vielä määritellä oliopohjainen konteksti, joka tuo käyttöön erilaisia välineitä piirtämistä varten. Näitä ovat esimerkiksi yksinkertaiset 2D-piirroksia varten tarvittavat työvälineet. [1, s. 200.] Tällä tavalla piirrettyjen kuvien etuna on mahdollisuus niiden muokkaamiseen. Jos verkkosivulle tuotaisiin tavallinen kuva tai kuvio, se olisi siinä aina samanlaisena, kunnes toisella ohjelmalla piirretään uusi kuva myöhemmin sitä korvaamaan, oli kuvassa tarvittu muutos miten pieni tahansa. Canvas-elementtiin piirretty kuva

puolestaan on mahdollista päivittää samantien hyvin pienellä muutoksella JavaScriptiin tai antamalla sille jokin uusi JavaSript-käsky. [1, s. 201.]

JavaScriptillä tehtävät piirtokäskyt antavat myös mahdollisuuden piirtää kuvia dynaamisesti. Tämä tarkoittaa sitä, että piirrettävä kuva voi saada arvonsa esimerkiksi käyttäjän sivulla sille antamista arvoista. Kun arvoja muutetaan, piirtäminen tehdään uudestaan uusilla arvoilla, mikä johtaa kuvan muuttumiseen reaaliajassa. [1, s. 201.] Tämä ominaisuus on canvas-elementin ehdoton vahvuus ja antaa mahdollisuuden tehdä monia asioita, esimerkiksi muuttuvia kuvaajia, jotka perustuvat tietokannan muuttuvaan dataan, täysimittaisia animaatioita ja jopa graafisia 2D- ja 3D-pelejä.

## 2.2 WebGL-ohjelmointirajapinta

WebGL on ohjelmointirajapinta, jolla saadaan luotua 3D-grafiikkaa verkkoselaimeen ilman ylimääräisiä liitännäisiä. Se toimii suoraan useimmissa tunnetuissa verkkoselaimissa ja käyttää toimiakseen laitteiden omaa grafiikkalaitteistoa, johon kuuluvat sekä tietokoneet että älypuhelimet ja kaikki niiden välistä, kunhan laitteessa on tarvittava grafiikkalaitteisto ja WebGL:ää tukeva verkkoselain. [5.]

WebGL toimii HTML5:n canvas-elementissä ja piirtää siihen kaiken halutun 2D- ja 3D-grafiikan. Tämän vuoksi sillä on myös käytössä kaikki Document Object Model -ohjelmointirajapinnan ominaisuudet, esimerkiksi sen tuetut ohjelmointikielet, joihin kuuluvat JavaScript, Java ja mahdollisuus myös Objective C:hen. [6.] Document Object Model -ohjelmointirajapinta mahdollistaa myös verkkosivujen rakenteen dynaamisen muokkauksen komentojen ja skriptien avulla, minkä ansiosta verkkosivuihin saadaan lisättyä toiminnallisuutta ja interaktiivisuutta [7.]

WebGL on matalan tason ohjelmointirajapinta, mikä tarkoittaa, että se on läheisissä tekemisissä konekielen kanssa [5]. Tämän takia sillä päästäänkin paremmin hyödyntämään näytönohjaimen ja ohjelmiston välisiä ominaisuuksia ja hienosäätöjä ja siksi on myös melko vaativa käyttää ja tarvitsee ison määrän koodia jo yksinkertaisenkin toiminnon toteuttamiseksi [5; 6].

Käyttökynnyksen madaltamiseksi WebGL:lle on luotu useita JavaScript-kirjastoja, jotka tarjoavat melko kattavan määrän WebGL:n eri tomintoja ilman, että joudutaan lainkaan kosketuksiin matalan tason ohjelmointikielen kanssa. [8.]

WebGL perustuu OpenGL ES 2.0:aan, joka on myös voittoa tavoittelemattoman Khronos Groupin hallitsema konsortio [5]. Se on alustariippumaton, mikä tarkoittaa, etteivät siinä olevat ominaisuudet ole riippuvaisia mistään tietyistä laitteista tai käyttöjärjestelmistä [6]. Sen ansiosta WebGL myös toimii monilla eri alustoilla. Näitä alustoja ovat muun muassa pöytätietokoneet, älypuhelimet, televisiot ja nykyaikaiset mobiililaitteet [8].

WebGL käyttää OpenGL Shading Language (GLSL) -värivarjostusta, jonka avulla grafiikkalaitteiden laskennat suoritetaan. Sen avulla sivusta voidaan myös tehdä nopeampi, koska sivulla muuten käytössä oleva JavaScript saattaa hidastua, jos se joutuu suorittamaan vaativampia laskutoimituksia. Sen sijaan, että annettaisiin JavaScriptin hoitaa WebGL:n tarvitsemia toimituksia, pystyy näytönohjain laskemaan ne huomattavasti nopeammin, koska GLSL suoritetaan suoraan näytönohjaimessa. [5.]

WebGL:n piirtämät 3D-mallit muodostuvat kolmioista. Niiden tuottamisprosessi alkaa, kun JavaScriptillä luodaan aluksi verteksitaulukot, joihin kerätään tärkeitä tietoja jokaisesta verteksistä, kuten sen sijainti 3D-maailmassa, tekstuuri, väri ja mahdollisen valaistuksen vaikuttaminen. Nämä tiedot voidaan saada joko WebGL:ään ladattavasta erillisestä 3D-tiedostosta, esimerkiksi .X3D-tiedostosta tai WebGL-ohjelmistokehyksien tarjoamista geometrisistä muodoista tai luomalla se aivan tyhjästä. [8.]

Tämän prosessin jälkeen kerätty tieto lähetetään grafiikkaprosessorille, jossa se syötetään verteksipuskuriin. Tämän jälkeen grafiikkaprosessori alkaa lukea tietoa, minkä jälkeen se ajaa sen verteksivarjostimen läpi. Verteksivarjostin puolestaan käsittelee tietyn määrän verteksiattribuutteja kerrallaan ja tekee niistä uuden joukon attribuutteja. Tässä prosessissa varjostin laskee niille uuden sijainnin näytölle ja niiden mahdolliset värit ja tekstuurit. [8.]

Seuraavaksi grafiikkaprosessori tekee vertekseistä kolmioita, joita se kokoaa yhteen ja lähettää sitten rasteroijaan. Rasteroija tekee kolmioista pikselin kokoisia fragmentteja, joista kaikista muodostuu yhtenäinen pinta. Tämän jälkeen ne ajetaan fragmenttivarjostimen läpi, joka määrittää jokaiselle pikselille väri- ja syvyysarvot. Tästä saatu tulos tallennetaan vielä kehyspuskuriin, joka puolestaan tuo kaikki valmiit kehykset ruudulle näytettäväksi muodostaen grafiikan käyttäjän nähtäväksi. [8.]

## WebGL:n toiminnan hyödyt ja ongelmat

WebGL:n tuottama grafiikka toimii näytönohjaimen teholla [5]. Tämä antaa sille sekä ison hyödyn että myös joitakin ongelmia. Hyötynä on näytönohjaimen mahdollistama suuri laskentateho, joka on ollut koko ajan nopeassa kasvussa. Näin myös verkkoselaimessa toimivat sovellukset voidaan tehdä entistä monipuolisemmiksi ja vaativammiksi. Ongelmana puolestaan on näytönohjaimen täydellinen puuttuminen tai sen liian vähäinen tehokkuus, jolloin esimerkiksi kaikki tietokoneetkaan eivät kykene toistamaan WebGL-sovelluksia riittävän tehokkaasti tai jopa ollenkaan [9]. Tämä ongelma kuitenkin häviää pian, sillä uusissa laitteissa on nykyään mukana riittävän tehokkaat grafiikkalaitteet yhdistettynä niiden tehokkaampiin suorittimiin. Tämän lisäksi Chrome-selaimella pystytään käyttämään SwiftShader-teknologiaa, joka hoitaa 3D-renderöintiä ohjelmatasolla, joten aina varsinaista näytönohjainta ei edes tarvita [10].

Verkkoselaimet on nyt vielä tehty käsittelemään WebGL:ää epäkäytännöllisesti, sillä ne piirtävät canvas-elementin aina uudestaan, vaikka mikään ei näyttäisi liikkuvan. Käytännössä siis WebGL-sisältö muodostuu näytönohjaimen tekemänä, mutta sen saaminen HTML-sivulle yhdessä muun HTML-sisällön kanssa aiheuttaa suorittimen mahdollisesti kovankin rasittumisen. Asia on verkkoselainten tekijöiden tiedossa, joten se varmasti toteutetaan viisaammin lähitulevaisuudessa. Ongelma ei kuitenkaan ole kovin merkittävä, eikä sitä juurikaan huomaa kevyemmissä WebGL-sovelluksissa. [5.]

## Turvallisuus

WebGL on herättänyt kysymyksiä sen turvallisuudesta [5]. Näytönohjain tarjoaa suuren laskentatehon, ja jos väärät henkilöt pääsisivät siihen käsiksi, se voisi merkitä isojakin ongelmia. Isoin WebGL:n turvallisuuspuutteiden arvostelija on ollut Microsoft, joka ei vielä tue WebGL:ää. [11.]

Turvallisuusongelmia on tullut niihin verkkoselaimiin, jotka ovat sallineet tuen WebGL:lle. WebGL:n toimimiseksi verkkoselaimista on täytynyt avata ja löysentää useita turvallisuusmäärityksiä, jotka tavallisesti olisi hyvä pitää suljettuina. Tämän ohella näytönohjaimia ei ole suunniteltu kestämään erilaisia hyökkäyksiä, joten jos verkkoselain päästää niihin käsiksi, voi vahinkoa syntyä. Heikkoudet eivät siis ole itsessään verkkoselaimissa vaan näytönohjaimien puutteellisissa ajureissa, joista

puuttuvat suojaukset sen jälkeen, kun haitallista koodia pääsee niihin verkkosivuilta. [11.]

Kesäkuussa 2011 Context Information Security -yritys onnistui löytämään WebGL:ää tukevista verkkoselaimista haavoittuvuuksia, joiden avulla pystyttiin esimerkiksi ottamaan käyttäjän tietämättä kuvakaappauksia hänen tietokoneeltaan ja ylikuormittamaan näytönohjain tietynlaisilla komennoilla, mikä sai näytönohjaimen jumiin ja huonoimmassa tapauksessa koko käyttöjärjestelmän kaatumaan. [12.] Nämä haavoittuvuudet johtuivat sekä verkkoselainten puutteellisista suojauksista että myös itse WebGL:n tavasta toimia. Verkkoselainten tekijät korjasivat puutteet seuraaviin versioihinsa, ja Khronos teki myös osaltaan erilaisia toimenpiteitä palvelunestojen estämiseksi ja muita tietoturvapäivityksiä. [13; 14.]

## 2.3 WebGL-ohjelmistokehykset

### CopperLicht ja CopperCube

CopperLicht on Ambieran kehittelemä WebGL-ohjelmistokehys, joka mahdollistaa WebGL:n käytön ensisijaisesti 3D-pelien tekemiseen, mutta myös muihin sovelluksiin. Se on saatavilla Ambieran verkkosivuilta (http://ambiera.com). CopperLicht tuo nopean JavaScript-3D-pelimoottorin ja paljon pelinrakennukseen tarkoitettuja työkaluja ja luokkia. [15.] Näiden lisäksi se tuo CopperLichtin rinnalla toimivan CopperCube-ohjelman, joka mahdollistaa 3D-editorillaan visuaalisen käyttöliittymän eri sovellusten tekemiseen [16].

CopperCube on kevyehkö 3D-editori, joka tukee julkaisemista suoraan WebGL-muodossa. Se on tällä hetkellä ainutlaatuinen 3D-editori, joka mahdollistaa WebGL-pelin tai muunlaisenkin 3D-sovelluksen tekemisen pääasiassa visuaalisesti. CopperCube tarjoaa myös mahdollisuuden käyttää JavaScriptiä sen rinnalla edellä mainitun CopperLichtin avulla. [16.]

3D-malleja hallitaan CopperCubessa objekteina, joita voidaan esimerkiksi liikuttaa, kääntää tai skaalata hieman samaan tapaan kuin 3D-mallinnusohjelmissa, esimerkiksi Autodesk 3ds Maxissa. Käyttöliittymä on melko yksinkertainen, ja kaikki toiminnot ovat yleensä enintään muutamien painalluksien päässä. Kullakin objektilla on omat Attributes-, Materials- ja Behaviour-välilehdet. Näiltä välilehdiltä löytyvät kaikki tiedot

objektista. Attributes-kohdasta voidaan määrittää objektin perustietoja, kuten sen nimi, sijainti, skaalaus ja näkyvyys sekä voiko siihen törmätä ja muita tämäntyyppisiä tietoja. CopperCuben materiaalityökaluilla voidaan määrittää objektin pintoja, tekstuureja ja muita vastaavia toimintoja sekä arvoja. CopperLicht tukee myös dynaamisia valoja, joita saadaan hallittua lisätyillä valonlähteillä ja säätämällä materiaalin pinnat dynaamisiksi [17]. CopperCubessa voidaan myös määritellä objekteille tai niiden välillä tapahtuville toiminnoille erilaisia ehtoja ja arvoja. Kaikki tämä tapahtuu ohjelman Behaviour-kohdassa, kun jokin tietty objekti on valittuna. Valmiina on kattava määrä erilaisia valmiita toimintoja, mutta on myös mahdollista lisätä manuaalisesti omia toimintoja [18].

Osa mainituista toiminnoista on hyvin yksinkertaisia, esimerkiksi kun painetaan välilyöntiä, niin soitetaan asetettu äänitiedosto. Toimintoja voidaan luoda paljon monimuotoisemminkin, kuten ensin antamalla jollekin objektille pelitoimintoja, joista sitten voidaan määritellä objekti esimerkiksi ampumaan jotakin. Ammus voidaan vielä määritellä tekemään jotain tiettyä osuessaan. Ajatuksena on siis, että toiminnoista voi tehdä pidempiä ketjuja, jotka muodostavat laajemman kokonaisuuden. [16.]

CopperCube tarjoaa melko paljon erilaisia toimintoja, joita yhdistelemällä voidaan saada mielenkiintoisia ratkaisuja, mutta jos ne eivät kuitenkaan riitä, on mahdollista tuoda sisään JavaScriptiäkin tai ActionScript 3:a, jos halutaan tehdä Flash-peli. CopperCube mainostaa itseään kuitenkin juuri editorina, jossa koodausta ei tarvita, eli se on vain lisänä, jos tarvetta tulee, ei edellytyksenä pelin rakentamiseen. [16.]

X3DOM

X3DOM on avoimen lähdekoodin WebGL-ohjelmistokehys, jonka avulla saadaan esitettyä interaktiivista 3D:tä suoraan HTML5-verkkoselaimessa. Se on saatavilla X3DOMin verkkosivuilta (http://www.x3dom.org). Se on ehdotus HTML5:n ja deklaratiivisen 3D-sisällön saattamisesta yhteen. Sen tarkoitus on toisin sanoen mahdollistaa X3D-elementtien käyttö suoraan DOMin hierarkkisessa puurakenteessa. [19.]

X3D-elementtien käyttäminen DOMissa mahdollistaa monia hyödyllisiä asioita, kuten kokonaisen interaktiivisen 3D-näkymän tuomisen HTML-puurakenteeseen ja sen muokkaamisen DOM-elementtien avulla. 3D:n reaaliaikainen renderöinti tehdään

WebGL:n avulla, joka on sulautettuna X3DOM-ratkaisuun. Tämän sekä DOMin käytön takia mitään erillistä liitännäistä ei tarvitse asentaa verkkoselaimeen. [19.]

X3DOM pohjautuu X3D:hen, jota se päätyi käyttämään, koska se on avointa ja perustuu vahvasti XML-kieleen. Tämän avulla myös animaatiot saatiin toimimaan kätevästi. Avoimuus, kasvanut tehokkuus ja sopivuus helposti muokattavaksi johtivat myös JavaScriptin käyttöön X3DOMin hallitsemiseksi. [20.]

Muut ohjelmistokehykset

CopperLichtin ja X3DOMin lisäksi WebGL:lle on olemassa monia muita ohjelmistokehyksiä. Jokaisella niistä on hieman eriävät toteutustavat ja erikoistumiskohteet, mutta kaikki pohjautuvat lopulta samaan WebGL:ään.

Taulukossa 1 on luettelo tämänhetkisistä WebGL:n ohjelmistokehyksistä ja lyhyt kuvaus jokaisesta. Yhteistä niille on JavaScript ja WebGL:n tarjoamat toiminnot, joihin ne kaikki pohjautuvat.

Taulukko 1. Olemassa olevat WebGL-ohjelmistokehykset [21].

| **WebGL-ohjelmistokehys** | **Kuvaus** |
|---|---|
| oogl.js | kevyt oliopohjainen JavaScript-toteutus |
| C3DL | JavaScript-kirjasto |
| Cesium | 3D-karttoihin erikoistunut JavaScript-kirjasto |
| CopperLicht | pelien tekoon erikoistunut JavaScript-kirjasto |
| CubicVR.js | JavaScript-kirjasto |
| EnergizeGL | yksinkertaisten 3D-esitysten tekeminen |
| GammaJS | JavaScript-kirjasto yksinkertaisille peleille |
| GLGE | helppokäyttöinen JavaScript-kirjasto |
| GlowScript | yksinkertainen JavaScript-kirjasto |
| GTW | erikoistunut käyttöliittymien tekemiseen |
| gwt-g3d | Google Web Toolkit -toteutus |
| GwtGL | Google Web Toolkit -toteutus |
| Inka3D | JavaScript-toteutus Autodesk Mayalle |
| J3D | JavaScript-kirjasto |
| Jax | tehokas JavaScript-ohjelmistokehys |
| JebGL | Java-toteutus ei-tuetuille selaimille |
| KickJS | erikoistunut pelien tekemiseen |
| KriWeb | pohjautuu Dart-ohjelmointikieleen |
| Lightgl.js | GLSL:ää lähempänä oleva JavaScript-kirjasto |
| MathBox | matemaattisten mallien esittäminen |
| O3D | JavaScript-ohjelmointirajapinta |
| Oak3D | helppokäyttöinen JavaScript-kirjasto |
| OpenWebGlobe SDK | erikoistunut 3D-maastojen esittämiseen |
| OSG.JS | OpenSceneGraphin JavaScript-toteutus |
| Parallax | Google Web Toolkit -toteutus |

| | |
|---|---|
| PhiloGL | nopeuteen panostava JavaScript-kirjasto |
| SceneJS | tehokas JavaScript-kirjasto |
| SpiderGL | JavaScript-kirjasto |
| StormEngineC | JavaScript-kirjasto |
| taccGL | HTML-sivujen animointi |
| TDL | nopeuteen panostava JavaScript-kirjasto |
| Three.js | helppokäyttöinen JavaScript-kirjasto |
| WGT | Google Web Toolkit -toteutus |
| Wt WGLWidget | C++:aan pohjautuva toteutus |
| X3DOM | X3D:hen pohjautuva toteutus |

Yksi merkittävä ohjelmistokehys on Three.js. Se on JavaScript-kirjasto, jonka avulla voidaan luoda 3D:tä verkkoselaimiin. Sen renderöijinä toimivat WebGL, <canvas>, <svg> ja CSS3D. Three.js:n ajatuksena on tarjota helppokäyttöinen ja suoraviivainen tapa tuottaa 3D:tä HTML5-selaimiin. Sen rakenne koostuu JavaScript-käskyihin, joiden avulla saadaan sekä hallittua koko 3D-esitystä että myös luotua tominnallisuuksia siihen. [21.] Metropolia Ammattikorkeakoulun opintojaksolla päädyttiin käyttämään CopperLichtin ja X3DOMin lisäksi Three.js-ohjelmistokehystä, koska se sopi opetukseen. Three.js-ohjelmistokehykselle on beta-vaiheessa oleva Three.js editor, joka on WebGL:llä toimiva selainpohjainen 3D-mallinnusohjelma (http://threejs.org/editor). Sen avulla voidaan luoda yksinkertaisia 3D-malleja ja viedä niitä ulos kätevästi suoraan lähdekoodina.

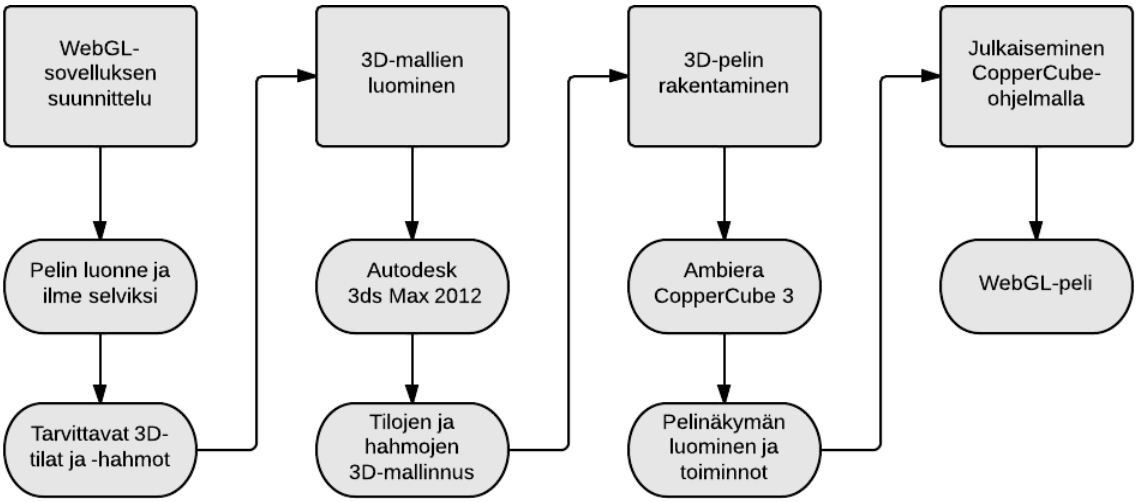
# 3 WebGL-sovelluksen suunnittelu

## 3.1 CopperCube-sovelluksen suunnittelu

Insinöörityössä tehdyn WebGL-sovelluksen tuli olla joko yksinkertainen 3D-peli tai vastaavasti kevyt 3D-sovellus, jonka avulla saataisiin selaimeen käytännöllistä ja interaktiivista 3D:tä. Jo alkuvaiheessa 3D-peli alkoi osoittautua liian laajaksi kokonaisuudeksi lopputyötä varten, koska sen tarvitsema koodimäärä olisi suuri käytössä olevilla puutteellisilla WebGL-ohjelmistokehyksillä. WebGL ja sen ohjelmistokehykset olivat vielä aika alkutekijöissään käyttäjäystävällisyydessä ja erilaisissa toiminnoissa, eli ison osan perusominaisuuksista olisi joutunut luomaan käytännössä alusta alkaen. Näin ollen WebGL-sovellukseksi alkoi muodostua jonkinlainen kevyehkö interaktiivinen esillepano, esimerkiksi tuoteluettelo tai tietyn tilan esittely. Tämä vaihtoehto olisi ollut käytännöllinen, ja se olisi tuonut esille tarvittavia asioita teknologian tutkimista varten. Toteutus olisi tehty X3DOM-ohjelmistokehystä käyttäen, joka tarjosi tämäntyyppisiä esitystapoja varten hyvät mahdollisuudet ja toiminnot.

Suunnitelmiin tuli kuitenkin vielä muutos, kun WebGL:n ohjelmistokehyksistä löytyi Ambieran tekemä CopperCube-ohjelma ja sen rinnalla toimiva CopperLicht-ohjelmistokehys. Ne mahdollistivat WebGL:n käyttämisen enemmän pelipainoitteiseen 3D-sovellukseen, joka pysyisi riittävän yksinkertaisella tasolla myös oppilaitoksen ja opiskelijoiden käyttöä ajatellen.

Kuvassa 1 esitellään suunnitellut prosessit WebGL-sovelluksen luomiseen. Kaikki lähti ideoinnissa siitä, minkälainen sovellus tulisi olemaan. Sen jälkeen luotaisiin 3D-mallit ideoiden pohjalta ja aseteltaisiin ne pelinäkymään sekä annettaisiin niille toimintoja. Lopuksi sovellus julkaistaisiin WebGL-muotoon.

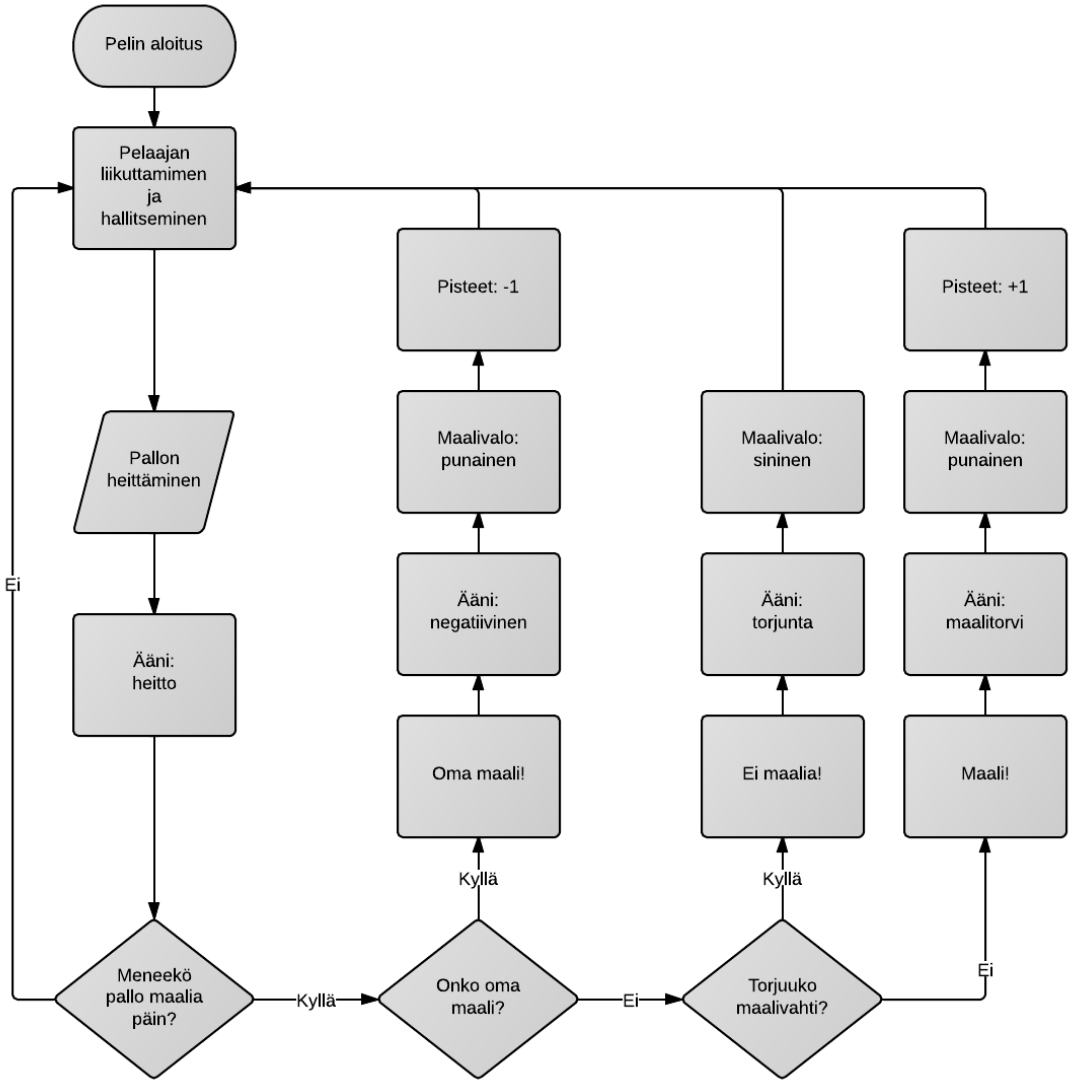



Kuva 1. WebGL-sovellusprosessin hahmottuminen eri vaiheineen.

## 3D-pelisovelluksen hahmottuminen

Pääsovelluksena tulisi siis olemaan yksinkertainen 3D-peli, jonka ideana olisi pelaajan liikuttaminen pelikentällä ja pallojen heittäminen maaliin. Pelihahmon tulisi pystyä liikkumaan vapaasti pelikentällä käyttäjän mielen mukaan, kuitenkin seinien ja pelikentän reunojen sisäpuolella. Pallon heitosta maaliin pistelaskuri laskisi tilanteen ja ilmoittaisi sen samantien pelaajalle. Haasteena olisi myös maalivahti, joka liikkuisi maalin edessä ja torjuisi palloja, ettei maalinteko olisi liian helppoa. Pelissä olisi myös tarpeellista olla törmäyksien tunnistus ja yksinkertainen fysiikkamoottori pallon ja pelaajan liikkumista varten. CopperCube mahdollisti nämä ominaisuudet sen tarjoaman pelimoottorin avulla. [15.]

Kuva 2 esittää pelin toimintaperiaatetta ja sen eri avainkohtia. Perustilassa pelaaja pystyisi liikkumaan 3D-alueella ja hallitsemaan pelihahmoaan esimerkiksi pyörimällä ja hyppimällä. Tapahtumien sarja alkaisi, kun pelaaja heittää pallon. Jos palloa ei heitettäisi maalia päin, se jatkaisi lentorataansa, kunnes törmäisi johonkin esteeseen, ja mitään muuta ei tapahtuisi. Jos puolestaan pallo olisi menossa maalia päin, niin riippuen siitä, kumpaa maalia päin se menisi ja onnistuisiko vastustajan maalivahti torjumaan sitä, joko annettaisiin pelaajalle piste tai poistettaisiin pelaajalta yksi piste.




Kuva 2. Pelin eteneminen ja sen tärkeimmät vaiheet.

Pelikenttänä toimisi koripallokenttää muistuttava alue, joka olisi muuten rajattu jääkiekkokaukalon tyyppiseksi kaukaloksi. Itse pelin toimintakin perustuisi näihin kahteen urheilulajiin, sillä pelaaja heittäisi palloja hieman koripalloheittojen tyyliin ja maaleina toimisivat jääkiekkomaiset maalit. Pelikenttä olisi siis pienehkö sisäkenttä. Tällaisen kentän on luonnollista olla joko sisähallissa tai jonkinlaisessa areenassa. Sopivimmaksi vaihtoehdoksi päätyi halli, koska se sopisi pelin tilanteeseen ja tunnelmaan paremmin. Kyseessä olisi vain yhden pelaajan pallonheittely maalivahtia vastaan, mikä sopii erinomaisesti harjoitushallissa tapahtuvaksi mittelöksi. Näin saatiin hahmoteltua itse pelin näyttämö.

Peli tarvitsisi vielä luonnollisesti pelivälinettä, jota heitetään maaliin. Tätä varten koripallo olisi täydellinen vaihtoehto. Pelin tyylin mukaisesti sitä kuitenkin muutettaisiin hieman enemmän kiekontyyliseksi pelivälineeksi teksturoinnin avulla.

Pelihahmon suhteen olisi kaksi toteutusvaihtoehtoa. Ensimmäinen olisi sen silmistä päin oleva näkymä, joka antaisi vaikutelman, että pelaaja itse olisi kentällä ja näkisi tapahtumat pelihahmon silmin. Toinen puolestaan antaisi hieman eri perspektiivin ja näyttäisi pelitapahtumat pelihahmon selän takaa hieman yläviistosta. Isoimpana erona näissä olisi pelihahmon näkyvyys, koska silmistä kuvatussa näkymässä pelihahmoa ei näkyisi. Tähän silmistä kuvattuun toteutustapaan päädyttiin kahdesta syystä: pelivälinettä olisi huomattavasti helpompi heittää ja tähdätä 3D-maailmassa, kun näkee suoraan oikeasta kuvakulmasta, minne päin palloa ollaan heittämässä, ja pelikaukalon pienen koon ansiosta siihen saisi paremman tunnelman, kun on kentän tasalla itse pelaajana.

Vastustajana olisi maalivahti, joka liikkuisi maalin edessä edestakaisin ja heiluttaisi käsiään torjuntoja varten pyrkien estämään maalien synnyn. Jotta pelin painopiste ei siirtyisi liikaa 3D-mallintamisen puolelle, tulisi maalivahdin olla riittävän yksinkertainen toteuttaa. Näin ollen siitä tulisi hieman humoristinen piirrostyylinen hahmo, eikä todellisuutta tavoitteleva ihmishahmo. Sen tarkoituksena olisi vain pyrkiä estämään helpot heitot maalia päin.

Pelissä olisi myös tarpeen olla mukana jonkinlainen pisteenlaskenta, joka pitäisi kirjaa tehtyjen maalien määrästä ja antaisi pelaajalle sekä tietoa että myös tavoitetta ja intoa tehdä enemmän maaleja. Tämä tulisi hoidettua television urheilulähetyksistä tutulla grafiikalla, joka pysyy aina näytöllä näyttäen tilanteen ja peliajan. Apuna käytettäisiin muuttujaa, johon maalien määrä tallennettaisiin. Lisäksi tähän vaikuttaisivat omaan maaliin heitetyt pallot, jotka vähentäisivät pisteitä. Näiden lisäksi peliin lisättäisiin vielä eri tapahtumiin liittyvät äänitehosteet ja mahdolliset muut interaktiiviset toiminnot, kuten maalivalon syttyminen, kun pelaaja heittää maalin.

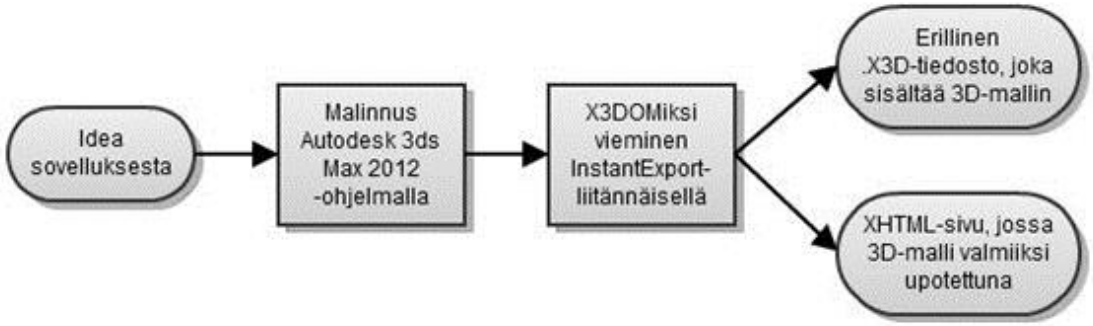
Edellä kuvailtu peli toteutettaisiin kahdella tavalla, joista ensimmäinen olisi CopperCube-ohjelman graafisella 3D-editorilla valmistettu tekijälle käyttäväystävällinen versio. Se olisi myös pelin pääversio. Toinen tapa samantyyppisen pelin tekemiseen olisi pääosin CopperLicht-ohjelmistokehyksellä valmistettava versio, joka ei juurikaan käyttäisi graafista 3D-editoria vaan painottuisi enemmän pelin hallitsemiseen koodilla.

Tämä lähestymistapa olisi kuitenkin paljon epäkäytännöllisempi tämän työn kannalta, joten sitä sivuttaisiin vain hieman esimerkkitasolla ja tarjoten katsaus sen toteuttamisperiaatteeseen.

## 3.2 X3DOM-esimerkit

Pääsovelluksen eli 3D-pelin lisäksi olisi myös toinen lähestymistapa WebGL:n tuomiseksi ja käyttämiseksi verkkoselaimessa: se perustuisi X3DOM-ohjelmistokehykseen. Pääsovellukseen verrattuna tämä olisi suppeampi kokonaisuus. Tämän vaihtoehdon tarkoituksena olisi esitellä hieman toisenlainen tapa tuoda 3D-grafiikkaa Autodesk 3ds Max -ohjelmasta suoraan verkkoselaimeen ilman erillistä ohjelmaa välissä. Tätä toimintatapaa varten tarvittaisiin enemmän työskentelyä lähdekoodin parissa.

Kuva 3 kuvaa 3D-mallien tuottamisen X3DOM-muotoon, jota voidaan puolestaan käyttää WebGL-verkkoselaimissa. Tarvittaessa voidaan vielä muokata sen toiminnallisuuksia HTML- ja JavaScript-koodilla.




Kuva 3. X3DOMin tuottamisprosessi käyttäen Autodesk 3ds Max -ohjelmaa.

Ajatuksena oli, että tehtäisiin monia pieniä X3DOM-esimerkkisovelluksia, joita sitten tarkasteltaisiin tarkemmin ja tutustuttaisiin niiden toimintamalliin ja rakenteeseen. Nämä esimerkkisovellukset perustuisivat pitkälti suoraan X3DOM-kehittäjien esimerkkeihin, mutta osa niistä olisi myös hieman poikkeavia. Niiden lähdekoodeihin lisättäisiin havainnollistamista varten koodien selityksiä, jotka helpottaisivat eri kohtien toiminnallisuuksien ymmärtämistä. Tarkoituksena olisi tuoda esille tärkeitä ja hyödyllisiä ominaisuuksia, joiden avulla saataisiin sekä tuotua 3D-malleja tehokkaasti verkkosivuille että hallittua niitä ja lisättyä interaktiivisuutta niihin. Korostettuja

ominaisuuksia olisivat muun muassa HTML- ja XHTML-rakenteet, eri toiminnot, kuten erillisen X3D-tiedoston tuominen sivulle inline-noodin avulla ja 3D-objektien painalluksen tunnistaminen sekä 3D-objektin arvojen muuttaminen reaaliajassa. Lisäksi läpinäkyvyys, moninäkymät ja 3D-noodien lisääminen ja vähentäminen reaaliajassa olisivat tutkimisen kohteina. [22.]

Lisäksi tehtäisiin myös yksinkertainen 3D-animaatio, jossa kuu kiertäisi maata radallaan ja sekä maapallo että kuu myös pyörisivät omien akseliensa ympäri. Muista esimerkeistä poiketen tämä animaatio ei pohjautuisi suoraan mihinkään X3DOM-kehittäjien esimerkkiin.

# 4 WebGL-sovelluksen toteutus

## 4.1 3D-peli

Pääsovellukseksi luotiin kevyt 3D-urheilupeli, joka koostui useista eri komponenteista: urheiluhalli, pelikenttä, pelihahmo, maalit, peliväline ja maalivahti. Yhdessä ne muodostivat kokonaisuuden, jossa pelaaja ohjasi pelihahmoaan pelikentällä tarkoituksenaan heittää pelivälinettä maalivahdin ohi ja tehdä näin mahdollisimman monta maalia. Maalitilanne päivittyi automaattisesti aina, kun pallo meni maaliin, ja tilanne näytettiin pelaajalle tulosikkunnassa.

Peli luotiin Autodesk 3ds Max 2012- ja Ambiera CopperCube 3.0.2 -ohjelmilla. Se toimii WebGL:ää tukevissa verkkoselaimissa ja käyttöliittyminä toimivat hiiri ja näppäimistö.

### 4.1.1 Mallintaminen Autodesk 3ds Max 2012 -ohjelmalla

Kaikki 3D-urheilupeliä varten tehdyt mallinnukset tehtiin Autodesk 3ds Max 2012 -ohjelmalla (http://usa.autodesk.com/3ds-max). Perusajatuksena oli, että 3D-mallit pysyisivät melko yksinkertaisina, mutta kuitenkin riittävän monimuotoisina, jotta niillä saataisiin kokeiltua erilaisia tuettuja toimintoja ja mahdollisia ongelmia WebGL:n kanssa.

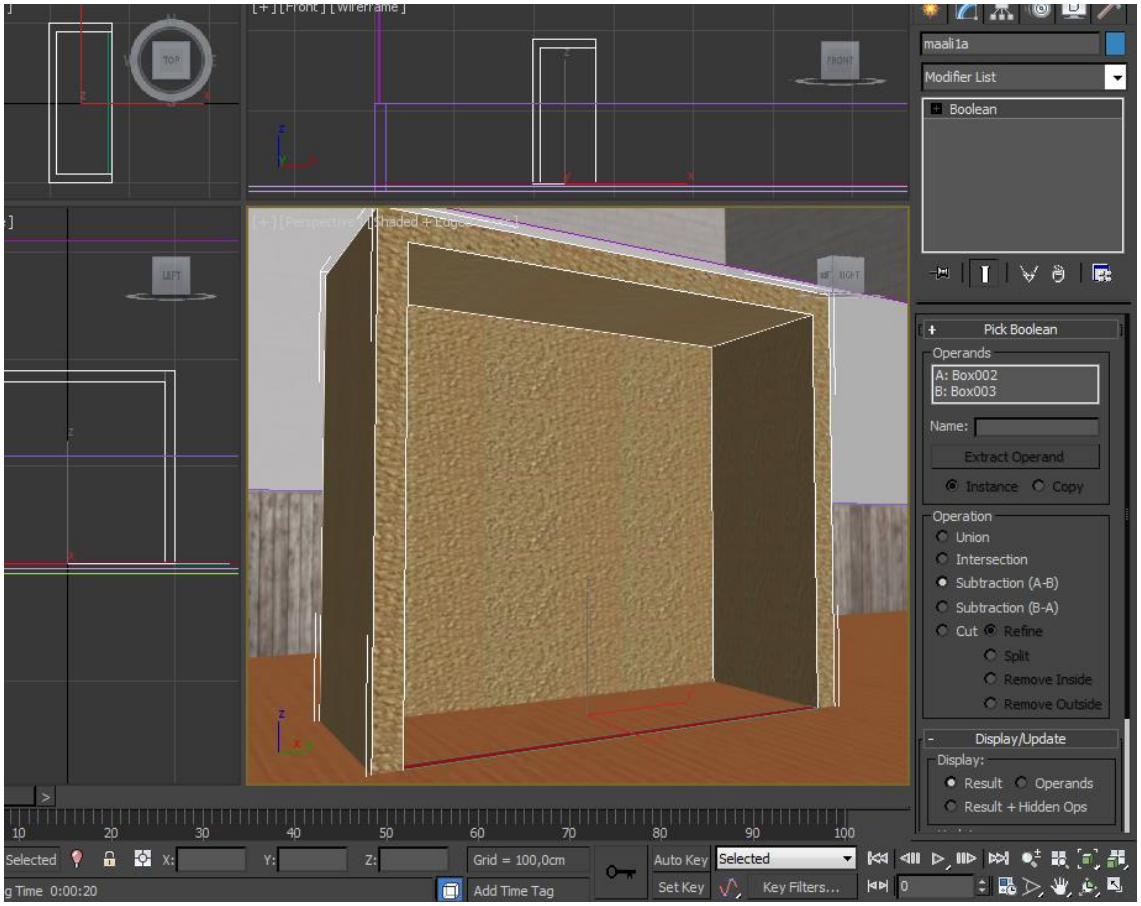
Ennen mallinnusten aloittamista tarkastettiin 3ds Max -ohjelman säädöt sopiviksi, jotta mittakaavat olisivat realistiset peliä varten. Varsinainen mallinnus alkoi urheiluhallista, joka toimi koko pelin perusrunkona, sillä kaikki pelin tapahtumat tapahtuisivat sen sisällä ja se olisi myös suurin yksittäinen kokonaisuus pelissä. Urheiluhalli oli rakenteeltaan erittäin yksinkertainen. Se koostui lattiasta, neljästä seinästä ja katosta, jotka yhdistettiin yhdeksi kokonaisuudeksi eli halliksi. Niiden mallinnuksessa ei ollut mitään ihmeellistä, vaan ne luotiin box-työkalun avulla käyttäen näppäimistösyöttöä, jonka avulla niistä saatiin juuri halutun kokoiset. Tämän jälkeen ne aseteltiin paikoilleen muodostamaan urheiluhallin rakenteet, ja niistä tehtiin yksi liikuteltava objekti valitsemalla kaikki osat yhdeksi ryhmäksi.

Urheiluhallin kanssa samaan kokonaisuuteen luotiin myös pelikenttä, jolla kaikki varsinainen pelaaminen tapahtuisi. Koska pelikenttä oli kiinteä osa hallia, se voitiin jo tässä vaiheessa mallintaa paikalleen halliin kiinni. Pääkohtina pelikentässä olivat

lattiapinta ja kaukalon reunat ja maalit, jotka yhdessä muodostivat 3D-pelin varsinaisen pelialueen. Tässä vaiheessa suunnitelmat muuttuivat hieman pois jääkiekkokaukalosta ja suuntautuivat aivan uudenlaiseen urheiluun, jossa yhdistyivät sekä jääkiekko että koripallo.

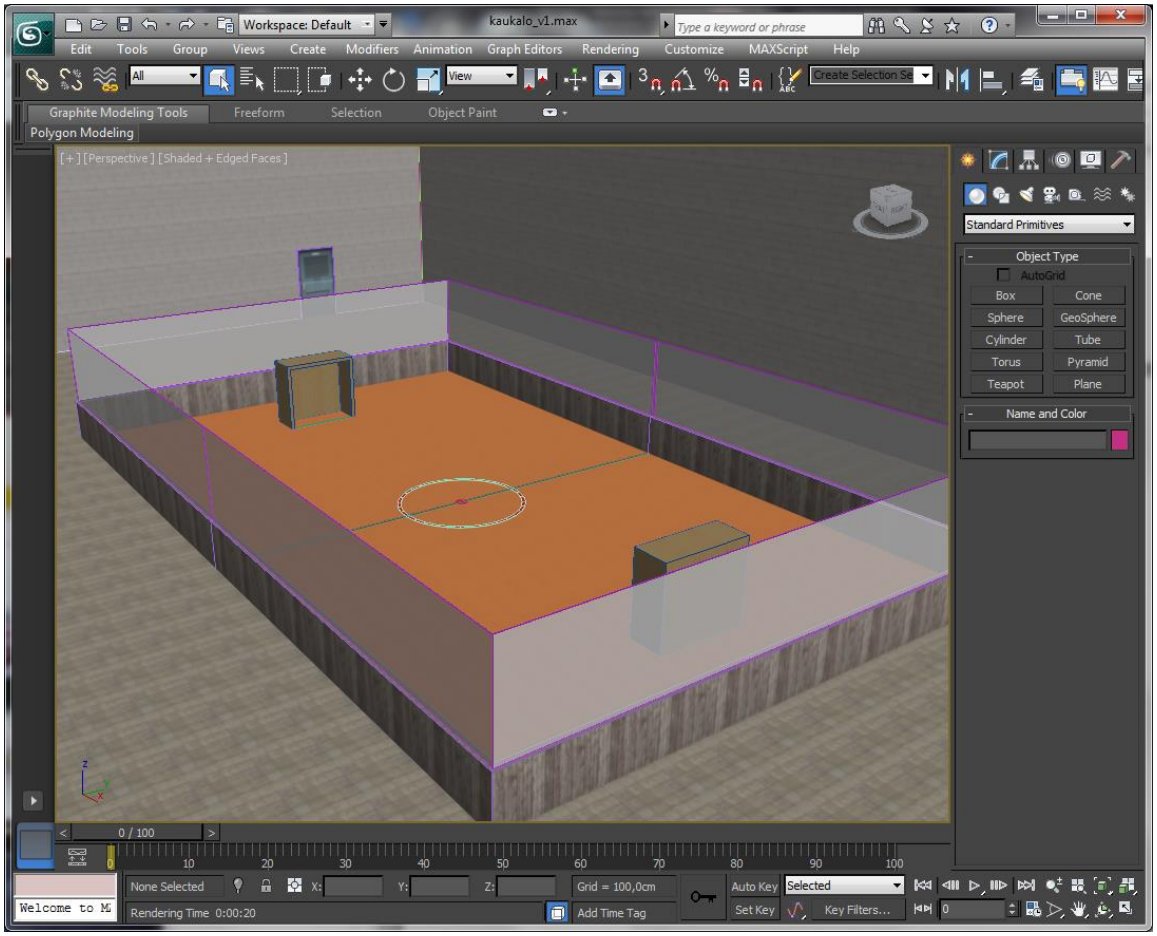
Pelikenttä niin ikään muodostui box-työkalulla tehdyistä osista. Itse kaukalo oli suorakulmion muotoinen ja sen reunat eivät nousseet kovinkaan korkeiksi. Sen sijaan reunojen päälle tehtiin jääkiekkokaukalomaiset läpinäkyvät pleksilasit, jotka toimivat suojina ja pitivät pelivälineen kentän sisällä tarjoten kuitenkin samalla ulkopuolisille näköyhteyden pelikentän tapahtumiin. Läpinäkyvät pleksilasit luotiin 3ds Max -ohjelman materiaalieditorilla raytrace-toiminnon avulla.

Maalit olivat yksi keskeisimpiä asioita tässä pelissä, joten ne luotiin seuraavaksi pelikentälle (kuva 4). Ne muodostuivat muutamista osista, jotka asetettiin myös yhteen ryhmäksi. Niihin tehtiin korostukset tolppiin ja ylärimaan, jotta ne erottuisivat helposti pelitilanteessa ja jotta pelaaja hahmottaisi maalin selkeästi kauempaakin. Toisen päädyn maali oli kopio ensimmäisestä maalista, ja se vain käännettiin 180 astetta ympäri ja laitettiin omalle paikalleen.




Kuva 4. Mallinnettu maali Autodesk 3ds Max 2012 -ohjelmassa.

Tässä vaiheessa oli luotu kaikki konkreettinen itse pelin tapahtumapaikkaa ja pelaajan ympärillä olevia alustoja varten, kuten kuvassa 5 esitetään. Kaikki tähän mennessä luotu oli siis samassa tilassa ja liitettyinä toisiinsa ryhmien avulla, niin että ne muodostivat yhden kokonaisuuden eli urheiluhallin. Tämä kokonaisuus vietiin myöhemmin yhtenä 3D-objektina CopperCube-ohjelmaan.

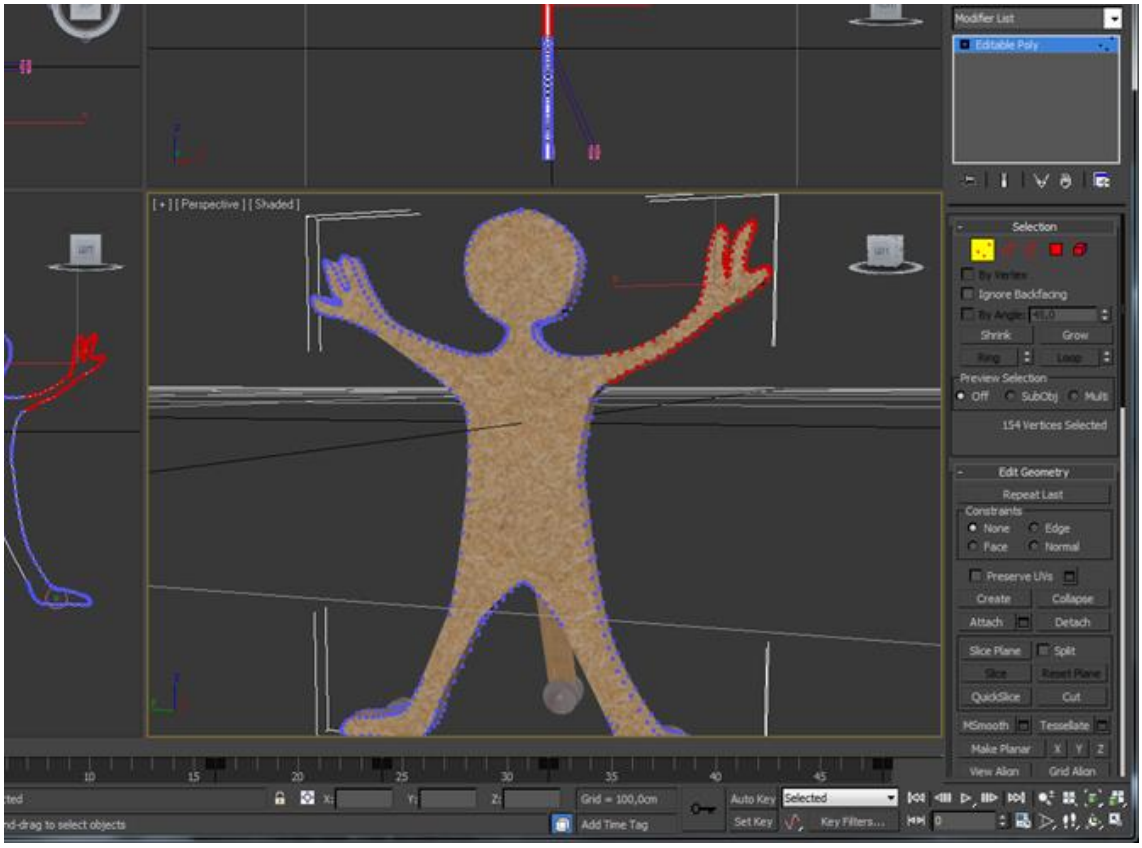



Kuva 5. Pelin urheiluhalli Autodesk 3ds Max 2012 -ohjelmassa.

Seuraavaksi luotiin maalivahti. Se tehtiin erillisenä 3D-objektina äskeisestä urheiluhallista ja myös täysin toisella tavalla. Maalivahdista piirrettiin pelkästään ääriviivat edestäpäin kuvattuna spline-työkalun avulla. Ääriviivat pursotettiin extrude-työkalulla, jolloin maalivahti sai niin sanotusti kehonsa ääriviivojen väliin. Tämä vaihe täytyi tehdä kahteen kertaan, ensin tavallisesti ja sitten peilikuvana, jotta maalivahdin keho täyttyi molempiin suuntiin. Tämän teki mahdolliseksi se, että kyseessä oli vanerilevystä tehty maalivahti. Edestäpäin katsottuna maalivahdilla oli muotoja, mutta sivulta päin sillä oli aivan tasainen pinta.

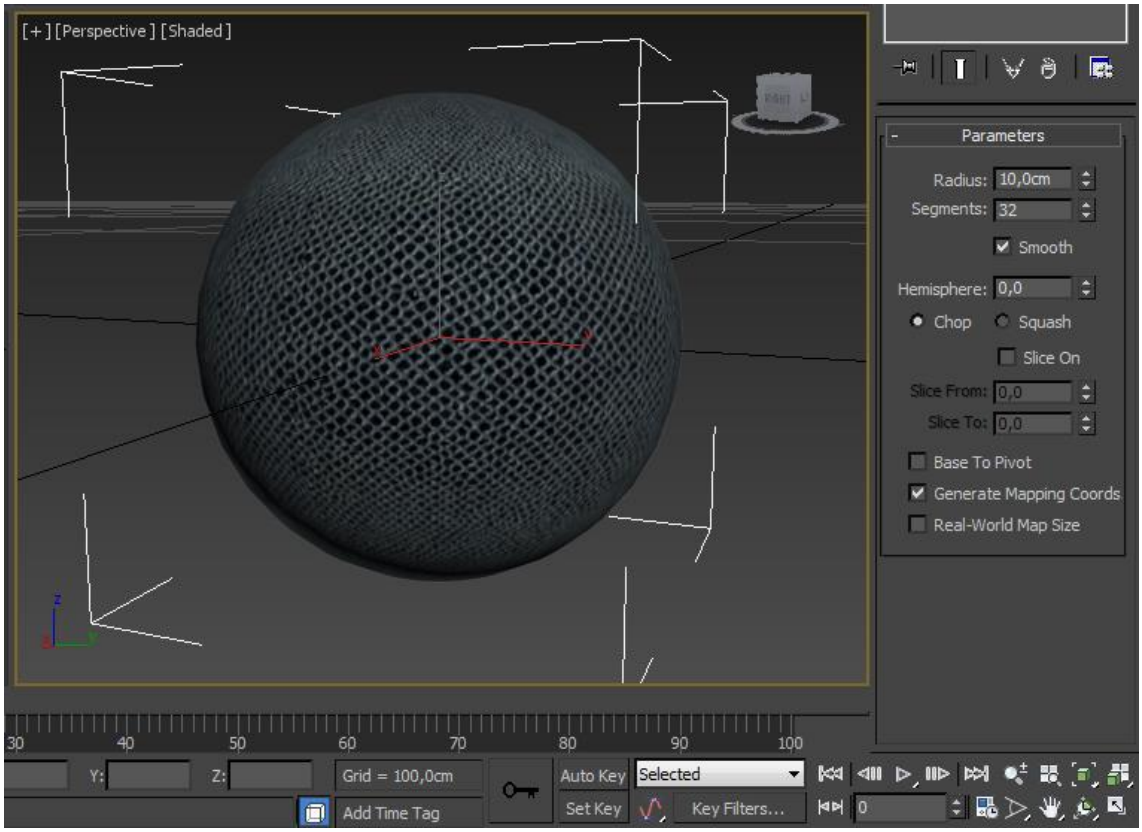
Humoristisen puolen vuoksi maalivahdille lisättiin kolmet pyörät alle. Kumpaankin jalkaan tuli yksi pyörä ja maalivahdille lisättyyn tukijalkaan, jota vasten se nojasi, tuli myös yksi pyörä. Näiden pyörien avulla maalivahdin oli tarkoitus luoda mielikuva liikkumisesta itse pelissä.

Muista mallinnuksista tähän asti poiketen maalivahdille tehtiin myös animaatio, jossa sen molemmat kädet liikkuivat ylös ja alas tarkoituksena torjua heitettyjä palloja. Tätä varten maalivahti konvertoitiin kuvan 6 mukaisesti editable poly -muotoon. Sitten animaatioon luotiin automaattisia avainkohtia, joissa jokaisessa kädet olivat tietyssä asennossa tietyssä kohtaa animaatiota. Kokonaisuutena näkymä vaikutti siltä, kuin kädet olisivat sulavassa liikkeessä koko ajan.




Kuva 6. Maalivahdin luominen ja animointi Autodesk 3ds Max 2012 -ohjelmalla.

Tässä vaiheessa luotiin peliväline eli pallo (kuva 7), joka oli lopullisessa koossaan karkeasti noin koripallon kokoinen. Se luotiin sphere-työkalulla. Peliväline oli myös oma irrallinen 3D-objekti urheiluhallista ja muista mallinnuksista. Pallolle luotiin myös muuntautumisanimaatio, jossa se puristui kasaan. Sitä käytettäisiin silloin, kun pallo osuu johonkin esteeseen.




Kuva 7. Pelivälineenä toimiva pallo luotuna Autodesk 3ds Max 2012 -ohjelmassa.

Lopuksi oli vielä tarvetta luoda lisämateriaalia peliin: maaliviivat, keskikentän punaviiva ja maalivalot. Kaikki viivat luotiin plane-työkalulla, jolla saatiin ohuet alustat kentän päälle. Maalivalot taas luotiin pilkkomalla capsule-työkalulla luotu malli boolean-operaattorin avulla ja liittelemällä vastaavia palasia yhteen.

Teksturointi

Kaikki luodut 3D-mallit tarvitsivat tietysti myös tekstuureja, joilla niihin saatiin näyttävyyttä ja eloa. Muutamaa yksinkertaista tekstuuria lukuun ottamatta käytössä olivat CGTextures-sivustolta (http://www.cgtextures.com) vapaasti käytettävät tekstuurit, joita sai käyttää esimerkiksi omissa projekteissaan, kunhan niiden alkuperä mainitaan jossain kohtaa [23]. Sivustolta löytyi kattava määrä käyttökelpoisia tekstuureja. Muut tekstuurit, kuten pelikentän maaliviivat, luotiin yksinkertaisesti Microsoft Paint -ohjelmalla.

Tekstuurit liitettiin omiin 3D-malleihinsa 3ds Max -ohjelman materiaalityökalulla, ja tarpeen tullen niitä myös vielä sommiteltiin sopimaan niihin paremmin sen avulla.

Tekstuurien liittäminen tosin onnistui vielä myöhäisemmässäkin vaiheessa CopperCube-ohjelman avulla.

Mallin vieminen ulos Autodesk 3ds Max -ohjelmasta

Tehdyt 3D-mallit täytyi viedä eri tiedostoformaatissa ulos 3ds Max -ohjelmasta, jotta niitä voitiin käyttää CopperCube-ohjelmassa. Tätä varten ei yhtä poikkeusta lukuun ottamatta tarvittu mitään ulkopuolisia lisäosia 3ds Max -ohjelmaan, vaan se onnistui suoraan sen omilla tarjoamillaan tiedostoformaateilla.

Kuten taulukosta 2 näkyy, vaihtoehtoja oli useita. Niistä käytettiin 3DS-tiedostoja, jotka toivat onnistuneesti kaikki staattiset 3D-mallit CopperCube-ohjelmaan. Animoidut 3D-mallit puolestaan vaativat hieman toisenlaisen prosessin, koska 3ds Max ei pystynyt viemään ulos animiointia sellaisessa muodossa, jota CopperCube tuki.

Taulukko 2. Autodesk 3ds Max- ja CopperCube-ohjelmien väliset toimivat tiedostoformaatit [16].

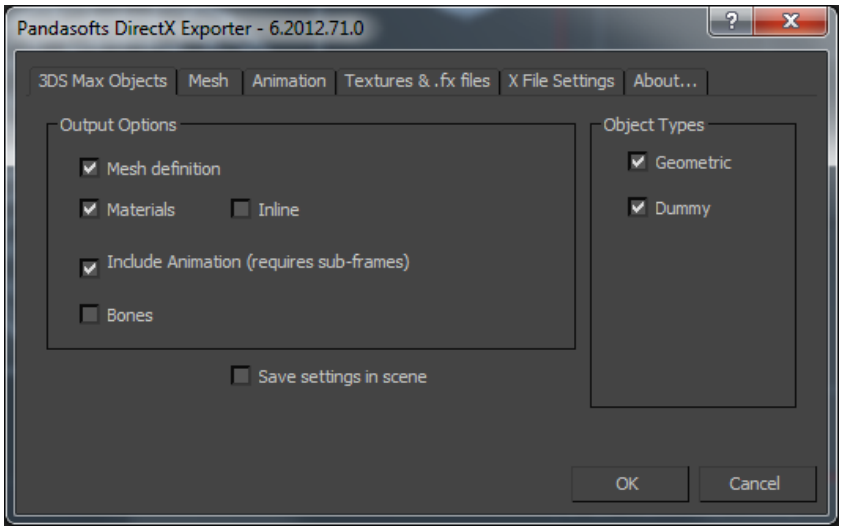
| **Tiedostoformaatti** | **Tiedostopääte** | **Lisätietoja** |
|---|---|---|
| Autodesk 3ds Max | .3ds | staattiset 3D-mallit |
| COLLADA | .dae | staattiset 3D-mallit |
| Wavefront OBJ geometry | .obj | staattiset 3D-mallit |
| DirectX | .x | animoidut 3D-mallit |

3D-mallien vieminen 3DS-muodossa ei vaatinut mitään erillisiä viemisasetuksia, vaan kaikki tapahtui helposti käytännössä pelkällä tallentamisella, koska se oli 3ds Max -ohjelman oma natiivi tiedostoformaatti, jota myös CopperCube osasi lukea.

Animoitujen 3D-mallien vieminen sen sijaan vaati erillisen liitännäisen 3ds Max -ohjelmaan. Tämä liitännäinen oli Panda DirectX, joka mahdollisti animaatioiden viemisen ulos x-tiedostomuodossa. Panda DirectX -liitännäisen saa Pandasoftin verkkosivuilta (http://www.andytather.co.uk/Panda/directxmax.aspx). CopperCube

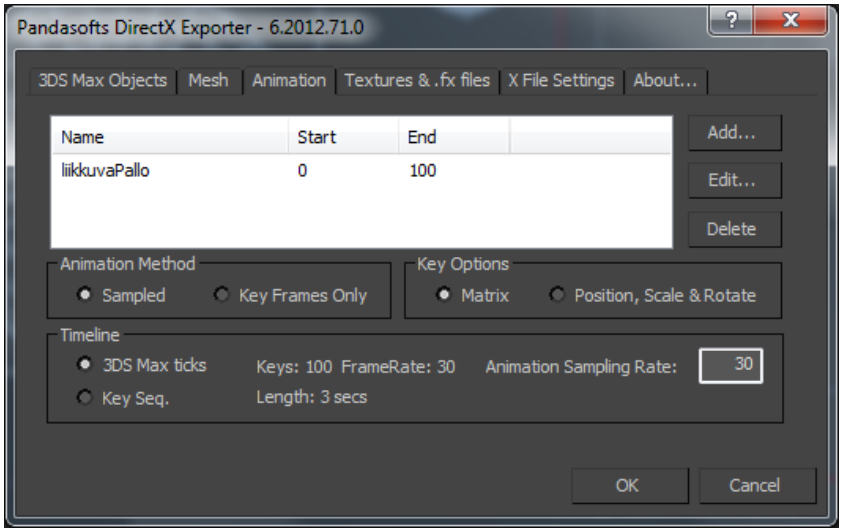
puolestaan pystyi lukemaan DirectX-tiedostoja ja tuomaan animaatioita 3ds Max -ohjelmasta. [24.]

Yksinkertaisten 3D-animaatioiden, kuten esineen liikkumisen paikasta A paikkaan B, luominen onnistui erittäin kätevästi CopperCube-ohjelman omilla työkaluilla. Niitä oli mahdollista tehdä myös kuitenkin 3ds Max -ohjelman puolella, ja niiden vieminen CopperCube-ohjelmaan oli melko helppoa samaisen Panda DirectX -liitännäisen avulla. Kuvassa 8 on esitelty Panda DirectX -liitännäisen viemisasetukset, joilla yksinkertaisen animaation vieminen CopperCube-ohjelmaan onnistui. Muuten asetukset olivat perusarvoissaan, mutta Include Animation -kohta täytyy olla valittuna. Tämän lisäksi oli vielä asetettava tarkemmat asetukset animaatiota varten.




Kuva 8. Panda DirectX -liitännäisen viemisasetukset.

Autodesk 3ds Max -ohjelmalla pallolle oli luotu animaatio, joka kesti 101 kehystä. Nämä kehykset otettiin kaikki mukaan animaatioon määrittämällä sen alku- ja loppukohdat, kuten kuvassa 9 on esitetty. Aikajanana käytettiin 3ds Max -ohjelman omaa järjestelmää. Kun haluttu animaatio saatiin valittua, se voitiin viedä ulos 3ds Max -ohjelmasta painamalla OK-painiketta.




Kuva 9. Animaation vieminen Panda DirectX -liitännäisellä.

### 4.1.2 Pelin luominen CopperCube 3.0.2 -ohjelmalla

Peli luotiin CopperCube-ohjelmalla, joka löytyy Ambieran verkkosivuilta (http://www.ambiera.com/coppercube). Tehtyjen 3D-mallien tuominen CopperCubeen onnistui yksinkertaisesti import-toiminnon avulla. Käytettävä tiedostoformaatti riippui siitä, oliko tuotava 3D-malli animoitu vai ei. Kahta lukuun ottamatta kaikki olivat staattisia 3D-malleja, joten niiden tuomiseksi käytettiin Static 3D Mesh from File -import-toimintoa ja valittiin tuotavaksi kaikki tehdyt 3DS-tiedostot.

Muutamien 3D-mallien tekstuurien kanssa ilmeni ongelmia tuomisen jälkeen. Tuodut 3D-mallit ilmaantuivat CopperCubeen hohtavan valkoisina objekteina, mikä tarkoitti, että sovellus ei onnistunut tuomaan niiden alkuperäisiä tekstuureja ja näin ollen ne jäivät paljaiksi 3D-malleiksi.Tähän löytyi ainakin kaksi mahdollista syytä. Ensimmäinen olivat viemisasetukset siinä vaiheessa, kun 3D-mallia oltiin viemässä ulos 3ds Max -ohjelmasta. Oli syytä tarkastaa huolellisesti, että käytetyt tekstuurit ovat mukana viemisasetuksissa. Toinen ongelma olivat 3ds Max -ohjelmalla tuotetut materiaalit, jotka joissain tapauksissa eivät täysin toimineet CopperCubella. Tämä kuitenkin korjattiin jo CopperCube 3.0.3 -versiossa [18]. Ongelman pystyi korjaamaan myös suoraan CopperCube-ohjelman omalla materiaalieditorilla, jossa materiaalille pystyttiin määrittämään tekstuureja. Sinne oli mahdollista tuoda ihan uusia tekstuureja tai mahdollisesti valita mukana tullut tekstuuri, jos se oli onnistunut tulemaan 3D-mallin mukana.

Poikkeuksena tekstuureista oli kaukalon pleksilasin läpinäkyvyys, joka alun perin luotiin 3ds Max -ohjelman avulla, sillä se ei toiminut CopperCube-ohjelman puolella sellaisenaan. Pleksilasin läpinäkyvyys ja pieni heijastus oli luotu raytrace-toiminnolla, jota ilmeisesti CopperCube ei tukenut. CopperCube-ohjelmalla oli kuitenkin omakin läpinäkyvyystoiminto, jolla pleksilasi saatiin taas läpinäkyväksi. Se ei kuitenkaan toiminut WebGL-muodossa vaan ainoastaan muissa sen tarjoamissa julkaisuformaateissa.

Mukana oli myös kaksi animoitua 3D-mallia: käsiään liikuttava maalivahti ja kokoon puristuva pallo. Niiden tuomiseksi CopperCube-ohjelmaan käytettiin Animated 3D Mesh -import-toimintoa. Tässä tapauksessa niiden täytyi siis olla DirectX-muodossa x-tiedostoina.

Kummankin 3D-mallin tuominen onnistui, mutta niiden animaatiot eivät suostuneet toimimaan CopperCube-ohjelman puolella. Pallon osalta tämä ei juuri yllättänyt, koska siinä oli käytetty hieman monimutkaisempaa animaatiota pallon muodon muuttamiseen. Maalivahdin animaation tuomisen epäonnistuminen kuitenkin yllätti hieman, koska käytännössä siinä ei ollut kuin käsien liikettä. Muuta syytä näihin ei löytynyt kuin 3D-mallien muodonmuutokset niiden animaatioissa. Yksinkertainen animaatio, jossa esimerkiksi esine liikkui ympäriinsä, onnistui ilman ongelmia. Vastaavat ongelmat siis ilmenivät, kun animoitu 3D-malli koki muodonmuutoksia.

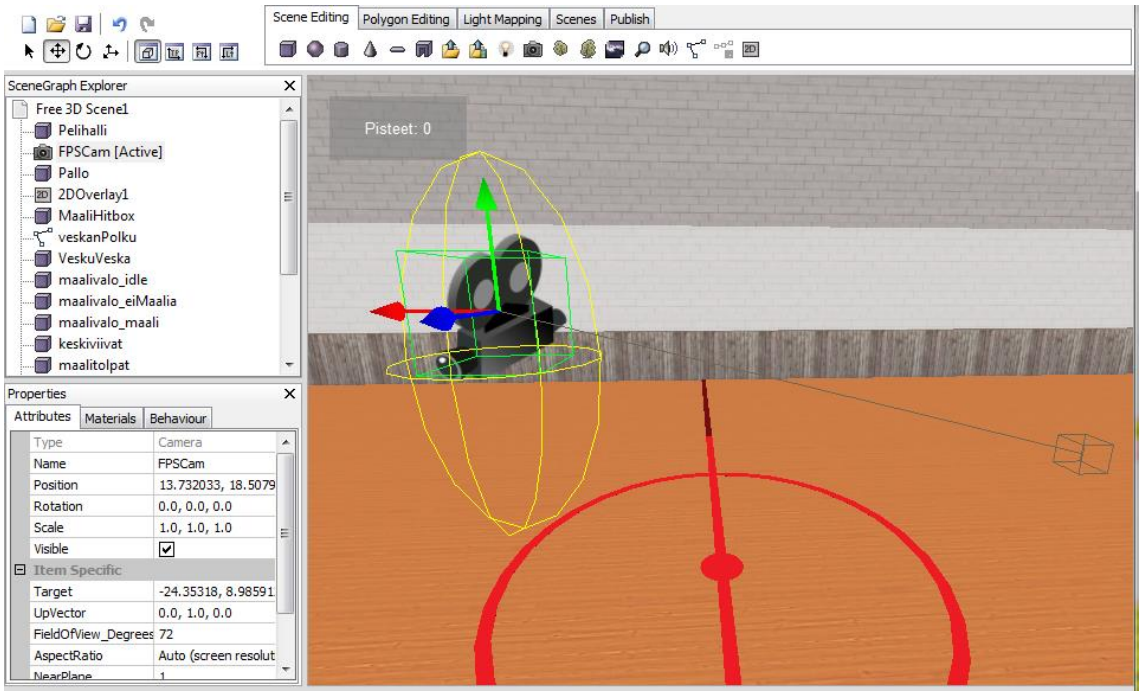
3D-mallien organisointi ja järjesteleminen

Sitä mukaa kuin 3D-malleja saatiin tuotua CopperCubeen objekteiksi, niille oli hyvä antaa kuvaavat nimet. Nimiä tarvittiin CopperCubessa erilaisten toimintojen antamiseksi objektien väleille ja myös CopperLichtissä, kun kutsutaan haluttuja objekteja.

CopperCubeen tuotaessa 3D-mallit tuntuivat ilmestyvän hieman satunnaisiin paikkoihin, joten niiden asettaminen editorin origoon oli isoksi avuksi. Ehkä tärkein tämä toimenpide oli ensimmäisenä tuodun urheiluhallin suhteen, koska kaikki objektit sen jälkeen tulivat hallin sisään ja helpotti huomattavasti, kun halli oli helpossa paikkaa origossa.

Vaikka 3D-malli luotiin järkevässä mittakaavassa 3ds Max -ohjelmassa, tässä vaiheessa huomattiin, että se oli liian pienessä koossa, jotta sitä olisi helpompi käsitellä CopperCube-ohjelman omassa virtuaalimaailmassa. Näin ollen objektin kaikkia skaalausarvoja muutettiin kymmenkertaisiksi, minkä seurauksena se oli luonnollisesti kymmenen kertaa isompi kuin alkuperäisessä tilanteessa. Tämän jälkeen se oli myös luonnollisemman kokoinen näkymään, jolla ohjelmaa hallinnoidaan. Sama skaalaustoimenpide suorittetiin kaikille tuoduille objekteille, jotta ne sopisivat vielä yhteen. Tämä ei varsinaisesti ollut ongelma tai virhe, mutta se helpotti huomattavasti CopperCube-ohjelman hallintakameran liikuttamista luodussa tilassa.

Peli tarvitsi vielä itse pelaajan ja kameran, joka näyttää, missä milloinkin mennään. Kuten jo suunnitteluvaiheessa oli päätetty, pelaaja näkisi pelin tapahtumat pelihahmon silmistä nähtynä. Tämä onnistui todella näppärästi CopperCubesta löytyvällä First Person Shooter Camera -kameratyypillä (kuva 10), jossa yhdistyivät sekä pelihahmon ominaisuudet että pelinäkymä ja sen hallinta.




Kuva 10. First Person Shooter Camera esillä CopperCube-ohjelmassa.

Kameraan oli mahdollista määrittää paljon ominaisuuksia aina sen sijainnista ja liikkumisnopeudesta erilaisiin pelillisiin toiminnallisuuksiin. Esimerkkejä näistä toiminnoista olivat pallon heittäminen ja pelihahmon liikkuminen.

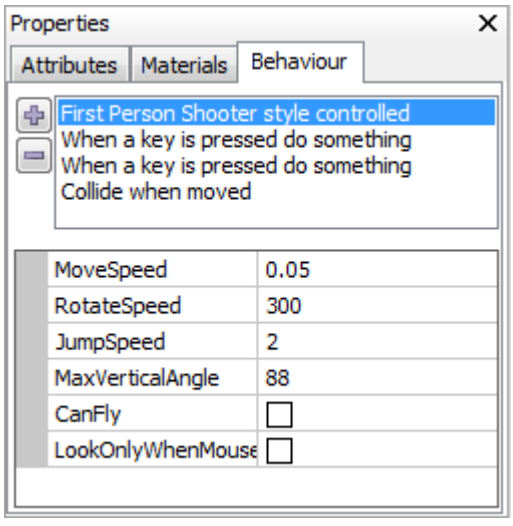
Muut 3D-mallit järjesteltiin paikoilleen, ja kahdelle niistä tehtiin lisätoimenpiteitä. Maalit jouduttiin kloonaamaan toisiksi erillisiksi objekteiksi niiden käytettävyydessä ilmenneen ongelman takia, ja pallosta tehtiin näkymätön, koska sitä itsessään objektina ei tarvittu vaan sen käyttäminen hoidettiin toimintojen avulla.

Pelin toiminnallisuudet

CopperCubessa oli oma kohta kaikenlaisten toiminnallisuuksien ja tapahtumien luomista varten. Se löytyi Behaviour-välilehdestä, jossa kaikille objekteille ja niiden väleille pystyttiin määrittämään haluttuja toimintoja ja yhteyksiä.

Pelaajan liikkuminen ja pallon heittäminen tapahtuivat saman objektin välityksellä. Kamera-objektille oli tarjolla kaksi oletustoimintoa, jotka olivat First Person Shooter style controlled- ja Collide when moved -toiminnot. Näistä jälkimmäinen määritti pelaajan fyysisen koon, gravitaation voiman ja pelaajan sijainnin suhteessa kameraan.

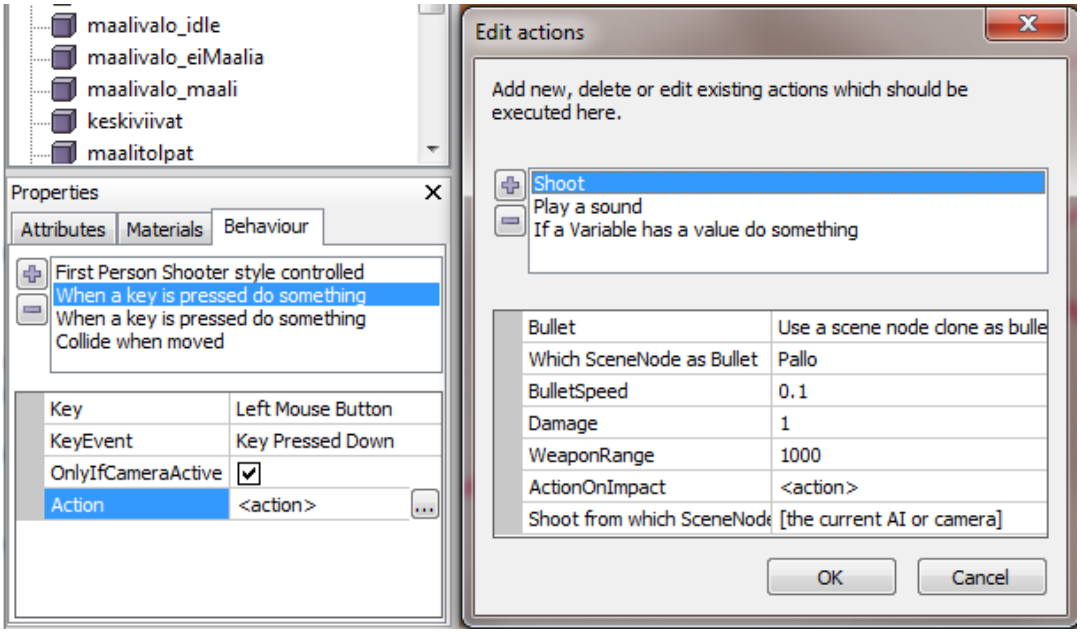
First Person Shooter style controlled -toimintoon voitiin määrittää halutut arvot esimerkiksi liikkumisnopeutta, pyörähdysnopeutta ja hyppykorkeutta varten (kuva 11). Tämä kameratyyppi määritti automaattisesti pelin liikkumisnäppäimet ja muut pelin hallintaa varten tarvittavat perusarvot.




Kuva 11. First Person Shooter style controlled -toiminnon asetukset.

Pallon heittämistä varten lisättiin uusi When a key is pressed do something -toiminto, jonka avulla pystyttiin määrittämään toimintoja, kun haluttua painiketta painettiin.

Toiminto kuului laajempaan Behaviors trigged by events -ryhmään, jolla oli mahdollista antaa erilaisia toimintoja objektien väleille. Painikkeeksi valittiin hiiren vasen painike. Kuvassa 12 on havainnollistettu toiminnon lisäämistä pallolle. Sen lisäksi, että pallolle määritettiin painike, sille määritettiin myös, mitä tulee tapahtumaan, kun tätä painiketta painetaan. Se tapahtui Action-kohdasta, josta avautui Edit actions -ikkuna. Tässä ikkunassa pystyttiin lisäämään erilaisia toimintoja, kuten ampuminen, jota käytettiin pallon heittämistä varten. Lisäksi siihen lisättiin äänitiedoston soittaminen ja muuttujan arvon asettaminen.



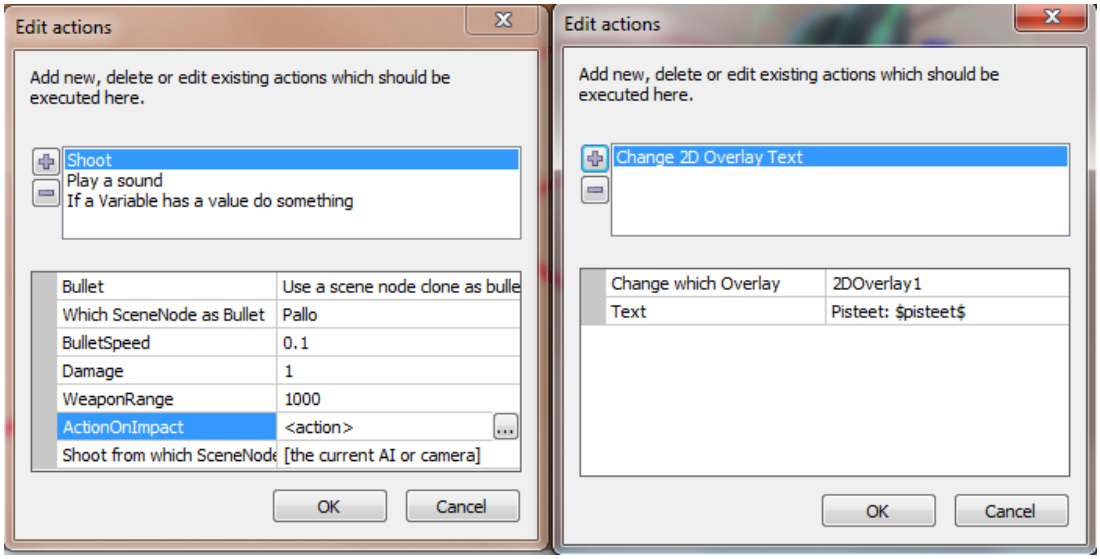

Kuva 12. Toimintojen lisääminen pallolle Behavior-välilehdessä.

Ampumistoiminnon asettaminen palloon onnistui Shoot-toiminnolla. Sen kohdassa Bullet pystyttiin määrittämään ammuksena käytettävä objekti valitsemalla siihen Use a scene node clone as bullet -vaihtoehto. Tämä mahdollisti minkä tahansa lisätyn objektin käyttämisen tässä kohtaa. Käytettäväksi objektiksi valittiin luotu peliväline eli pallo. Sille määritettiin vielä halutut arvot ammuksen nopeudeksi ja etäisyydeksi.

Äänien lisääminen onnistui Play a sound -toiminnolla, johon voitiin lisätä haluttu äänitiedosto. Tiedostoformaatteina kävivät ainakin .wav, .ogg, .mod ja .it, mutta esimerkiksi .mp3 ei ollut tuettu [16]. Tässä pelissä kaikki käytettävät äänet olivat wav-muodossa, ja ne olivat Freesound-sivustolta (http://www.freesound.org). Sieltä löytyi paljon sen käyttäjien luomia ääniä, joita suurinta osaa kuka tahansa sai käyttää

vapaasti [25]. Tähän kohtaan lisättiin ääni pallon heittoa varten, eli kun pelaaja painaa hiiren vasenta painiketta, pallo lähtee lentämään ja samalla kuuluu heittoääni.

Viimeinen tärkeä pallon määritys oli kuvassa 13 näkyvä ActionOnImpact, joka määrittää, mitä tapahtuu, kun pallo osuu toiseen objektiin. Siihen valittiin Change 2D Overlay Text -toiminto, joka tuli päivittämään myöhemmässä vaiheessa luodun pistetaulun aina, kun pallo osui mihin tahansa toiseen objektiin. Muuttujan arvon sai CopperCubessa näkyviin lisäämällä molemmille puolille $-merkit eli tässä tapauksessa $pisteet$-komennolla. Tämä toiminto ei siis laskenut mahdollista uutta pistetilannetta, vaan se ainoastaan päivitti käyttäjälle esitetyn pistetaulun ajantasalle.




Kuva 13. ActionOnImpact-toiminnon määrittäminen CopperCube-ohjelmassa.

Viimeiseksi tähän kohtaan lisättiin hienosäätöä If a Variable has a value do something -toiminnolla, joka tarkastaa halutun muuttujan, tässä tapauksessa pisteet-muuttujan ja sen sillä hetkellä olevan arvon. Jos muuttujan arvo oli tässä vaiheessa määrittelemätön, kuten vielä esimerkiksi pelisession alussa, sen arvoksi asetettiin nolla. Tämä tehtiin sen takia, että kun ensimmäinen pallo osui johonkin muuhun kuin maaliin ja 2D Overlay -teksti päivitettiin, se tarvitsi jonkin asetetun arvon näkyäkseen pistetaulussa.

Tässä vaiheessa kameran eli pelaajan kaikki halutut toiminnot oli määritelty, eli kun pelaaja painaa hiiren vasenta painiketta, pelaaja heittää pallon, joka päivittää pistetilanteen osuessaan johonkin, soittaa heittoäänen ja tarkastaa, onko pistemuuttujalla jo arvoa. Tämän lisäksi kameraan määriteltiin myös toinen

näppäintoiminto, joka oli hyppäämistä varten luotu ääni, kun painetaan välilyöntiä. Se luotiin samalla tavalla kuin pallon heittoääni.

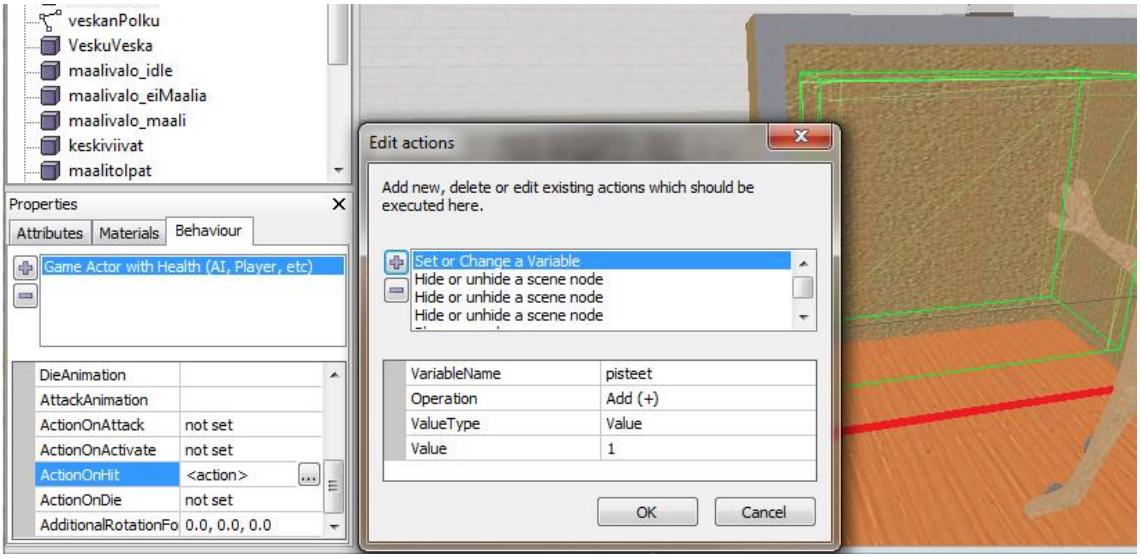
Maalin tunnistus

Kun pelaaja heitti pallon maalia päin ja se meni maaliin, silloin pelin täytyi tunnistaa tapahtuma ja laskea uusi pistetilanne. Tämä onnistui siten, että maali-objekteille annettiin tekoälytoiminnot, jotka käsittelivät niitä pelihahmoina, jotka eivät tehneet mitään mutta joihin voitiin osua pelivälineenä toimivalla pallolla ja sitten määrittää, mitä siitä seurasi. Käytännössä käytettiin CopperCuben tarjoamaa pelihahmojen tekoälyä, joka oli luotu juuri pelejä varten, mutta riisuttiin se hahmoriippuvaisista toiminnoista, kuten liikkumisesta.

Näin pallo tunnisti, kun se osui maalin varaamaan alueeseen pelinäkymässä. Tämä alue oli maalin bounding box -alue, joka tarkoittaa objektin varaamaa näkymätöntä aluetta, jonka avulla voidaan testata eri objektien välisiä törmäyksiä [26]. Tällaisenaan se oli kuitenkin todella kömpelö ratkaisu, koska nyt maali voitiin tehdä mistä tahansa kulmasta heitettynä, vaikka pallo ei mennyt itse maalin sisään. Käytännössä pelaaja pystyi menemään maalin taakse ja heittämään pallon sen takaseinään ja tekemään sitä kautta maalin. Tämän takia maalit tuotiin lopulta koko urheiluhallin mukana yhtenä objektina ja jälkeenpäin maaleista tuotiin vielä erilliset versiot.

Tässä vaiheessa siirrettiin erilliset maalit oikeille kohdilleen jo pelikentällä olevien maalien sisään ja skaalattiin ne sopivan kokoisiksi niin, että ne mahtuivat alkuperäisten maalien sisään. Tämän jälkeen erilliset maalit muutettiin näkymättömiksi ottamalla niiden Visible-valinta pois päältä. Näin luotiin maalintekoa varten omat bounding box -objektit, jotka eivät näkyneet itse pelinäkymässä.

Seuraavaksi objektit muutettiin myös pelihahmoiksi Behaviour-välilehden kautta antamalla nille tekoäly Game Actor with Health -toiminnolla. Sen asetuksista poistettiin liikkuminen asettamalla liikkumisnopeus nollaan ja maalinteon tunnistaminen lisättiin ActionOnHit-ominaisuuteen. Kun pallo osuu tähän maalin bounding box -alueeseen, haluttiin, että pisteitä tuli yksi lisää. Tätä varten lisättiin Set or Change a Variable -toiminto, johon annettiin aikaisemmin luotu pisteet-muuttuja ja määritettiin se kasvamaan yhdellä ylöspäin kuten kuvassa 14 on esitetty.




Kuva 14. Behaviour-välilehden asetuksia maalin bounding box -alueelle.

Lisäksi maaleille lisättiin myös maalivalojen muutokset tässä kohtaa Hide or unhide a scene node -toiminnolla. Siihen voitiin määrittää, muuttuuko jokin objekti näkyväksi vai näkymättömäksi eli tässä tilanteessa, mitkä maalivalot muuttuivat näkymättömiksi ja mikä muuttui näkyväksi, kun pallo meni maaliin. Maalin tapahtuessa haluttiin punainen maalivalo päälle maalin merkiksi, joten sen arvo muutettiin näkyväksi ja muut valot näkymättömäksi. Vastaavasti lisättiin myös maalitorven ääni, kun pallo osuu maalin bounding box -alueeseen.

Maalivahdin liikkuminen ja osuman tunnistus

Maalivahti saatiin liikkeelle lisäämällä sille oma polku, jota se voitiin laittaa seuraamaan. Tämä onnistui Create a path -työkalulla, jolla pystyttiin luomaan halutut välietapit polulle ja muokkaamaan polun liikerataa. Polusta tehtiin yksinkertainen suora polku, jossa maalivahti tuli liikkumaan oikealta vasemmalle ja takaisin maalin edessä, tarkoituksenaan estää pallon lentorata maaliin.

Seuraavaksi maalivahti liitettiin tehtyyn polkuun valitsemalla maalivahti ja lisäämällä Follow a Path -toiminto Behaviour-välilehdeltä. Toiminnon PathToFollow-kohtaan laitettiin luotu polku, joka tässä tapauksessa oli nimeltään veskanPolku. Tämän jälkeen maalivahti lukittui kiinni valittuun polkuun ja alkoi liikkua edestakaisin maalin edessä.

Kun heitetty pallo osui maalivahtiin, haluttiin, ettei pallo jatkanut matkaansa, maalivalo näytti sinistä valoa torjunnan merkiksi ja kuultiin torjuntaääni. Toteutus tehtiin samaan

tapaan kuin maalien tunnistamisessa eli lisättiin Behaviour-välilehdestä maalivahdin olevan pelihahmo ja asetettiin sen liikkumisnopeus myös nollaksi, koska maalivahdille luotiin jo oma hallittu liikkuminen polun avulla. ActionOnHit-ominaisuuteen määriteltiin, mitä tapahtuu, kun pallo osuu siihen. Näitä asioita olivat sinisen maalivalon muuttaminen näkyväksi ja muut maalivalot näkymättömiksi Hide or unhide a scene node -toiminnolla sekä äänitiedoston lisääminen Play a sound -toiminnolla.

Pisteiden laskeminen

Pistetilanteen näyttämistä varten luotiin 2DOverlay-objekti pelinäkymän vasempaan ylänurkkaan, ja siihen lisättiin perusteksti pelin alkua varten Attributes-välilehdellä. Tämä objekti toimi pelissä pistetauluna samaan tapaan kuin televisiourheiluissa näytetään pistetilanne näytöllä.

Pelin edetessä haluttiin, että näytettävä pistetilanne päivittyi sitä mukaa kuin maaleja tehtiin. Tätä varten luotiin aikaisemmin mainittu muuttuja nimeltä pisteet. Muuttujan arvon päivittäminen tehtiin siis maalien bounding box -ominaisuuksien asetuksissa pallon osuttua niihin. Itse pistetaulu puolestaan päivittyi aikaisemmin palloon lisätyllä ActionOnImpact-toiminnolla, joka päivitti pistetaulun tekstin kokonaan. Tällä toteutustavalla koko tekstikenttä kirjoitettiin aina uudestaan eikä vain päivitetty pisteet-muuttujaa siinä.

Julkaiseminen

CopperCube tarjosi neljä erilaista julkaisuformaattia 3D-pelin julkaisemiseksi. Mukana oli kahdelle eri käyttöjärjestelmälle tehdyt standalone-sovellukset ja kaksi eri vaihtoehtoa verkkojulkaisuja varten.

Näistä taulukon 3 vaihtoehdoista valittiin WebGL. Sen asetuksista voitiin määrittää pelin käyttämän canvas-elementin koko verkkoselaimessa ja se, kuinka nopeasti sitä päivitettiin. Kun peli julkaistiin, se avautui itsestään verkkoselaimessa ja loi tallennuspaikkaan sekä omat tiedostonsa että HTML-sivun, jolla 3D-peli oli valmiina pelattavaksi.

Taulukko 3. CopperCube-ohjelman tukemat julkaisuformaatit [16].

| **Julkaisuformaatti** | **Tiedostopääte** | **Käyttöalusta** |
|---|---|---|
| WebGL | .html | verkkoselain |
| Flash | .swf | verkkoselain |
| Windows | .exe | erillinen sovellus |
| Mac OS X | .app | erillinen sovellus |

WebGL ei ohjelmistokehyksen versiosta riippuen välttämättä toiminut suoraan omalta tietokoneelta verkkoselaimien turvallisuusasetuksien takia. Tämä voitiin kuitenkin kiertää väliaikaisesti vaihtamalla selaimen turvallisuusasetuksia, mikä ei kuitenkaan ole yleisesti suositeltavaa. Vastaavasti julkaistu sovellus voitiin viedä omalle verkkopalvelimelle internetiin ja avata sieltä.

Selaimen turvallisuusasetuksen pystyi muuttamaan Firefoxissa kirjoittamalla osoiteriville about:config, jolla päästiin verkkoselaimen asetuksiin, ja sitten muuttamalla security.fileuri.strict_origin_policy-asetuksen arvo niin, että se antoi luvan suorittaa WebGL:ää paikallisesti suoraan omalta tietokoneelta. Google Chromessa vastaavasti täytyi lisätä ”--allow-file-access-from-files”-parametriksi selaimen pikakuvakkeeseen, jotta se salli paikallisesti ajettavat tiedostot. Kun WebGL:ää ei enää tarvittu, oli hyvä muistaa vaihtaa arvo takaisin oletusarvoonsa tai poistaa parametri käytetystä tavasta riippuen, ettei tietokone altistuisi turvallisuusuhille. CopperCuben 3.0.2-versiosta lähtien tätä toimenpidettä ei ole enää tarvittu.

### 4.1.3 Pelin CopperLicht-versio

Toinen mahdollinen tapa pelin rakentamiseksi oli Ambieran valmistaman CopperLicht-ohjelmistokehyksen avulla (http://www.ambiera.com/copperlicht). CopperCube-ohjelmasta poiketen se oli ilmainen WebGL-ohjelmistokehys, mutta siinä ei vastaavasti ollut graafista käyttöliittymää tai editoria 3D-mallien ja toimintojen hallitsemista varten.

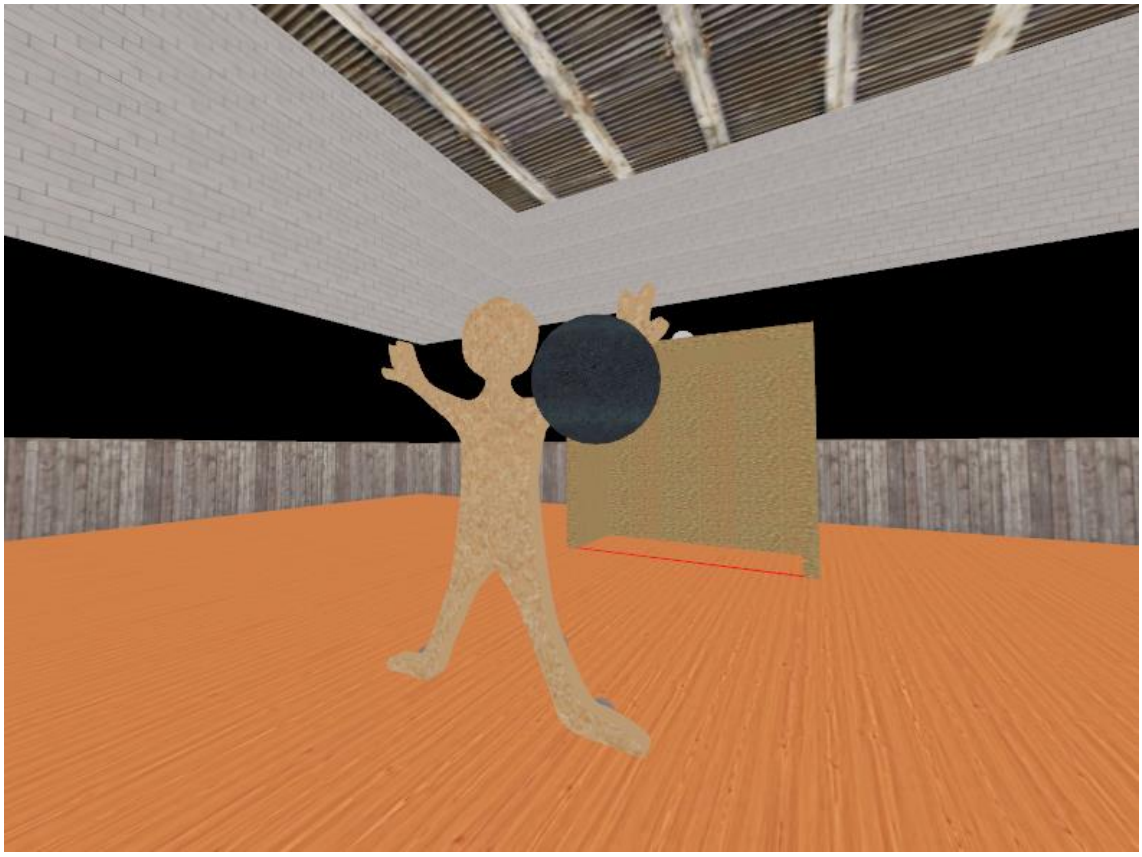
Sen sijaan kaikki nämä tehtiin käyttäen apuna JavaScript-koodia. Tässä käytännössä kuitenkin tarvittiin CopperCubea välikätenä, jotta halutut 3D-mallit saatiin käsiteltäviksi CopperLichtiin. Tämä johtui siitä, että itsessään CopperLicht osasi ladata vain ccbjs-tiedostoja, joita CopperCube tuotti. Jos esimerkiksi haluttiin tuoda 3DS-tiedosto suoraan CopperLichtiin, sille olisi pitänyt kirjoittaa oma latausfunktio, joka käsittelee 3D-tiedoston datan tuonnin ja käyttöönoton. [27.]

CopperLichtin kehittäjän, nimimerkki niko, mukaan ccbjs-tiedostojen suosimiseen päädyttiin, koska ne olivat kevyempiä ja pienempiä tiedostoja kuin alkuperäiset 3D-mallien tiedostot, mikä vähentää JavaScriptin tarvittua määrää ja näin ollen parantaa niiden suorituskykyä sekä vähentää ladattavan tiedon määrää käyttäjälle ja palvelimelle. [27.]

Prosessina tämä oli sama siihen asti, kun 3D-mallit tuotiin CopperCube-ohjelmaan, mutta sen sijaan, että niitä alettiin käsitellä CopperCubessa, ne julkaistiin välittömästi WebGL:nä. Tämä loi tarvittavan ccbjs-tiedoston, joka piti kaiken olennaisen tiedon sisällään. Kaikki tästä eteenpäin tehtiin CopperCuben ulkopuolella käyttäen juuri luotua tiedostoa. Tiedoston muokkaamiseen käytettiin Notepad++-ohjelmaa, joka on tekstinmuokkaus- ja koodausohjelma. Kaikki muokkaus tehtiin myös CopperCuben luomaan HTML-sivuun, johon edellä mainittu ccbjs-tiedosto ladattiin.

Pelin CopperLicht-versio tehtiin todella riisutusti ja puhtaassa kokeilumielessä, jotta saatiin kosketusta siihen, miten pelin tai jonkin muun näytteillepanon luominen aloitettaisiin koodipuolella. Näin ollen 3D-pelin CopperLicht-versio perustui Ambieran esimerkkien pohjalle, ja sen lähdekoodiin merkittiin koodin eri kohtien toiminnot ja toimintaperiaatteet.

Kuvassa 15 esitellään saman urheilupelin versiota CopperLichtin avulla. Ulkonäöllisesti se ei juurikaan poikennut CopperCube-pelistä, mutta eroavaisuudet tulivat vastaan sen hieman toisenlaisessa toimivuudessa. Hienosäätämällä CopperLicht-version toiminnot olisi saatu yhtä sulaviksi kuin CopperCube-versiossa. Toteutustapa oli myös täysin erilainen, koska CopperLicht-version tekemiseen ei juurikaan käytetty CopperCubea aloituksen jälkeen.



Kuva 15. Maalivahdin torjunta CopperLicht-version pelitilanteessa.

Kuvan 15 pelinäkymässä näkyvät urheiluhalli monine osineen, maalivahti, peliväline eli pallo ja myös pelihahmona toimiva kamera, josta päin kuvattuna pelinäkymä on. Kaikki nämä tuotiin alussa CopperCubeen, minkä jälkeen ne julkaistiin suoraan WebGL:ksi. Käytännössä urheiluhalli oli semmoisenaan valmiina paikallaan, mutta maalivahti, peliväline ja kamera asetettiin paikoilleen JavaScriptin avulla.

Esimerkkikoodissa 1 oli alkujaan CopperCuben luoma HTML-sivun pohja, jota on muokattu edelleen halutun laiseksi. Head-elementissä haettiin CopperLichtin JavaScript-tiedosto, joka sisälsi kaikki CopperLicht-ohjelmistokehyksen ominaisuudet, joita sen toimimiseen tarvittiin. Canvas-elementillä puolestaan luotiin piirtoalusta, johon ladattiin CopperCuben luoma ccbjs-tiedosto startCopperLichtFromFile-komennolla. Tämän jälkeen peliin oli mahdollista lisätä haluttuja objekteja ja toimintoja JavaScript-komennoilla.

```
<html>
<head>
  <script type="text/javascript" src="copperlichtdata/copperlicht.js"></script>
</head>
<body>
  <canvas id="3darea" width="800" height="600" style="background-color:#000000"></canvas>
  <script type="text/javascript">
    var engine = startCopperLichtFromFile('3darea', 'copperlichtdata/pelikaukalo_copperlicht.ccbjs');
    engine.OnLoadingComplete = function() {
      var scene = engine.getScene();
      if (!scene)
      return;
      var maalivahti = scene.getSceneNodeFromName('veska');
      maalivahti.Pos.X = -50;
      maalivahti.Pos.Y = 10;
      maalivahti.Pos.Z = 0;
    }
  </script>
</body>
</html>
```

Esimerkkikoodi 1. CopperLicht-sivulle ladattu ccbjs-tiedosto ja maalivahdin asettaminen.

Urheiluhalli oli jo paikallaan CopperCube-ohjelmasta julkaistessa, mutta esimerkiksi maalivahti saatiin paikoilleen maalin eteen hakemalla se aluksi getSceneNodeFromName-komennolla, jolla pystyttiin hakemaan mitä tahansa mukana ollutta objektia sen nimellä. Maalivahti-objektin nimi oli veska, jolla se kutsuttiin ja asetettiin maalivahti-muuttujaan. Tämän avulla maalivahti saatiin haluttuun paikkaansa maalivahti.Pos.X-komennon avulla, jolla pystyi liikuttamaan maalivahtia x-akselilla. Y- ja z-akseleille tehtiin samat määritykset, ja näin maalivahti oli maalin edessä.

Esimerkiksi kamera luotiin samaan tapaan. Ensiksi haettiin haluttu kameratyyppi sille kuuluvalla komennolla ja asetettiin sillekin aloitussijainti, minkä jälkeen sen liikkumisnopeus ja tarvittaessa muitakin ominaisuuksia voitiin asettaa.

Lyhyesti sanottuna vastaavanlaisilla JavaScript-komennoilla saatiin sekä järjestettyä 3D-objekteja että lisättyä niihin toimintoja. Ambieran sivuilla on melko kattava dokumentaatio eri komennoista ja niiden toiminnoista, joita CopperLichtissä voidaan käyttää.

## 4.2 X3DOM-esimerkit

### 4.2.1 Perusrakenne

Esimerkkinä käytetty HTML5-sivu koostui peruselementeistä ja noudatti tavallista syntaksia, mutta lisänä siihen tuli X3DOMia varten X3D-elementtejä eli X3D-noodeja, joita hyödynnettiin tuomaan ja määrittämään 3D:tä suoraan verkkosivulle [20]. Näiden esimerkkien tarkoitus oli esitellä X3DOMin peruspiirteet.

Kun sivua tehtiin HTML-sivuna, tuli muistaa, että kaikki käytettävät noodit eivät toimineet itsestään sulkeutuvilla tageilla, kuten esimerkiksi viewpoint- ja materiaali-noodit [28].

Yksinkertaisen sivun lähdekoodi näyttää esimerkkikoodin 2 tapaiselta. Se koostui tavalliseen tapaan HTML-elementistä, jonka sisässä olivat sekä head- että body-elementit. Jotta X3DOM pystyi toimimaan, se tarvitsi oman JavaScript-tiedostonsa ja lisäksi CSS-tyylitiedoston, jotka tuotiin head-elementissä [29].

```
***x3dom.html
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" type="text/css" href="http://www.x3dom.org/download/x3dom.css" />
  <script type="text/javascript" src="http://www.x3dom.org/download/x3dom.js"></script>
</head>
<body>
  <x3d width="400px" height="400px">
    <scene>
      <viewpoint position="0 0 10"></viewpoint>
      <shape>
        <appearance>
          <material diffuseColor="green"></material>
        </appearance>
        <box></box>
      </shape>
    </scene>
  </x3d>
</body>
</html>
```

Esimerkkikoodi 2. Yksinkertainen X3DOM-rakenne HTML-sivulla.

Itse 3D-sisältö aloitettiin X3D-noodilla, joka määritti suorakulmaisen piirtoalustan, joka tuli sisältämään kaikki 3D-objekteja varten tarvittavat perustiedot ja noodit [29]. Lisäämällä siihen atribuutteja voitiin määrittää tarkemmin haluttu 3D-alue, kuten esimerkiksi alueen leveys ja korkeus. Ne saatiin määritettyä myös näppärästi tyylitiedostoon [30].

Seuraavaksi tuli scene-noodi, jolla ilmoitettiin, että oltiin luomassa 3D-näkymää [29]. Tässä vaiheessa oli hyvä määrittää, mistä päin luotua 3D-näkymää tarkasteltiin. Se onnistui viewpoint-noodilla, joka toimi kuin kamera [31]. Se voitiin asettaa haluttuun paikkaan antamalla sille position-attribuutti.

Seuraavaksi määriteltiin luotavaksi geometriaa, mikä onnistui shape-noodilla. Tämän noodin sisään tuli kaikki tarvittava tieto siitä objektista, joka haluttiin saada sivulle. Sen ulkonäkö voitiin määrittää appearance-noodilla, jonka avulla määriteltiin esimerkiksi objektin materiaalin arvoja ja tekstuureja. [29]. Tähän asetettiin objektin materiaalin väri vihreäksi määrittämällä material-noodin diffuseColor-attribuutti kyseiseksi väriksi.

Tässä oli jo melkein kaikki tarvittava, mutta kaikkein tärkein puuttui vielä, itse objekti eli nähtävä 3D-malli. Yksinkertaisen mallin pystyi luomaan suoraan kirjoittamalla halutun noodin nimen ja sulkemalla sen tagi [29]. Esimerkissä haluttu objekti oli laatikko, joka onnistui käyttämällä box-noodia. Vastaavasti olisi voitu käyttää esimerkiksi sphere- tai cone-noodeja tai muita vastaavia geometrisiä kuvioita [29].

Tässä oli kaikki, mitä tarvitiin yksinkertaisen X3DOM-sivun luomiseen. Tehtyä laatikkoa pääsi nyt tarkastelemaan WebGL:ää tukevan verkkoselaimen avulla.

Vaativammat 3D-mallit on käytännössä aina tuotu jostakin 3D-mallinnusohjelmasta, koska 3D-malleissa on niin paljon tietoa, ettei niiden käsinkirjoittamisessa olisi mitään järkeä. Kaikki monimutkaisemmatkin sivut 3D-malleineen kuitenkin toimivat tällä samalla periaatteella ja perusrungolla. Lisänä on vain enemmän tarkentavia noodeja, attribuutteja arvoineen ja JavaScriptiä niiden mahdollisten interaktioiden luomiseksi.

Jos kyseessä oli vastaavasti XHTML-sivu, se oli käytännössä melkein samanlainen kuin HTML-versio, mutta siinä oli muutamia poikkeavuuksia. Esimerkiksi X3D-noodien

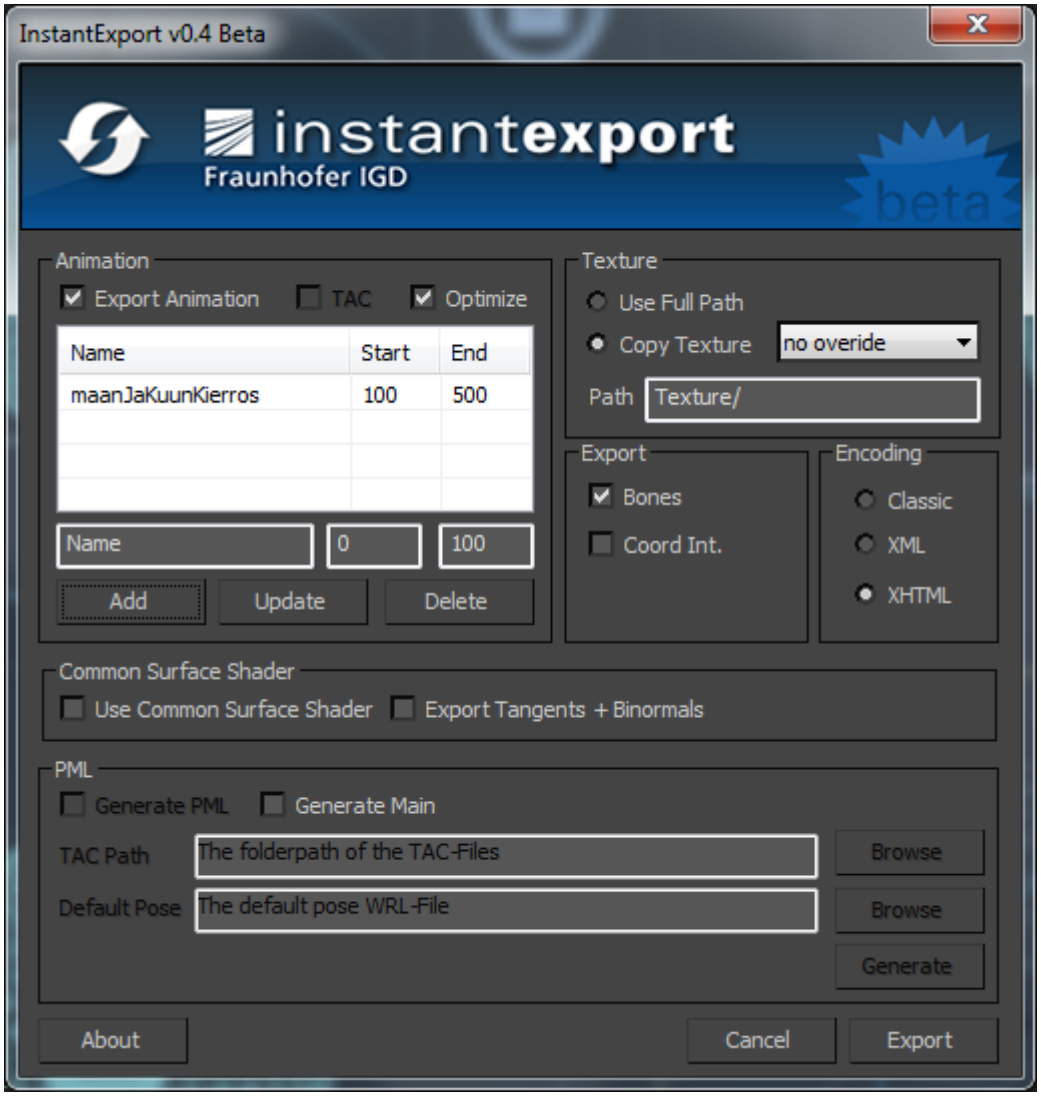
kirjoitusasussa oli oltava tarkka, koska monet niistä vaativat, että ne oli kirjoitettu oikein isoja ja pieniä kirjaimia myöten. Jos aloitettiin esimerkkikoodin 2 mukaan <x3d>-tagilla, ei sivu olisi tullut näyttämään haluttua aluetta lainkaan, vaan se täytyi kirjoittaa malliin <X3D>. Itsestään sulkeutuvat tagit puolestaan toimivat XHTML-sivulla. [32]. Autodesk 3ds Max -ohjelmassa oleva InstantExport-liitännäinen tuotti sivut juuri XHTML-muodossa [33].

### 4.2.2 3D-mallin vieminen ulos Autodesk 3ds Max 2012 -ohjelmasta

Kun haluttu 3D-malli luotiin 3ds Maxissa, se täytyi seuraavaksi saada siitä ulos oikeassa muodossa X3DOMia varten. Tämä onnistui käyttämällä InstantRealityn tekemää X3D-viejää nimeltä InstantExport (http://www.instantreality.org/downloads/dailybuild/?dir=/InstantExport). Se on 3ds Maxiin saatava liitännäinen, joka löytyy viemisvaihtoehdoista.

Kun vieminen aloitettiin, aukesi eteen ikkuna, jossa valittiin viemisasetukset. Aluksi valittiin kansio, johon malli haluttiin viedä. Tämän jälkeen valittiin tiedostotyypiksi InstantExport (*.WRL, *.XHTML, *.X3D), annettiin sille osuva nimi ja siirryttiin seuraavaan kohtaan tallennuspainikkeella.

Kuvassa 16 on esitetty viemisasetukset, kun mukana oli myös animaatiota. Encoding-kohtaan valittiin joko XHTML tai XML riippuen lopputuotteen halutusta tarkoituksesta ja esillepanotyylistä. XHTML-vaihtoehto teki uuden XHTML-sivun, jossa malli oli suoraan sivulle upotettuna, mukanaan kaikki mahdolliset tekstuureista mahdollisiin animaatioihin. XML-vaihtoehto puolestaan teki erillisen X3D-tiedoston, josta löytyivät kaikki vastaavat tiedot, mutta se täytyi kutsua sivulle erillisen inline-komennon avulla.



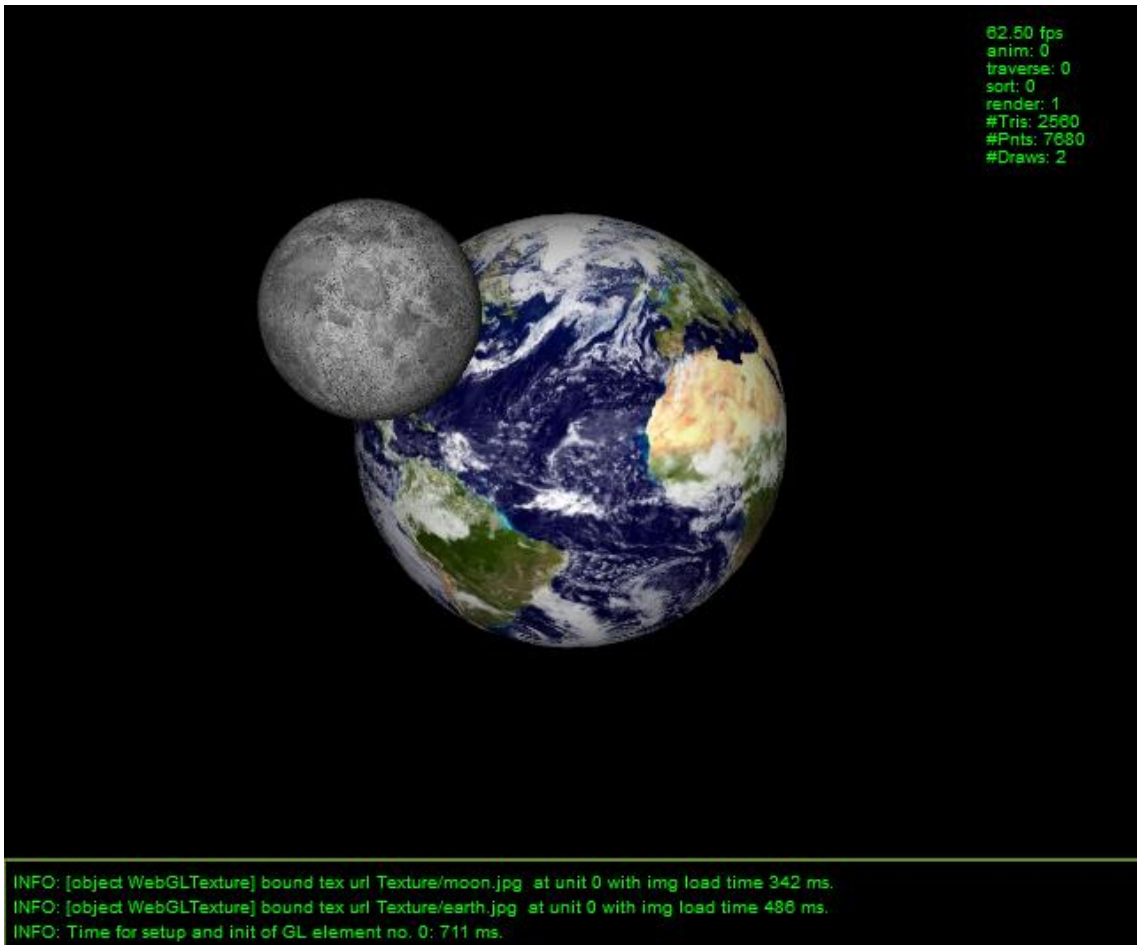

Kuva 16. InstantExport-asetukset.

Yksinkertaisimmillaan kaikki tarvittavat asetukset olivat tässä. Jos mukana oli animaatiota, valittiin Export Animation -valinta, annettiin animaation kohdalle tai kohdille nimet ja alkamis- sekä päättymishetket. Tämän jälkeen painettiin vielä Add-painiketta, jolloin se ilmestyi animaatiolistaan.

Viemisprosessi suoritettiin loppuun painamalla Export-painiketta. Valittuun kansioon ilmestyi tässä esimerkissä animaatiolla varustettu XHTML-tiedosto ja kansio käytettyjä tekstuureita varten.

Kuva 17 näyttää loppunäkymän verkkoselaimessa kun 3D-animaatio vietiin ulos 3ds Maxista. Animaatio toimii siis nyt suoraan verkkoselaimeen upotettuna. Tämän lisäksi

kuva 17 esittelee myös tieto- ja lokinäkymät, jotka antoivat tietoa tapahtuneista asioista WebGL-esityksessä.




Kuva 17. Animaatio toiminnassa verkkoselaimessa.

### 4.2.3 Inline-noodi

Yksi muista mukana olleista esimerkeistä koski inline-komentoa, jolla voitiin tuoda ulkoisia X3D-tiedostoja X3DOM-sivurakenteeseen pelkän URL-viitteen avulla [34]. Tämä tapa oli erittäin hyödyllinen lähdekoodin pitämiseksi siistinä X3DOM-sivulla, ja se myös auttoi 3D-mallien hallinnoimista, kun ne pystyttiin jakamaan erillisiin paikkoihin, joista ne haettiin sivulle kyseisellä komennolla.

Esimerkkikoodissa 3 on tavallinen X3DOM-sivun rakenne, johon tuotiin inline-noodilla ulkoinen X3D-tiedosto nimeltä veska.x3d, joka sisälsi kaikki 3D-mallin tiedot ja määrittelyt. Yhteensä tiedostossa oli 515 riviä koodia, koska 3D-malli oli melko

monimutkaisen muotoinen. Sen sijaan, että 3D-malli olisi sulautettu suoraan sivulle ilman inline-noodia, se saatiinkin siihen siististi yhdellä rivillä koodia.

```
***inline_esimerkki.xhtml
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <link rel="stylesheet" type="text/css" href="http://www.x3dom.org/download/x3dom.css" />
  <script type="text/javascript" src="http://www.x3dom.org/download/x3dom.js"></script>
</head>
<body>
  <X3D width="400px" height="400px" >
    <Scene>
      <Viewpoint position="0 0 15" />
      <Transform>
        <Inline DEF="veska" url="veska.x3d" />
      </Transform>
    </Scene>
  </X3D>
</body>
</html>
```

Esimerkkikoodi 3. Inline-noodin käyttö X3DOM-sivun rakenteessa.

Jos 3D-mallille tuli tarvetta tehdä mallinnuksellisia muutoksia, sekin onnistui helposti 3ds Max -ohjelmassa, minkä jälkeen sieltä tuotiin uudestaan ulos X3D-tiedosto ja korvattiin sillä vanhempi versio. Itse sivuun ei tällä menettelyllä tarvinnut koskea lainkaan.

# 5 WebGL:n analyysi ja tulevaisuus

Verkkoselaimet

Yhtenä WebGL:n tavoitteena on ollut, että se voisi olla WWW:n standardi ja helppo käyttää ilman käyttäjän toimenpiteitä tai selaimeen asennettavia liitännäisiä [5]. Useimmissa tapauksissa tämä onkin onnistunut, mutta edessä on vielä haastavia kysymyksiä ja ongelmia, joita pitäisi pystyä ratkaisemaan.

Taulukko 4 näyttää niiden verkkoselaimien tämänhetkisen tilanteen, jotka tukevat WebGL:ää. Internet Explorer on vielä tänäkin päivänä yksi maailman käytetyimmistä verkkoselaimista, joten se on erittäin tärkeässä roolissa kehittäjien ja käyttäjien kannalta [39]. Jos oikeasti halutaan luoda uutta standardia ja kaikille tarkoitettua palvelua, luonnollisesti tarvitaan mukaan myös yksi alan johtava yhtiö. Näin ei kuitenkaan vielä ole. Käytetyimmistä verkkoselaimista vain Microsoftin Internet Explorer ei tue WebGL:ää, ja näillä näkymin on vielä epävarmaa, muuttuuko tilanne [11].

Taulukko 4. Verkkoselaimien WebGL-tuki.

| **Verkkoselain** | **WebGL-tuki** |
|---|---|
| Google Chrome | tukee natiivisti versiosta 9.0 eteenpäin [35] |
| Internet Explorer | mikään versio ei tue [11] |
| Mozilla Firefox | tukee natiivisti versiosta 4.0 eteenpäin [36] |
| Safari | versio 5.1 tukee vain OS X 10.6:ssa tai uudemmissa käyttöjärjestelmissä; manuaalinen aktivointi [37] |
| Opera | tukee 12.0 alpha -versiosta eteenpäin [38] |

Ongelma Internet Explorerin ja WebGL:n välillä on toistaiseksi ratkaistu kahdella eri tavalla. Ensimmäinen tapa on käyttää IEWebGL-liitännäistä, joka lisää Internet Explorer -selaimeen tuen WebGL:n toimintaa varten [40]. Toinen tapa puolestaan vaatii

Chrome Frame -liitännäisen, joka toistaa WebGL:ää Internet Explorer -selaimessa avaamalla sen Google Chrome -ikkunan sisään [41].

Chrome Frame vaatii toimiakseen WebGL-sivulla olevan pienen metatiedon, jotta Internet Explorer osaisi avata sen Chrome Frame -ikkunaan. Esimerkkikoodissa 4 on esitelty sen käyttöönotto X3DOM-sivulla. Se on lisätty head-elementin sisään meta-elementtiin, ja verkkoselain, johon on asennettu Chrome Frame, osaa tulkita sen ja avata sivun Chrome Frame -ikkunassa. Näin saadaan käyttöön sen tarjoamat verkko- ja renderöintiteknologiat, joiden avulla esimerkiksi juuri WebGL toimii. [42.]

```
***chromeFrameEsimerkki.html
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="X-UA-Compatible" content="chrome=1">
  <link rel="stylesheet" type="text/css" href="http://www.x3dom.org/download/x3dom.css" />
  <script type="text/javascript" src="http://www.x3dom.org/download/x3dom.js"></script>
</head>
<body>
  <x3d width="400px" height="400px">
  </x3d>
</body>
</html>
```

Esimerkkikoodi 4. Chrome Frame -liitännäisen käyttöönotto X3DOM-sivulla.

Monet mobiiliselaimetkin tukevat WebGL:ää, kuten esimerkiksi Firefoxin, Safarin ja Operan selainten mobiiliversiot [43; 44]. Näiden lisäksi myös BlackBerry PlayBook, Nokia N900 ja Sony Ericsson Xperia tukevat WebGL:ää omissa selaimissaan [45; 46; 47]. Myös Android-käyttöjärjestelmän verkkoselain tukee WebGL:ää monissa Android-laitteissa [43].

Mielipide

WebGL on omasta mielestäni onnistunut tarkoituksessaan tuoda interaktiivista 3D:tä verkkoselaimiin ilman ylimääräisiä liitännäisiä. WebGL itsessään tarjoaa laajan alustan, jolle voidaan tehdä monenlaisia eri 3D-sovelluksia aina peleistä hyödyllisiin esillepanoihin asti ja käytännössä mitä vain niiden väliltä.

WebGL tarjoaa myös riittävän monimuotoisen ja tehokkaan alustan vaativillekin 3D-verkkosovellusten ystäville, mutta se on myös monille liian hankalan tuntuista päästä sisään. Tämän vuoksi sille onkin kehitetty useita WebGL-ohjelmistokehyksiä, jotka tarjoavat helpomman väylän WebGL:ään JavaScriptin avulla. Näin käyttäjän ei tarvitse olla tekemisissä esimerkiksi GLSL-kielen tai muiden vaativimpien WebGL-käskyjen kanssa.

WebGL tarjoaa HTML5:n myötä erinomaisen kanavan tuoda internetin käyttäjille entistä näyttävämpää grafiikkaa ja interaktioita, mikä on internetin kehityksessä mielestäni olennaista. Tällaisten tekniikoiden avulla voidaan ehkä saada varteenotettava tapa luoda graafisia ohjelmia suoraan verkkoselaimiin, samaan tapaan kuin monet helpommin toimivat ohjelmat ovat jo löytäneet pilvipalveluihin.

Nyt jo esimerkiksi Google käyttää WebGL:ää karttapalvelunsa MapsGL-versiossa. Myös Nokialla on oma WebGL:ää käyttävä karttapalvelu. Näiden lisäksi WebGL:n avulla on luotu esitys ihmiskehon anatomiasta, jota pääsee katselemaan hyvin tarkasti ja voi rajata näkymät eri anatomian osiin. Nämä ovat vain alkuesimerkkejä kaikista mahdollisuuksista, joita WebGL:n avulla voidaan tuoda sivuille. Odotan mielenkiinnolla ja innolla, miten WebGL kehittyy ja mitä kaikkea sen avulla voidaan tehdä.

Palaute virtuaalitekniikoiden opintojaksolta

WebGL oli osana myös Metropolia Ammattikorkeakoulussa tarjottavaa virtuaalitekniikoiden opintojaksoa. Tämän työn 3D-peli ja esimerkit toimivat opintojaksossa opetusmateriaalina tarkoituksena tutustuttaa opiskelijat teknologian käyttöön ja mahdollisuuksiin. CopperCube-peli ja esimerkit saivat myönteistä palautetta opintojakson opiskelijoilta. CopperCube-pelistä sanottiin, että se muun muassa auttoi hahmottamaan, mitä CopperCuben avulla pystyi tekemään.

Toisaalta CopperCube sai myös kritiikkiä, koska joidenkin opiskelijoiden mielestä se keskittyi liikaa pelien tekemiseen eikä niinkään muuhuun sisältöön. Tämä on totta, ja ohjelmaa onkin enemmän markkinoitu juuri pelien ja muiden vastaavien tuottamiseen, vaikka sillä pystyy toisenkinlaisiakin tuotoksia tekemään.

## Tulevaisuus

WebGL on ollut tulemassa jo jonkin aikaa, ja sen on ollut tarkoitus tulla WWW:n standardiksi 3D:n tuottamisessa verkkoselaimeen HTML5:n tulemisen yhteydessä. Sen tulevaisuus koki kuitenkin jo alkuvaiheissa takaiskun, kun Microsoft ilmoitti, että Internet Explorer -verkkoselain ei tukisi WebGL:ää. Jos tähän ei ole tulossa muutosta, se voi mahdollisesti estää WebGL:n käytön leviämistä huomattavasti, koska ison asiakasryhmän ulkopuolelle jäänti saattaa pitää yritykset etäällä tekniikasta.

WebGL on kuitenkin tähän mennessä lupaavin tapa saada interaktiivista 3D:tä selaimeen. Se tarjoaa monille eri tahoille jotain. Loppukäyttäjät voivat käyttää sitä verkkoselaimissaan samaan tapaan kuin esimerkiksi jpg-kuvan katsomista, eli käytännössä he vain menevät sivulle, jolle se on asetettu esille. Kaikki muu tapahtuu automaattisesti, eikä käyttäjän tarvitse asentaa selaimeensa mitään. Koodaajat saavat myös käyttöönsä tehokkaat työkalut, joiden avulla grafiikkalaitteista saadaan yhä enemmän hyötyä. Myös ei niin koodauspainotteiset henkilöt saavat tehtyä monenlaisia 3D-sovelluksia tarjolla olevien WebGL-ohjelmistokehyksien avulla.

WebGL on myös vahvasti esillä, kun verkkoselaimiin tehdään selainpohjaisia hyötyohjelmia, kuten esimerkiksi WebGL:llä toimiva 3D-mallinnusohjelma 3DTin (http://www.3dtin.com). Sen avulla pystytään mallintamaan 3D-malleja suoraan verkkoselaimessa, viemään niitä sieltä ulos ja jakamaan muiden käyttäjien kesken.

Kuten näistä selainpohjaisista lähestymistavoista huomataan, ollaan menossa helposti saatavilla oleviin selainpohjaisiin ohjelmiin ja editoreihin. Niiden avulla pystytään jo nyt tekemään merkittäviä asioita, ja kehityksen myötä ne tarjoavat varteenotettavan vaihtoehdon perinteisille editoreille. Niiden näyttävissä ja monimuotoisissa toiminnoissa WebGL on tärkeä tekijä.

Kaikki tämä yhdessä tarjoaa hyvin laajan käyttäjäkunnan, joten WebGL:llä saattaa olla hyvinkin valoisa tulevaisuus, vaikka mainittuja kysymyksiä onkin. Mikään uusi teknologia ei ole varma läpimurrosta, mutta WebGL:llä on mielestäni kaikki mahdollisuudet onnistua tavoitteessaan päästä HTML5:n mukana standardiksi tuoda interaktiivista 3D:tä verkkoselaimiin.

# 6 Yhteenveto

Insinöörityön tarkoituksena oli tutkia WebGL:ää ja sen tarjoamia mahdollisuuksia tuottaa interaktiivista 3D:tä oppilaitoksen käyttämillä ohjelmilla ja saada se näkymään verkkosivuilla mahdollisimman vaivattomasti. Insinöörityön tuloksia käytetään osana Metropolia Ammattikorkeakoulun opintojaksotarjontaa, joten opintojaksoja varten tarvittiin myös esimerkkejä tästä prosessista.

Työssä tutkittiin Metropolian käyttämää Autodesk 3ds Max -mallinnusohjelmaa ja sen eri tapoja tuottaa WebGL-sisältöä verkkosivuille. Kokeilujen jälkeen päädyttiin kahteen toimivaan tapaan, jotka olivat Ambieran CopperCube ja X3DOM. 3D-mallien ja -animaatoiden vieminen onnistui vain minimaalisilla lisäyksillä oppilaitoksen jo aiemmin hankkimaan 3ds Max -mallinnusohjelmaan. Opetusta varten luotiin kaksi eri lähestymistapaa. Ensimmäinen oli laajempi 3D-peli, joka aluksi mallinnettiin ja vietiin CopperCube-ohjelmaan, jossa siihen lisättiin toiminnallisuudet ja viimeistely. Toinen oli suppeampi kokoelma yksittäisiä esimerkkejä, jotka toivat esiin päätoimintoja sekä yleisellä tasolla näkymää, miten tekniikka toimii.

Lisäksi molempia lähestymistapoja varten luotiin tarkat ohjeet kohta kohdalta, miten ne luotiin ja mitä missäkin kohdassa tapahtui. Tarkoituksena oli antaa opiskelijoille runko ja esimerkki, joka opettaa, innostaa ja antaa kosketuksen konkreettiseen tuotokseen.

Insinöörityö onnistui saavuttamaan sille ennalta asetetut kriteerit ja täyttämään opetuksessa vaadittavat sisällöt. Insinöörityön pohjalta opiskelijat saivat onnistuneesti tutustuttua aiheeseen ja luotua omia 3D-tuotoksiaan käyttäen WebGL:ää. Metropolia Ammattikorkeakouluun hankittiin myös tämän insinöörityön pohjalta Ambiera CopperCube -ohjelma, joka on nyt opiskelijoiden käytössä.

## Lähteet


1 Korpela, Jukka. 2011. HTML5 – uudet ominaisuudet. Jyväskylä: Docendo.

2 About W3C. Verkkodokumentti. World Wide Web Consortium. <http://www.w3.org/Consortium>. Luettu 9.3.2012.

3 FAQ. Verkkodokumentti. WHATWG. <http://wiki.whatwg.org/wiki/FAQ>. Päivitetty 23.2.2013. Luettu 7.3.2012.

4 Harris, Andy. 2011. HTML5 for Dummies Quick Reference. United States of America: Wiley Publishing.

5 WebGL: Frequently Asked Questions. 2010. Verkkodokumentti. Learning WebGL. <http://learningwebgl.com/cookbook/index.php/WebGL:_Frequently_Asked_Questions>. Päivitetty 31.1.2012. Luettu 25.2.2012.

6 Getting Started. Verkkodokumentti. Khronos Group. <http://www.khronos.org/webgl/wiki/Getting_Started>. Päivitetty 10.4.2011. Luettu 24.2.2012.

7 Document Object Model (DOM). 2005. Verkkodokumentti. World Wide Web Consortium. <http://www.w3.org/DOM>. 19.1.2005. Luettu 11.3.2012.

8 Caballero, Luz. 2011. An introduction to WebGL. Verkkodokumentti. <http://dev.opera.com/articles/view/an-introduction-to-webgl>. 13.10.2011. Luettu 7.3.2012.

9 Bridge, Henry. 2011. GPU acceleration + old drivers = :(. Verkkodokumentti. <http://blog.chromium.org/2011/03/gpu-acceleration-old-drivers.html>. 1.3.2011. Luettu 12.3.2012.

10 Bauman, John, Salomon, Brian. 2012. GPU accelerating 2D Canvas and enabling 3D content for older GPUs. Verkkodokumentti. <http://blog.chromium.org/2012/02/gpu-accelerating-2d-canvas-and-enabling.html>. 9.2.2012. Luettu 12.3.2012.

11 WebGL Considered Harmful. 2011. Verkkodokumentti. Microsoft TechNet. <http://blogs.technet.com/b/srd/archive/2011/06/16/webgl-considered-harmful.aspx>. 16.6.2011. Luettu 28.4.2012.

12 Forshaw, James, Stone, Paul, Jordon, Michael. 2011. WebGL – More WebGL Security Flaws. Verkkodokumentti. <http://www.contextis.com/research/blog/webgl-more-webgl-security-flaws>. 16.6.2011. Luettu 28.4.2012.

13 Thomson, Iain. 2011. Khronos defends WebGL after security attacks from Microsoft and Context. Verkkodokumentti. <http://www.v3.co.uk/v3-uk/news/2079688/khronos-defends-webgl-security-attacks-microsoft-context>. 17.6.2011. Luettu 28.4.2012.

14 WebGL Security. 2011. Verkkodokumentti. Khronos Group. <http://www.khronos.org/news/permalink/webgl-security>. Päivitetty 16.5.2011. Luettu 29.4.2012.

15 CopperLicht Features Overview. Verkkodokumentti. Ambiera. <http://ambiera.com/copperlicht/features.html>. Luettu 4.3.2012.

16 Features of CopperCube. Verkkodokumentti. Ambiera. <http://ambiera.com/coppercube/features.html>. Luettu 4.3.2012.

17 CopperLicht. Verkkodokumentti. Ambiera. <http://ambiera.com/copperlicht/index.html>. Luettu 5.3.2012.

18 CopperCube. Verkkodokumentti. Ambiera. <http://ambiera.com/coppercube/index.html>. Luettu 5.3.2012.

19 X3DOM: FAQ. Verkkodokumentti. X3DOM. <http://www.x3dom.org/?page_id=122>. Luettu 14.3.2012.

20 X3DOM: About. Verkkodokumentti. X3DOM. <http://www.x3dom.org/?page_id=2>. Luettu 14.3.2012.

21 WebGL: User Contributions, Frameworks. Verkkodokumentti. Khronos Group. <http://www.khronos.org/webgl/wiki/User_Contributions>. Päivitetty 5.4.2013. Luettu 7.4.2013.

22 X3DOM: Examples. Verkkodokumentti. X3DOM. <http://www.x3dom.org/?page_id=5>. Luettu 14.3.2012.

23 CGTextures License. Verkkodokumentti. CGTextures. <http://www.cgtextures.com>. Luettu 2.4.2012.

24 CopperCube: 3D Character Animation. Verkkodokumentti. Ambiera. <http://www.ambiera.com/coppercube/doc/cnt_animation.html>. Luettu 3.3.2012.

25 Frequently Asked Questions. Verkkodokumentti. Freesound. <http://www.freesound.org/help/faq/#licenses>. Luettu 2.4.2012.

26 Bounding box. Verkkodokumentti. Valve Corporation. <https://developer.valvesoftware.com/wiki/Bounding_box>. Päivitetty 3.4.2011. Luettu 17.4.2012.

27 Niko. 2010. CopperLicht Forum: Loading Models. Verkkodokumentti. Ambiera. <http://www.ambiera.com/forum.php?t=1046>. 4.14.2010. Luettu 22.3.2012.

28 HTML5 Hello World. Verkkodokumentti. X3DOM. <http://x3dom.org/x3dom/example/x3dom_helloWorld.html>. Luettu 15.3.2012.

29 First steps with X3DOM. Verkkodokumentti. X3DOM. <http://x3dom.org/docs/dev/tutorial/firststeps.html>. Luettu 14.3.2012.

30 Styling with CSS. Verkkodokumentti. X3DOM. <http://x3dom.org/docs/dev/tutorial/styling.html>. Luettu 14.3.2012.

31 Behr, Johannes, Jung, Yvonne. 2012. X3DOM: Getting declarative (X)3D into HTML. Verkkodokumentti. <http://www.web3d.org/realtime-3d/files/documents/S2012/x3dom_Getting-Declarative-X3D-HTML_behr.pdf>. 2012. Luettu 11.3.2012.

32 XHTML Hello World. Verkkodokumentti. X3DOM. <http://x3dom.org/x3dom/example/x3dom_helloWorld.xhtml>. Luettu 15.3.2012.

33 3ds Max Export. Verkkodokumentti. X3DOM. <http://x3dom.org/docs/dev/tutorial/max_export.html>. Luettu 17.3.2012.

34 Extensible 3D (X3D): Part 1: Architecture and base components. Verkkodokumentti. Web3D Consortium. <http://www.web3d.org/files/specifications/19775-1/V3.2/Part01/components/networking.html#Inline>. Luettu 19.3.2012.

35 Paul, Ryan. 2011. Chrome 9 goes stable with WebGL and Chrome Instant. Verkkodokumentti. <http://arstechnica.com/information-technology/2011/02/chrome-9-goes-stable-with-webgl-and-chrome-instant>. 4.2.2011. Luettu 28.4.2012.

36 Firefox 4 Release Notes. 2011. Verkkodokumentti. Mozilla. <http://www.mozilla.org/en-US/firefox/4.0/releasenotes>. 22.3.2011. Luettu 28.4.2012.

37 New in OS X Lion: Safari 5.1 brings WebGL, Do Not Track and more. 2011. Verkkodokumentti. FairerPlatform. <http://fairerplatform.com/2011/05/new-in-os-x-lion-safari-5-1-brings-webgl-do-not-track-and-more>. 3.5.2011. Luettu 28.4.2012.

38 Kleinhout, Huib. 2011. Introducing Opera 12 alpha. Verkkodokumentti. <http://my.opera.com/desktopteam/blog/2011/10/13/introducing-opera-12-alpha>. 13.10.2011. Luettu 28.4.2012.

39 Top 5 Browsers from Jan to Mar 2013. 2013. Verkkodokumentti. StatCounter. <http://gs.statcounter.com/#browser-ww-monthly-201301-201303-bar>. Luettu 24.3.2013.

40 IEWebGL – WebGL for Internet Explorer. Verkkodokumentti. IEWebGL. <http://iewebgl.com>. Luettu 5.5.2012.

41 Google Chrome Frame. Verkkodokumentti. Google. <https://developers.google.com/chrome/chrome-frame>. Päivitetty 18.7.2012. Luettu 8.5.2012.

42 Google Chrome Frame FAQ. Verkkodokumentti. Chromium. <http://www.chromium.org/developers/how-tos/chrome-frame-getting-started/chrome-frame-faq>. Luettu 8.5.2012.

43 WebGL on Mobile Devices. 2011. Verkkodokumentti. iChemLabs. <http://www.ichemlabs.com/1375>. 12.11.2011. Luettu 11.5.2012.

44 Philip. 2012. Opera Mobile 12. Verkkodokumentti. Opera Software. <http://my.opera.com/mobile/blog/2012/02/27/opera-mobile-12>. 27.2.2012. Luettu 11.5.2012.

45 Halevy, Ronen. 2011. PlayBook OS 2.0 Developer Beta Includes WebGL, Flash 11, & AIR 3.0. Verkkodokumentti. <http://www.berryreview.com/2011/10/18/playbook-os-2-0-developer-beta-includes-webgl-flash-11-air-3-0>. 18.10.2011. Luettu 11.5.2012.

46 WebGL on N900. 2010. Verkkodokumentti. Suihkulokki. <http://suihkulokki.blogspot.com/2010/06/webgl-on-n900.html>. 7.6.2010. Luettu 11.5.2012.

47 Nilsson, Tobias. 2011. Xperia phones first to support WebGL. Verkkodokumentti. <http://blogs.sonymobile.com/wp/2011/11/29/xperia-phones-first-to-support-webgl>. 29.11.2011. Luettu 11.5.2012.