TAMPEREEN TEKNILLINEN YLIOPISTO

JARI-PEKKA VOUTILAINEN

VUOROVAIKUTTEISTEN WEB-SOVELLUSTEN KEHITTÄMINEN

Diplomityö

Tarkastajat: Tommi Mikkonen, Arto Salminen ja Matti Anttonen
Tarkastaja ja aihe hyväksytty
Tieto- ja sähkötekniikan
tiedekuntaneuvoston kokouksessa
6. huhtikuuta 2011

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
VOUTILAINEN, JARI-PEKKA: Vuorovaikutteisten web-sovellusten kehittäminen
Diplomityö, 54 sivua, 8 liitesivua
Joulukuu 2011
Pääaine: Ohjelmistotuotanto
Tarkastajat: Professori Tommi Mikkonen, assistentti Arto Salminen ja tutkija Matti Anttonen
Avainsanat: Web-ohjelmointi, HTML5, CSS, WebGL, Ajax, JavaScript

Web-ohjelmointiympäristö on kehittynyt viime vuosina nopeasti dokumenttien esitystekniikasta kohti täysivertaista ohjelmointiympäristöä, jossa voi toteuttaa ulkonäöllisesti samankaltaisia ohjelmia kuin työpöytäohjelmoinnissakin. Vielä viime vuosina monimutkaisemmat verkkosovellukset joutuivat turvautumaan liitännäiskomponentteihin toteuttaessaan interaktiivisuutta, kehittyneitä animaatioita ja audiovisuaalisen median yhdistämistä näihin. Tämä vaatii käyttäjää asentamaan liitännäisen selaimeen nähdäkseen kyseisellä teknologialla toteutetun sisällön. Tulevat HTML5 ja sen WebGL- sekä muut oheisspesifikaatiot tarjoavat standardoidut tekniikat toteuttaa liitännäisien tarjoamia toiminnallisuuksia. Näillä tekniikoilla kehittäjät voivat toteuttaa vastaavaa sisältöä ilman liitännäisiä kuin ennen liitännäisten avustuksella.

Tässä diplomityössä tutkitaan, kuinka hyvin nämä uudet teknologiat soveltuvat vuorovaikutteisen verkkosovelluksen toteuttamiseen ja esimerkkinä toteutettiin kolmiulotteinen ikkunointisovellus. Samalla selvitetään mitä ongelmia ja rajoitteita nämä teknologiat sekä selain asettavat työpöytämäisen sovelluksen toteuttamiselle. Lisäksi esitetään ratkaisuja näihin ongelmiin ja rajoitteisiin.

Teknisessä kontribuutiossa toteutettiin täysin kustomoitava 3D-ympäristö, johon voi sisällyttää olemassa olevia canvas-sovelluksia pienin muutoksin. Resurssien lataaja ja välityspalvelin mahdollistavat monipuolisten resurssien käyttämisen siten, että selaimen tietoturva on huomioitu.

Työn tuloksina havaittiin renderöintinopeuden, puutteellisten rajapintojen, JavaScript-kielen tietoturvan sekä selainten välisten toteutuserojen aiheuttavan rajoitteita työpöytämäisiä sovelluksia verkkoon toteuttaessa. Näitä rajoitteita voidaan kiertää optimoinneilla, käyttökohteeseen sopivilla kirjastovalinnoilla sekä tietoturvan huomioivien tekniikoiden käyttämisellä.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
VOUTILAINEN, JARI-PEKKA: Developing interactive web applications
Master of Science Thesis, 54 pages, 8 Appendix pages
December 2011
Major: Software Engineering
Examiners: Professor Tommi Mikkonen, assistant Arto Salminen and researcher Matti Anttonen
Keywords: Web programming, HTML5, CSS, WebGL, Ajax, JavaScript

Web programming capabilities have evolved rapidly during the last few years, from presentation of documents towards full-fledged programming environment, where it is possible to develop programs similar to the desktop environment. Until the last few years, complex web applications have had to rely on plugin components while providing interactivity, animations and combining audiovisual media with these. This requires the user to download and install the plugin to the browser to view the content.

Upcoming HTML5 standard and its WebGL and other supplementary specifications provide standardized techniques to use functionalities similar to those of plugins. With these techniques developers can implement similar content without the reliance on plugin components.

In this Master's thesis, these new specifications are studied with respect to how well they are suited for implementing interactive web applications and as a practical example three dimensional windowing environment has been implemented. The thesis also studies what problems and limitations these technologies as well as the browser impose to implementing desktop-like applications. Solutions for these problems and limitations were also addressed.

In the technical contribution fully customizable 3D-environment was implemented. Embedding existing canvas-applications is possible with only few changes to the application code. The resource loader and proxy enable usage of versatile resources so that the browser security has been noted in the design.

As the main findings, rendering speed, insufficient interfaces, security of JavaScript and differences between browser implementations all set limits to implementing desktop-like applications in the web. These limitations can be circumvented with optimizations, application based library choices, and usage of techniques that are designed with security as their main design pattern.

ALKUSANAT

Tämä diplomityö toteutettiin osana Ohjelmistotekniikan laitoksella ollutta Lively Goes 3D -projektia, jossa tutkittiin uusia selainteknologioita. Haluan kiittää työn ohjaajaa professori Tommi Mikkosta hyvistä kommenteista ja kärsivällisestä puuttuvien pilkkujen metsästyksestä. Lisäksi haluan kiittää Matti Anttosta, Arto Salmista, Niko Mäkitaloa ja Heikki Peltolaa hyvistä kommenteista ja rakentavasta palautteesta. Kiitokset myös Janne Lautamäelle, Johannes Koskiselle ja Timo Aholle hyvistä ideoista teknisen toteutuksen suhteen sekä Terhi Kilamolle, joka vihjasi mahdollisesta diplomityöpaikasta.

Lisäksi haluan kiittää Ulla-Talvikki Virtaa ja Emma Luukkaa henkisestä tuesta ja kärsivällisyydestä.

Tampereella, 16.11.2011

Jari-Pekka Voutilainen

SISÄLLYS

LYHENTEET JA TERMIT

3D-objekti	Objekti, joka koostuu mallista ja sen pinnassa käytettävästi materiaalista. Muodostaa näkymässä kolmiulotteisen kappaleen.
Ajax	Asynchronous JavaScript and XML. Kokoelma teknologioita, joilla mahdollistetaan verkkosivun osittainen päivittäminen reaaliaikaisesti ilman käyttäjän interaktiota.
CORS	Cross-Origin Resource Sharing. Standardi, joka määrittää miten verkkopalvelimen pitää tarjota resurssinsa, jotta niitä voidaan käyttää toisesta verkkolähteestä.
CSS	Cascading Style Sheets. Tyylisivukieli, jota käytetään kuvaamaan miltä merkkauskielellä kirjotettu dokumentti näyttää. Tyylisivu mahdollistaa tyylillisten sääntöjen, kuten värit, fontit, elementtien sijoittelun ja koon, määrittämisen erillään verkkosivun sisällöstä.
DOM	Document Object Model. Alustariippumaton tapa ilmaista objektien kokoelma, jotka muodostavat selaimessa esitetyn sivun. Toimii rajapintana selaimen ja skriptikielien välillä mahdollistaen joustavan kommunikaation.
HTML	HyperText Markup Language. Merkkauskieli, jolla kuvataan tekstipohjaisen dokumentin rakenne. Dokumentista merkitään osioita mm. otsikoiksi, kappaleiksi ja listoiksi. HTML kirjoitetaan tagien muodossa, jotka ilmaistaan pienempi kuin ja suurempi kuin -merkeillä.
HTTP	HyperText Transfer Protocol. Tiedonsiirtoprotokolla, joka mahdollistaa hajautetun kommunikoinnin verkon välityksellä.
JavaScript-moottori	Selaimen tai palvelimen komponentti, joka suorittaa JavaScript-koodin.
JSON	JavaScript Object Notation. Tekstipohjainen standardi, joka tarjoaa JavaScript-pohjaisen formaatin tiedon välitykseen. Standardi määrittää mallin, jolla kuvataan yksinkertaisia avain-arvo pareja.
Kamera	3D-ohjelmoinnissa kameralla tarkoitetaan katsojan sijaintia 3D-näkymässä, joka piirretään ruudulle.
Liitännäinen	Selaimeen asennettava lisäosa (engl. plugin), joka lisää selaimeen uusia toiminnallisuuksia, kuten videotoistoa ja animaatioita.

Malli	3D-ohjelmoinnissa kulmapisteistä muodostuva kappale, joka määrittää minkä muotoinen 3D-objekti on näkymässä.
Mashup-sovellus	Useista lähteistä sisältöä dynaamisesti yhdistelevä sovellus.
Mashware	Tapa kehittää Mashup-sovelluksia.
Materiaali	3D-ohjelmoinnissa kappaleiden pinnassa esitettävä materiaali. Muodostuu väreistä, tekstuureista ja valaistuksen vaikutuksesta.
Persistointi	Tilan säilyttäminen pidemmäksi aikaa kuin mitä tilan luonut prosessi on olemassa. Esimerkiksi tekstinkäsittelyohjelmissa tekstidokumentit persistoidaan levylle, joka säilyy vaikka tekstinkäsittelyohjelma suljetaan.
Renderöinti	Selaimessa renderöinnissä muodostetaan dokumentista, tyylitiedostoista ja resursseista verkkosivu. 3D-ohjelmoinnissa renderöinnissä muodostetaan malleista, tekstuureista ja valaistuksista ruudulla näytettävä kuva.
RIA	Rikas verkkosovellus (engl. Rich Internet Application). Verkkosovellus, joka on toteutettu siten, että se vaikuttaa työpöytäsovellukselta käyttöliittymältään ja toiminnaltaan.
Same Origin Policy	Selaimen tietoturvaominaisuus, joka määrittää mihin osoitteisiin JavaScript-sovellus saa tehdä kyselyitä.
Tekstuuri	Kuva tai video, jota käytetään materiaalin värin muodostamiseen.
Web 2.0	Verkkosovellukset, joissa keskitytään tiedon jakamiseen, sosiaaliseen verkostoitumiseen sekä käyttäjäkeskeiseen suunnitteluun.
WYSIWYG	What You See Is What You Get. Editori, jonka käyttöliittymässä nähdään editoitu dokumentti samanlaisena kuin mitä se on valmiina dokumenttina.
XHTML	Extensible HyperText Markup Language. XHTML on XML:n sovellus, joka laajentaa HTML:ää. XHTML-dokumentin jäsennys on tiukempaa kuin HTML:ssä.
XML	Extensible Markup Language. Standardoitu tapa esittää dokumentteja ohjelmallisesti käsiteltävässä muodossa.

1. JOHDANTO

Web-ohjelmointiympäristön kehitys voidaan jakaa sukupolviin, joissa edistys on varsinkin viime vuosina tapahtunut harppauksin. Ensimmäinen sukupolvi 1990-luvun alkupuolella koostui yksinkertaisista, staattisin kuvin varustetuista verkkosivuista. Nämä verkkosivut eivät sisältäneet animaatioita tai interaktiivista sisältöä, ja navigointi toteutettiin hyperlinkeillä, jotka klikatessa latasivat kokonaan uuden verkkosivun. Toisen sukupolven voidaan katsoa alkaneen interaktiivisuuden lisääntyessä verkkosivuilla. JavaScriptin esittely vuonna 1995 mahdollisti interaktiivisen sisällön tuottamisen helpommin, ja liitännäiskomponentit (engl. plugin), kuten Flash[1], QuickTime[2], RealPlayer[3] ja Shockwave[4], yleistyivät nopeasti. Nämä tekniikat mahdollistivat kehittyneet animaatiot, videot ja audion sisällytettäväksi verkkosivuille, ja verkkosovellukset siirtyivät perinteisistä sivuista multimediaesityksien suuntaan. Kolmas sukupolvi käsittää työpöytäsovellustyyliset verkkosovellukset, joita toteutetaan ”*Web 2.0*”-teknologioilla. Näitä sovelluksia kutsutaan rikkaiksi internet-sovelluksiksi (engl. Rich Internet Application, RIA), jotka toteutetaan mahdollistamalla yksittäisten käyttöliittymäelementtien päivittäminen koko sivun päivittämisen sijaan kun jokin muuttuu. RIA-sovelluksissa usein vältetään linkkipohjaisia navigaatioita, jonka sijaan suositaan suoria manipulointitekniikoita, jotka ovat peräisin työpöytäsovelluksista.[1]

Vielä viime vuosina monimutkaisemmat verkkosovellukset joutuivat turvautumaan liitännäiskomponentteihin toteuttaessaan interaktiivisuutta, kehittyneitä animaatioita ja audiovisuaalisen median yhdistämistä näihin. Tämä muodostaa ongelman, jossa käyttäjä joutuu asentamaan liitännäisen selaimeen nähdäkseen kyseisellä teknologialla toteutetun sisällön. Esimerkkinä mainittakoon, että toukokuussa 2011 Adobe Flash -liitännäinen oli asennettuna 95 % kaikista selaimista [2].

Tulossa oleva HTML5 ja sen jo julkaistu WebGL- sekä muut oheisspesifikaatiot tarjoavat standardoidun tavan toteuttaa liitännäisten tarjoamia toiminnallisuuksia. Nämä spesifikaatiot määrittävät kuinka selain pitää toteuttaa, jotta verkkosovelluksen kehittäjä voi toteuttaa vastaavaa sisältöä ilman liitännäisiä kuin aikaisemmin liitännäisten avustuksella.

Tässä diplomityössä tutkitaan, kuinka hyvin nämä uudet teknologiat soveltuvat interaktiivisen verkkosovelluksen toteuttamiseen ja esimerkkisovelluksena on toteutettu kol-

[1] `http://www.adobe.com/software/flash/about/`
[2] `http://www.apple.com/quicktime/`
[3] `http://europe.real.com/realplayer/`
[4] `http://www.adobe.com/products/shockwaveplayer/`

miulotteinen ikkunointisovellus. Samalla selvitetään mitä ongelmia ja rajoitteita nämä teknologiat sekä selain asettavat työpöytämäisen sovelluksen toteuttamiselle. Lisäksi esitetään, ratkaisuja näihin ongelmiin ja rajoituksiin.

Luvussa 2 esitellään kolmannen sukupolven web-ohjelmointiympäristö ja sen teknologiat. Lisäksi esitellään HTML5 ja sen oheisspesifikaatiot verkkosovelluksen kehittäjän näkökulmasta. Luvussa 3 käsitellään kirjastoja, jotka abstrahoivat teknologioiden yksityiskohtia piiloon verkkosovelluksen kehittäjältä täten yksinkertaistaen sekä nopeuttaen sovelluksen kehittämistä. Luvussa esitellään sekä selaimessa että palvelimella suoritettavia kirjastoja. Luvussa 4 kuvataan diplomityön teknisenä kontribuutiona toteutettu Lively3D-ikkunointisovellus. Sovelluksesta kuvataan yksityiskohtaisesti käytetyt teknologiat ja kirjastot. Lisäksi kuvataan sovelluksen rakenne sekä kuinka valmista web-sisältöä voidaan hyödyntää sovelluksessa. Luvussa 5 arvioidaan, kuinka hyvin kyseiset teknologiat ja kirjastot soveltuvat ikkunointisovelluksen toteuttamiseen. Lisäksi arvioidaan selaimen asettamia ongelmia ja rajoitteita sekä niiden ratkaisuja. Luvussa 6 tehdään johtopäätöksiä arvioinnin tuloksista.

2. WEB-OHJELMOINTIYMPÄRISTÖ

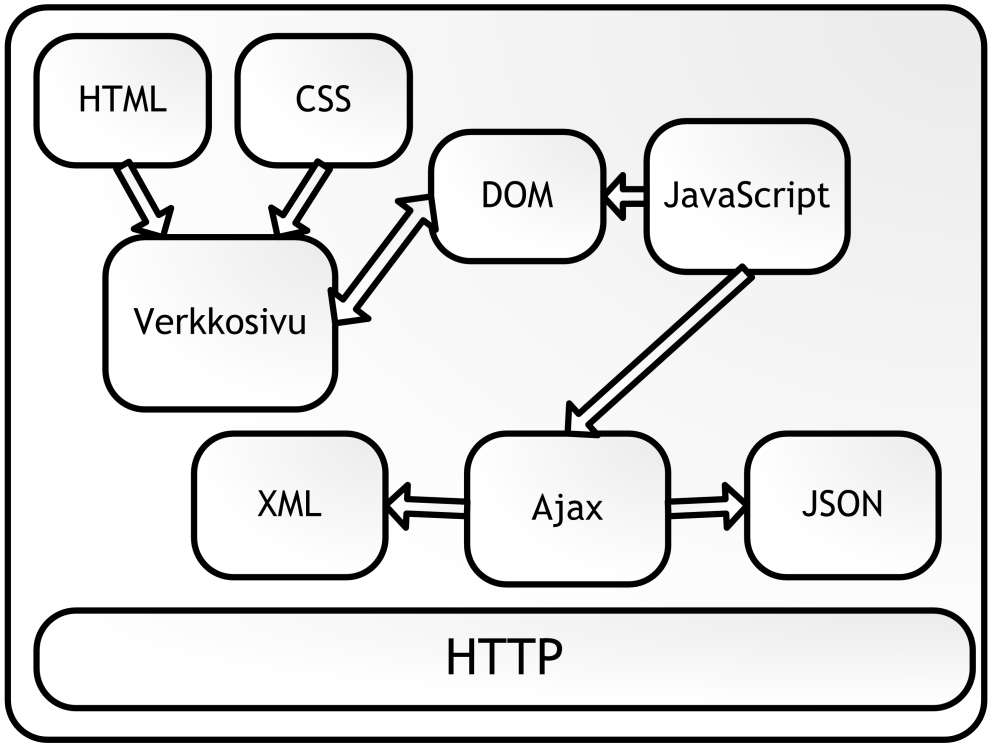
Web-ohjelmointiympäristö koostuu selaimeen rakennetuista teknologioista sekä erikseen asennettavista lisäosista. Näillä mahdollistetaan *Web 2.0* -sovellukset, jotka käsittävät verkkosovellukset, joissa keskitytään tiedon jakamiseen, sosiaaliseen verkostoitumiseen sekä käyttäjäkeskeiseen suunnitteluun. Keskeisimpiä teknologioita Web 2.0:n toteuttamiseen ovat olleet Ajax (Asynchronous Javascript and XML) [3], Adobe Integrated Runtime (AIR) [4], Google Web Toolkit (GWT) [5], Google Gears[1], JavaFX [6], Microsoft Silverlight [7] ja Ruby on Rails [8]. Näiden lisäksi on kymmeniä järjestelmiä verkkosovelluksien kehittämiseen, joista suurimmalla osalla ei ole kovin isoja mahdollisuuksia suureen menestykseen [9]. Viime vuosina web-standardit ovat kehittyneet vastaamaan tarvetta dynaamisimmille verkkopalveluille. Standardit yksinkertaistavat vanhoja vaatimuksia ja mahdollistavat teknologian sisällyttämisen selaimeen. Tämä vähentää selainkohtaisten lisäosien tarvetta.

2.1. Johdatus web-teknologioihin

Nykyinen web-ohjelmointiympäristö nojautuu vahvasti useaan teknologiaan, jotka ovat web-selaimen keskeisiä komponentteja [10]. Näitä teknologioita ovat mm. HTTP (HyperText Transfer Protocol), HTML (HyperText Markup Language), CSS (Cascading Style Sheets), JavaScript-skriptikieli, DOM (Document Object Model), JSON (JavaScript Object Notation), XML (Extensible Markup Language) sekä Ajax (Asynchronous JavaScript and XML). Nämä teknologiat yhdessä muodostavat nykyaikaisen verkkosivun ja kuvassa 2.1 esitetään kuinka teknologiat asettuvat toisiinsa nähden verkkosovelluksessa. Seuraavassa käydään edellä mainitut teknologiat läpi yksityiskohtaisella tasolla.

Asiakas-palvelin -mallin toteuttava järjestelmä on hajautettu sovellus, joka jakaa ohjelman suorituksen palvelimen ja asiakkaiden välillä. Selainsovelluksissa suoritus jakautuu asiakkaana toimivan käyttäjän selaimen ja verkkosovelluksen tarjoavan palvelimen välillä. Palvelin tarjoaa keskitetyt resurssit kuten HTML-, CSS- ja JavaScript-tiedostot, mahdollisen tietokanta-abstraktion sekä sovelluslogiikan. Selain renderöi verkkosivut ja suorittaa JavaScript-koodin. Palvelin suorittaa sovelluksesta ne osat, joista tarvitsee pitää kirjaa sekä ne, jotka käyttävät yhteisiä resursseja. Käyttäjää koskettavat, kuten käyttöliittymä, jäävät selaimelle suoritettavaksi.

[1] `http://gears.google.com/`



Kuva 2.1: Web-teknologiat verkkosovelluksessa.

HTTP eli *HyperText Transfer Protocol* on verkkoprotokolla, joka mahdollistaa hajautetun kommunikoinnin verkon välityksellä ja se muodostaa internetin pohjan [11]. Protokolla toteuttaa kysely-vastaus -periaatteen asiakas-palvelin -mallissa. Verkkosovelluksissa asiakkaana toimii selain, joka kommunikoi HTTP:n avulla sovelluksen tarjoavan palvelimen kanssa.

HTML eli *HyperText Markup Language* on standardoitu verkkosivujen merkkauskieli. Se tarjoaa keinot kuvata tekstipohjaisen dokumentin rakenteen [12]. Tämä saavutetaan siten, että tietyt tekstit merkataan mm. otsikoiksi, kappaleiksi ja listoiksi. Lisäksi HTML tarjoaa keinot laajentaa dokumenttia interaktiivisilla lomakkeilla, upotetuilla kuvilla sekä muilla objekteilla. HTML kirjoitetaan avainsanojen muodossa, jotka ilmaistaan pienempi kuin ja suurempi kuin -merkeillä. Nykyinen HTML4-versio standardoitiin vuonna 2000, seuraavan HTML5-version suunnittelu aloitettiin 2004 nimellä Web Applications 1.0 ja sen odotetaan valmistuvan 2014[2].

CSS eli *Cascading Style Sheets* on tyylisivukieli, jota käytetään kuvaamaan miltä merkkauskielellä kirjoitettu dokumentti näyttää [13]. Tyylisivu mahdollistaa tyylillisten sääntöjen, kuten värit, fontit, elementtien sijoittelun ja koon, määrittämisen erillään verkkosivun sisällöstä. Usein CSS-määritykset sisällytetään HTML-dokumenttiin esittelemään uusia ulkonäkömäärityksiä napeille, fonteille, ja muille esitystapaan liittyville yksityiskohdille.

[2] `http://www.w3.org/2011/02/htmlwg-pr.html`

JavaScript on ECMAScriptin murre, joka on web-ympäristössä dominoivassa asemassa oleva skriptikieli [14]. HTML:n ja CSS:n lisäksi verkkosivut voivat sisältää suoritettavaa koodia. Syntaksiltaan JavaScript muistuttaa C- ja Javakieliä, mutta semantiikaltaan JavaScript on paljon dynaamisempi, tulkattu kieli, joka lainaa ominaisuuksia muilta dynaamisilta kieliltä kuten Smalltalk [15], Lisp [16] ja Self [17]. Vaikka JavaScript on täysin kykeneväinen ohjelmointikieli, verkkosivut yleensä käyttävät pieniä skriptejä, joilla animoidaan tai muuten elävöitetään verkkosivuja.

DOM eli *Document Object Model* on alustariippumaton tapa ilmaista objektien kokoelma, jotka muodostavat selaimessa esitetyn sivun [18]. DOM:n avulla skriptikielet voivat ohjelmallisesti tutkia ja muuttaa sivun tilaa. Käytännössä DOM toimii rajapintana selaimen ja skriptikielen välillä, mahdollistaen joustavan kommunikaation molempien välillä.

JSON eli *JavaScript Object Notation* on tekstipohjainen standardi, joka tarjoaa JavaScript-pohjaisen tallennusmuodon tiedonvälitykseen [19]. Standardi määrittää mallin, jolla kuvataan yksinkertaisia tietorakenteita ja avain-arvo pareja. JSON:n määrittää tyypit numeroille, merkkijonoille, totuusarvoille, taulukoille, vapaavalintaisille objekteille sekä määrittelemättömän arvon.

XML eli *Extensible Markup Language* on standardoitu tapa esittää dokumentteja ohjelmallisesti käsiteltävässä muodossa [20]. XML määrittää säännöt, joiden mukaan dokumentti muodostetaan. Nämä säännöt on suunniteltu yksinkertaisuutta, yleiskäyttöisyyttä ja internet-käyttöä silmällä pitäen. Vaikka XML on suunniteltu dokumentteja ajatellen, nykyään on jo satoja sovelluksia, jotka hyödyntävät XML:ää tietorakenteiden esittämiseen.

Ajax eli *Asynchronous JavaScript And XML* on kokoelma teknologioita, joilla mahdollistetaan verkkosivun osittainen päivittäminen reaaliaikaisesti ilman käyttäjän vuorovaikutusta [21]. Ajaxissa tehdään asynkronisia kyselyitä palvelimelle, joiden vastauksien mukaan muokataan DOM:ia ja siten päivitetään verkkosivua. Asynkronisuudesta johtuen sovelluksen käyttäjä voi jatkaa sovelluksen käyttämistä, vaikka vastaus palvelimelta ei ole vielä saapunut.

Same Origin Policy on selaimen tietoturvaominaisuus, joka rajoittaa mihin osoitteisiin JavaScript-sovellus saa tehdä kyselyitä. Se määrää, että sovellus saa tehdä kyselyitä vain niihin resursseihin, jotka tulkitaan olevan samassa osoitteessa. Tämä tulkinta tehdään protokollan, portin ja verkkotunnuksen perusteella taulukon 2.1 mukaisesti. Jos sovellus tekee kyselyn osoitteeseen, jonka Same Origin Policy kieltää, selain estää kyselyn lähettämisen ja palauttaa virhekoodin.

Mashup-sovellukset ovat sovelluksia, joissa dynaamista sisältöä yhdistetään useista lähteistä. Verkkosovelluksilla ei ole työpöytäsovelluksen tyypillisiä rajoitteita, vaan ne voidaan julkaista maailmanlaajuisesti yhtä aikaa ja ne voivat vapaasti käyttää julkisesti saatavilla olevia verkkopalveluita. Kun Mashup-teknologioita sovelletaan ohjelmistotuo-

Taulukko 2.1: Same Origin Policyn säännöt

Osoite	Tulos	Syy
http://www.example.com/dir/page.html	Ok	Sama portti, protokolla ja verkkotunnus.
http://www.example.com/dir2/other.html	Ok	Sama portti, protokolla ja verkkotunnus.
http://www.example.com:81/dir/other.html	Virhe	Sama protokolla ja verkkotunnus, mutta eri portti.
https://www.example.com/dir/other.html	Virhe	Eri protokolla.
http://en.example.com/dir/other.html	Virhe	Eri verkkotunnus.
http://example.com/dir/other.html	Virhe	Eri verkkotunnus.
http://v2.www.example.com/dir/other.html	Virhe	Eri verkkotunnus.

tantoon, saadaan uusi tapa kehittämään ohjelmistoja, jota kutsutaan *Mashwareksi*. [1]

Mashwaressa ohjelmistokehittämisen mentaliteetti on ympäri maailmaa olevien resurssien dynaamisessa yhdistelyssä. Sovelluksen pääohjelma ladataan kehittäjän verkkopalvelimelta, joka lataa dynaamisesti muut vaadittavat komponentit kuten widget-kirjaston käyttöliittymän esittämiseen, osakevisualisointi-kirjaston osakegraafien luomiseen, osaketiedot pörssin tarjoamasta palvelusta sekä lokalisointikomponentit. Kaikki nämä ladataan dynaamisesti eri verkkopalvelimilta käyttäjän selaimeen sellaisenaan. Tämänkaltaiset sovellukset kehitetään käyttäen mahdollisimman paljon olemassa olevia komponentteja, jotka haetaan verkosta tarvittaessa. Sovellus ajettaisiin käyttäjän selaimessa, jolloin laskentatehovaatimukset siirretään palvelimelta käyttäjälle.

Edellä esiteltyihin web-teknologioihin liittyy lukuisia ongelmia, kuten HTML4:n monimutkaisuus ja monien yleisesti käytössä olevien tekniikoiden käyttö liitännäisien avulla. Näiden ongelmien ratkaisuun on kehitetty uusia spesifikaatioita, joista ehkä keskeisimmät ovat HTML5 ja WebGL. Tutustumme niihin seuraavassa.

2.2. HTML5

HTML5 on seuraava versio HTML-standardista, joka määrittää miten verkkosivut rakentuvat. HTML5 yksinkertaistaa HTML4:n määrittämää rakennetta, lisää uusia elementtejä ja rajapintoja sekä määrittää miten virheelliset dokumentit pitää käsitellä. Lisäksi HTML5 tuo uusia elementtejä mm. semanttiseen merkkaukseen, median käsittelyyn sekä 2D-piirtoon. [22]

Rakenne. HTML5 yksinkertaistaa dokumentin rakennetta. XHTML:n monimutkainen tyyppimäärittely, joka pohjautui verkkosivujen taaksepäin yhteensopivuuteen, on yksinkertaistettu helposti muistettavaan muotoon [22]. Ohjelmassa 2.1 on esitetty, minkälainen XHTML-standardin mukainen dokumentti yksinkertaisimmillaan on ja ohjelmassa 2.2 on HTML5:n mukainen vastaava dokumentti.

Ohjelma 2.1: XHTML:n mukainen tyyppimääritys

```
<!DOCTYPE html
    PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
```

```
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>Title of the document</title>
  </head>
  <body>
    The content of the document......
  </body>
</html>
```

Ohjelma 2.2: HTML5:n mukainen tyyppimääritys

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>Title of the document</title>
  </head>
  <body>
    The content of the document......
  </body>
</html>
```

Semanttiset elementit. HTML5 määrittää uusia semanttisia elementtejä kuten *section*, *nav*, *article*, *aside*, *hgroup*, *header*, *footer*, *time* ja *mark* [22]. Näillä elementeillä voidaan kuvata dokumentin rakennetta kattavammin, jolloin sivun rakenteen muodostaminen on helpompaa kuin nykyisillä heading-elementtien numeroinneilla, div-elementtien luokilla ja id:llä. Ohjelmassa 2.3 on kuvattu kuinka header-, content- ja footer-elementit toteutetaan käyttäen div-elementtejä ja css-luokkia. Ohjelmassa 2.4 sama on toteutettu HTML5:n header-, article- ja footer-elementeillä.

Ohjelma 2.3: XHTML:n elementeillä toteutetut header-, content- ja footer-komponentit.

```
<div id="header" class="header">
  <h1>My Weblog</h1>
  <p class="tagline">A lot of effort went into making this effortless.</p>
</div>
<div id="content" class="content">
  <div class="post">First post of really important blog.</div>
  <div class="post">Second post of really important blog.</div>
</div>
<div id="footer" class="footer">
  This blog was created by very important blogger.
</div>
```

Ohjelma 2.4: HTML5:n header-, article- ja footer-elementit.

```
<header>
  <h1>My Weblog</h1>
  <p class="tagline">A lot of effort went into making this effortless.</p>
</header>
<div id="content" class="content">
  <article>First post of really important blog.</article>
  <article>Second post of really important blog.</article>
</div>
<footer>
  This blog was created by very important blogger.
</footer>
```

Mediaelementit. HTML5 esittelee mediatoistoon audio- ja videoelementin. [22]. Näillä elementeillä mahdollistetaan median toisto suoraan selaimessa ilman lisäosia. Audioelementti toistaa äänisisältöä ja videoelementti videosisältöä, johon mahdollisesti liittyy myös äänidata. Elementit myös mahdollistavat selaimessa oletuskäyttöliittymäkomponenttien käyttämisen sekä vaihtoehtoja sille kuinka mediaa toistetaan sivun latauksen edetessä. Ohjelmassa 2.5 on esitetty kuinka audio- ja video-elementit luodaan käyttäen selaimen käyttöliittymäkomponentteja.

Ohjelma 2.5: Mediaelementit oletuskäyttöliittymällä

```
<audio controls preload >
  <source src="demo.mp3" />
</audio>

<video id="movie" width="320" height="240" preload controls>
  <source src="demo.webm" type='video/webm; codecs="vp8, vorbis"' />
</video>
```

Canvas. Canvas-elementti tarjoaa skripteillä varustetun resoluutioriippuvaisen piirtoalustan, jota voidaan käyttää renderöimään graafeja, peligrafiikkaa tai muita visuaalisia elementtejä [22]. Ohjelmassa 2.6 näytetään kuinka canvas-elementti määritetään. Esimerkin canvas-elementille annetaan nimi, leveys ja korkeus.

Ohjelma 2.6: Canvas-elementin määritys

```
<canvas width="300" height="300" id="example"></canvas>
```

Paikallinen tietosäilö. Web Storage on mekanismi jolla voidaan tallentaa avain/arvopareja selaimeen ja myöhemmin noutaa näitä [22]. Mekanismin tavoitteena on tarjota keinot, joilla rakentaa interaktiivisia sovelluksia ominaisuuksilla kuten mahdollisuudella työskennellä ilman verkkoyhteyttä. Web Storage yrittää paikata evästeiden (engl. cookie) puutteita tarjoamalla isomman, turvallisemman ja helpomman tavan tallentaa tietoa. Web Storage -tietosäilöön tallennetut tiedot ovat jaettavissa muiden sovelluksien kesken, kunhan sovelluksia ajetaan samassa selaimessa. Ohjelmassa 2.7 on esitetty kuinka paikalliseen tietosäilöön voidaan tallentaa ja noutaa aikaisemmin tallennettu tieto.

Ohjelma 2.7: Web storagen käyttö

```
var testObject = "teststring";
localStorage.setItem("test", testObject);

var retrievedObject = localStorage.getItem("test");
```

Dokumentin muokkaaminen. ContentEditable-attribuutit ovat HTML5:n tarjoamia rajapintoja, joilla selaimeen avattua dokumenttia voidaan muokata siten, että muutokset tallentuvat selaimessa olevaan dokumenttiin [22]. Näillä rajapinnoilla voidaan tekstidokumenttia muokata kuin selain olisi rikas tekstieditori. Rajapinnat mahdollistavat käskyjen, kuten tekstin kursivointi, lihavointi, koon muuttaminen ja värien manipulointi, lähettämisen selaimelle, joka muokkaa avoinna olevaa dokumenttia ilman että käyttäjän tarvitsee HTML:ää ymmärtää. Rajanpintoja käytetään WYSIWYG-editoreissa, joiden tuotok-

set voidaan tallentaa palvelimelle käyttäen normaaleja lomakkeiden lähetystoimintoja.

Elementtien siirtäminen hiirellä. Vedä ja pudota -menetelmä (engl. Drag and Drop) on ominaisuus, jossa voi elementtejä poimia hiirellä ja siirtää toiseen paikkaan [22]. Selaimen rajapinta antaa tiedon minkälaista dataa siirretään, mitä dataa siirretään sekä tapahtumista kuten siirron aloituksesta ja lopetuksesta. Siirrettävä data voi olla tekstiä, linkkejä, kuvia, tiedostoja, HTML:ää ja XML:ää tai DOM-puun solmuja. Ohjelmassa 2.8 on kuvattu kuinka elementti määritetään vastaanottamaan hiirellä pudotettuja tiedostoja. Verkkosovellukselle tiedosto tuodaan tiedosto-objektina, josta tiedostonkäsittelijä tarkistaa tiedostontyypin ja jos kyseessä on kuva, se lisätään verkkosivulle.

Ohjelma 2.8: Vedä ja pudota -menetelmän käyttö HTML-elementissä.

```
<input type="file" id="input" onchange="handleFiles(this.files)">
<script type="text/javascript">
function handleFiles(files) {
  for (var i = 0; i < files.length; i++) {
    var file = files[i];
    var imageType = /image.*/;
    if (!file.type.match(imageType)) {
      continue;
    }
    var img = document.createElement("img");
    img.classList.add("obj");
    img.file = file;
    preview.appendChild(img);
    var reader = new FileReader();
    reader.onload = (function(aImg) {
        return function(e) { aImg.src = e.target.result; };
      })(img);
    reader.readAsDataURL(file);
  }
}
</script>
```

Dokumenttien välinen viestintä. HTML5:ssä dokumentit voivat viestiä keskenään, mikä mahdollistaa Same Origin Policyn kiertämisen turvallisesti [22]. Viestissä lähetetään itse viesti sekä tieto siitä kuka viestin lähetti. Vastaanottaessa voidaan lähettäjän perusteella päättää käsitelläänkö viesti vai ei. Ohjelmassa 2.9 on esitetty kuinka verkkosovellus voi lähettää toiselle sovellukselle viestin ja kuinka toinen sovellus voi tämän ottaa vastaan.

Ohjelma 2.9: Dokumenttien välinen viestintä.

```
otherWindow.postMessage(message, targetOrigin);

//message receiver in other window
window.addEventListener("message", receiveMessage, false);
function receiveMessage(event)
{
  if (event.origin !== "http://example.org:8080")
    return;

}
```

Selaimen historian hallitseminen. Historiarajapinta (engl. History API) mahdollistaa selaimen historian käsittelyn [22]. Rajapinnalla selaimen voi käskyttää siirtymään sivuhistoriassa sekä eteen- että taaksepäin. Lisäksi voidaan tallentaa tilatietoa sivuun liittyen, joka ladataan kun kyseiselle sivulle selaimen historiatiedoissa mennään. Ohjelmassa 2.10 esitetään kuinka sovellus voi siirtyä eteen- ja taaksepäin sivuhistoriassa.

Ohjelma 2.10: Sivuhistoriassa liikkuminen.

```
window.history.back();
window.history.forward();
window.history.go(-5);
```

Uusien MIME-tyyppien ja protokollien rekisteröinti. HTML5 mahdollistaa MIME-tyyppien ja protokollien rekisteröinnin ohjelmallisesti [22]. Tämä tarjoaa kehittäjälle mahdollisuuden käyttää epästandardeja sovelluskohtaisia MIME-tyyppejä ja protokollia. Rekisteröidessä selaimelle kerrotaan protokolla, verkko-osoite johon protokolla kohdistuu sekä käyttäjäystävällinen nimi protokollan käsittelijälle. Käsittelijä kutsutaan automaattisesti, kun käyttäjä aktivoi uuden protokollan sisältävän verkko-osoitteen. Palvelimella voidaan sen jälkeen jäsentää protokolla sivupyynnön perusteella.

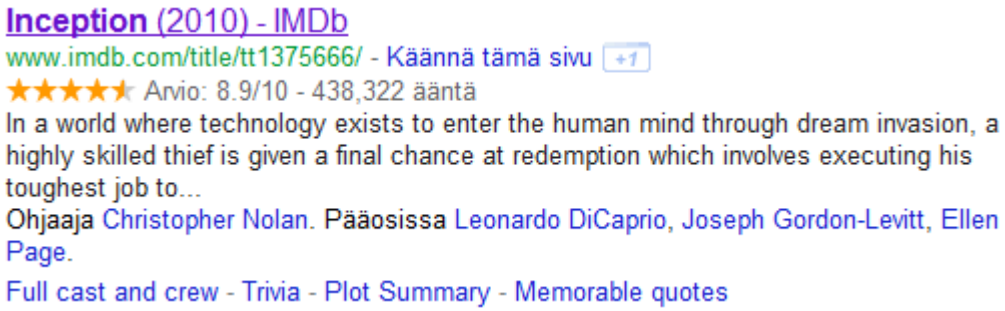
Microdata. Microdata on spesifikaatio, jolla voidaan merkitä semanttisesti minkä tyyppistä tieto on [22]. Hakukoneet, verkon selaajat ja selaimet jäsentävät näitä tietoja verkkosivuilta ja käyttävät niitä tarjoamaan rikkaamman kokemuksen käyttäjälle. Ohjelmassa 2.11 on kuvattu kuinka elokuvan tietoja merkataan semanttisesti ja kuvassa 2.2 on esitetty kuinka Googlen hakutulokset nämä merkkaukset esittävät.

Ohjelma 2.11: Microdatan käyttö semanttiseen merkkaukseen.

```
<div itemprop="aggregateRating" itemscope itemtype="http://schema.org/AggregateRating">
  <span itemprop="ratingValue">8.9</span>
  <span itemprop="bestRating">10</span>
  <span itemprop="ratingCount">438,322</span>
</div>

<div itemscope itemtype="http://schema.org/Movie">
  <p itemprop="description">
    In a world where technology exists to enter the human mind through dream invasion, a
        highly skilled thief is given a final chance at redemption which involves
       executing his toughest job to date: Inception.
  </p>
  <a  href="/name/nm0634240/" itemprop="director">Christopher Nolan</a>
  <a itemprop="actors">Leonardo DiCaprio</a>
  <a itemprop="actors">Joseph Gordon-Levitt</a>
  <a itemprop="actors">Ellen Page</a>
</div>
```

2.3. HTML5:n ulkopuoliset teknologiat

HTML5:n ohessa tulee spesifikaatioita, jotka eivät ole osa varsinaista HTML5:n spesifikaatiota, mutta jotka usein liitetään HTML5:een. Näihin kuuluu mm. verkkotyöläiset,



Kuva 2.2: Googlen hakutuloksissa esiintyvät microdatalla merkityt tiedot.

käyttäjän paikannus, indeksoitu tietokanta, verkkokanavat, tiedostojärjestelmän sekä tiedoston käsittelyrajapinnat.

Verkkotyöläiset. Verkkotyöläiset (engl. Web Worker) määrittävät rajapinnan, joka mahdollistaa skriptin ajamisen taustalla itsenäisesti käyttöliittymäskripteistä [23]. Tämä mahdollistaa pitkäkestoiset skriptit, joita ei keskeytetä painalluksilla ja muilla käyttäjäinteraktioilla. Työläiset ovat yleensä raskaampaan laskentaan suunniteltuja eikä niitä ole tarkoitus käyttää runsaasti, koska järjestelmän resurssien kulutus on huomattavaa. Työläiset sallivat rinnakkaisen koodin ajamisen useassa säikeessä, joka mahdollistaa selaimen normaalin toiminnan koodin ajon aikana. Työläiset voivat olla joko omistautuneita tai jaettuja. Omistautuneet työläiset luodaan tietyn sovelluksen käyttöön, kun taas jaetut työläiset tarjoavat palvelujaan useille muille sovelluksille. Kommunikaatio työläisen ja sovelluksen välillä toteutetaan viestien jäsennyksellä. Ohjelmassa 2.12 kuvataan kuinka omistautunut työläinen luodaan laskemaan alkulukuja ja löydetyt alkuluvut välitetään dokumentille ilman että selaimen suoritusta estetään.

Ohjelma 2.12: Omistautunut työläinen.

```
var worker = new Worker('worker.js');
//worker receives message and updates the webpage
worker.onmessage = function (event) {
  document.getElementById('result').textContent = event.data;
};

//worker calculates primes in separate process and posts them to browser
var n = 1;
search: while (true) {
  n += 1;
  for (var i = 2; i <= Math.sqrt(n); i += 1)
  if (n % i == 0)
    continue search;
  postMessage(n);
}
```

Paikannus. Paikannusrajapinta (engl. Geolocation) mahdollistaa käyttäjän antamaan sijaintinsa verkkosovelluksille [24]. Yksityisyyden suojan vuoksi käyttäjältä kysytään aina lupa ennen kuin selain välittää sijainnin sovellukselle. Käyttäjä paikannetaan joko IP-osoitteen, MAC-osoitteiden, WLAN-yhteyden sijainnin, GPS:n tai puhelinverkon tukia-

semien avulla. Sovellus saa paikannuksen yhteydessä tietoonsa leveys- ja pituuspiirien koordinaatit, joita se voi käyttää haluamallaan tavalla. Ohjelmassa 2.13 on esitetty kuinka rajapintaa käytetään pyytämään käyttäjältä sijaintia ja luvan saadessaan sijainti piirretään kartalle, joka sijoitetaan verkkosivulle.

Ohjelma 2.13: Paikantaminen ja sijaintitiedon piirtäminen kartalle.

```
function get_location() {
  navigator.geolocation.getCurrentPosition(show_map);
}

function show_map(loc) {
  $("#geo-wrapper").css({'width':'320px','height':'350px'});
  var map = new GMap2(document.getElementById("geo-wrapper"));
  var center = new GLatLng(loc.coords.latitude, loc.coords.longitude);
  map.setCenter(center, 14);
}
```

Indeksoitu tietokanta. Selaimiin on toteutettu tietokanta, joka tarjoaa keinon tallentaa merkittäviä määriä rakenteellista dataa, jota voi hakea tehokkailla kyselyillä [25]. Tietokantaan tallennetaan avain/arvo -pareja, jotka on rakennettu transaktiomallin päälle. Tietokantaan voi tehdä sekä synkronisia että asynkronisia kyselyitä. Asynkroniset kutsut on toteutettu takaisinkutsu-menetelmällä ja synkroniset pohjautuvat verkkotyöläisiin. Ohjelmassa 2.14 näytetään kuinka tietokanta määritetään sovellukseen ja sieltä noudetaan customers-säilöstä avaimella dataa.

Ohjelma 2.14: Indeksoidun tietokannan käyttö.

```
var db;
var request = mozIndexedDB.open("MyTestDatabase");
request.onerror = function(event) {
  alert("Why didn't you allow my web app to use IndexedDB?!");
};
request.onsuccess = function(event) {
  db = request.result;
};

var transaction = db.transaction(["customers"]);
var objectStore = transaction.objectStore("customers");
var request = objectStore.get("444-44-4444");
request.onerror = function(event) {
  // Handle errors!
};
request.onsuccess = function(event) {
  // Do something with the request.result!
  alert("Name for SSN 444-44-4444 is " + request.result.name);
};
```

Verkkokanavat. Verkkokanavat (engl. WebSockets) on teknologia, joka mahdollistaa kaksisuuntaisen kommunikaatioväylän selaimen ja palvelimen välillä [26]. Kanavan kautta verkkosovellukset voivat kommunikoida palvelimen kanssa reaaliaikaisesti sen sijaan, että jatkuvasti kyseltäisiin onko jokin muuttunut. Ohjelmassa 2.15 on kuvattu kuinka kanava luodaan ja sen kautta viestitään molempiin suuntiin palvelimen ja selaimen välillä.

Ohjelma 2.15: Verkkokanavan käyttö.

```
//create websocket connection to example.com using custom protocol
var mySocket = new WebSocket("http://www.example.com/socketserver",
  "my-custom-protocol");

//send message to server
mySocket.send("Here's some text that the server is urgently awaiting!");

//receive message from server
mySocket.onmessage = function(e) {
  console.log(e.data);
}
```

Tiedostojärjestelmä. Tiedostojärjestelmän rajapinta (engl. Filesystem API) tarjoaa mahdollisuuden lukea ja kirjoittaa tiedostoja ja hakemistoja hiekkalaatikkoon rajoitetussa ympäristössä [27]. Tiedostojärjestelmästä voidaan tehdä väliaikainen tai persistoitu. Väliaikaiset tiedostojärjestelmät voidaan poistaa selaimen niin halutessa. Ohjelmassa 2.16 on esitetty kuinka hiekkalaatikkoon tallennettu tiedosto voidaan lukea ja sen sisältö lisätä verkkosivulle.

Ohjelma 2.16: Tiedostojärjestelmän käyttö.

```
function onInitFs(fs) {

  fs.root.getFile('log.txt', {}, function(fileEntry) {

  // Get a File object representing the file,
  // then use FileReader to read its contents.
    fileEntry.file(function(file) {
       var reader = new FileReader();

       reader.onloadend = function(e) {
         var txtArea = document.createElement('textarea');
         txtArea.value = this.result;
         document.body.appendChild(txtArea);
       };
       reader.readAsText(file);
    }, errorHandler);
  }, errorHandler);
}

window.requestFileSystem(window.PERSISTENT, 1024*1024, onInitFs, errorHandler);
```

Tiedosto. Tiedoston rajapinta (engl. File API) mahdollistaa tiedostojen käsittelyn verkkosovelluksessa [28]. Rajapinnan kautta voidaan selvittää tiedoston nimi, koko ja tyyppi. Tiedostonlukijalla voidaan lukea tiedostorajapinnan käyttämä tiedosto sovelluksen käytettäväksi. Lisäksi tiedostonkirjoittajalla voidaan kirjoittaa tiedostoja suojattuun ympäristöön. Tiedoston lukua on esitelty ohjelmissa 2.8 ja 2.16

2.4. WebGL

WebGL [29] on alustariippumaton standardi laitteistokiihdytetyn 3D-grafiikan esittämiseen selaimessa. Standardin tarkoitus on mahdollistaa tämä ilman lisäosien asennusta,

ja vaikka se on suunniteltu 3D-grafiikkaa silmällä pitäen, sitä voidaan käyttää myös 2D-grafiikassa. WebGL perustuu OpenGL ES 2.0:aan [30], ja se käyttää OpenGL:n GLSL-sävytinkieltä [31]. WebGL ajetaan HTML5:n canvas-elementissä, ja sen tietosisältö on käytettävissä DOM-rajanpinnan kautta. WebGL tarjoaa JavaScript-rajapinnan, jonka kautta OpenGL -standardin mukaista koodia voidaan tuottaa.

2.4.1. Johdatus 3D-grafiikkaan

Kolmiulotteisen grafiikan esittäminen tietokoneen näytöllä jakautuu kolmeen osaan: sovellus-, geometria- ja rasterointivaiheisiin [32]. Sovellusvaiheessa tehtävänä on luoda kolmiulotteisesta mallista piirtämiseen soveltuva versio. Tuloksena on tyypillisesti pisteistä, viivoista ja monikulmioista sekä niihin liittyvästä oheistiedosta koostuva malli, joka voidaan lähettää grafiikkapiirtojärjestelmälle piirrettäväksi. Sovellusvaiheen tehtävät hoidetaan tietokoneen prosessorilla. Näihin tehtäviin kuuluvat mm. kappaleiden määrittely, materiaalien, tekstuurien sekä valonlähteiden määritys, siirto-, skaalaus- ja kiertomuunnosten määrittely sekä animaation ja fysiikan laskenta.

Geometriavaiheessa piirrettävälle pisteistä, viivoista ja monikulmioista koostuvalle mallille tehdään geometrisia muunnoksia ja leikkauksia, joiden lopputuloksena malli on valmis piirrettäväksi kuvan muodostavaan pikseliruudukkoon. Geometriavaiheen tehtäviä ovat mm. kameramuunnos, valaistuksen laskenta, perspektiivimuunnos, leikkaus sekä muunnos ikkunakoordinaatistoon.

Rasterointivaiheessa tuotetaan lopullinen kaksiulotteinen pikseleistä koostuva rasterikuva. Geometriavaiheen jälkeen piirrettävät primitiivit on jo kuvattu kuvaruudun koordinaatistoon, kuitenkin kolmiulotteisina niin, että pisteillä on myös syvyys. Vaiheen tehtäviin kuuluvat mm. pikselin värin ja syvyyden määritys, jossa pikseli teksturoidaan ja sumutetaan riippuen etäisyydestä katsojaan. Lisäksi pikselille tehdään siluettipuskuri-, alpha- ja syvyystestit, jotka määrittävät piirretäänkö kyseinen pikseli videopuskuriin. Jos uusi pikseli päätetään piirtää, sillä korvataan vanha pikseli kuvasta, tai sen väri yhdistetään vanhaan väriin sekoittamalla.

3D-grafiikan muunnokset perustuvat matriisilaskentaan. Sovellusvaiheessa määritettävät muunnokset toteutetaan 4x4-matriiseilla, jotka kerrotaan yhteen. Geometriavaiheessa kamera- ja perspektiivimuunnokset toteutetaan myös 4x4-matriiseilla, joiden lopputulos sovelletaan malliin yhdessä sovellusvaiheelta saadun muunnosmatriisin kanssa ennen leikkausta ja ikkunakoordinaatistoon siirtämistä.

Piirtorajapinta, kuten OpenGL[3] tai DirectX[4], abstrahoi geometria- ja rasterointivaiheen rajapinnan taakse. OpenGL:ssä asetetaan tarvittavat piirto- ja syvyyspuskurit sekä niihin liittyvät testit. Lisäksi asetetaan geometriset muunnosmatriisit ja sävytysmallit sekä valaistuksen, pintamateriaalien ja tekstuurien asetukset. Lopuksi primitiivit piirretään.[32]

[3] http://www.opengl.org/
[4] http://msdn.microsoft.com/directx/

2.4.2. 3D-grafiikka selaimessa

WebGL:n rajapinta on toteutettu alhaisemmalla tasolla kuin vastaavat OpenGL:n rajapinnat, joten ohjelmistokehittäjä joutuu toteuttamaan yleisesti käytettyjä OpenGL:n toimintoja itse. Tämä johtaa ylimääräiseen vaivaan toteuttaessa 3D-sovellusta, joten useita WebGL-kirjastoja on kehitetty helpottamaan ja nopeuttamaan ohjelmistokehittäjän työtä. Näitä kirjastoja käsitellään tarkemmin jatkossa.

Yksinkertaisimmillaan WebGL:n käyttäminen vaatii vain canvas-elementin, verteksi- ja fragmenttisävyttimet ja sovelluslogiikan, joka määrittää mitä renderöidään. Canvas-elementin määritys tehdään, kuten ohjelmassa 2.6 on esitetty. Sävyttimiä käytetään sekä muokkaamaan geometriaa että yksittäisten pikselien värin määrittämiseen. [32]

Verteksisävytin korvaa kiinteät verteksimuunnokset ja verteksikohtaisen valaistuksen laskennan. Tulokset interpoloidaan yksittäisille pikseleille ja nämä toimivat syötteinä fragmenttisävyttimelle, joka määrittää pikselin värin, näin korvaten kiinteät teksturointi- ja sumulaskennan. Sävyttimet määritetään WebGL:ssä GLSL-sävytinkielellä ja näistä on esimerkki ohjelmassa 2.17. Esimerkki pohjautuu Learning WebGL -blogin[5] esimerkkeihin.

Ohjelma 2.17: Yksinkertaiset verteksi- ja fragmenttisävyttimet

```
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec4 aVertexColor;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;

    varying vec4 vColor;

    void main(void) {
        gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
        vColor = aVertexColor;
    }
</script>

<script id="shader-fs" type="x-shader/x-fragment">
    #ifdef GL_ES
    precision highp float;
    #endif

    varying vec4 vColor;

    void main(void) {
        gl_FragColor = vColor;
    }
</script>
```

Verteksisävyttimessä on määritetty GLSL-kielellä syötteet eli *attribuutit*, globaalit muuttujat eli *uniformit* ja omat muuttujat, jotka ovat *varying*-tyyppiä. Ulostulo kirjoite-

[5] http://learningwebgl.com/blog/?page_id=1217

taan erikoismuuttujiin kuten *gl_Position*. Fragmenttisävyttimessä #ifdef-lohkossa määritetään, että käytetään liukulukuja ja pikselien väritykseen käytetään verteksisävyttimeltä saatuja interpoloituja väriarvoja.

Sovelluslogiikassa alustetaan canvas-elementin WebGL-konteksti sekä aiemmin määritellyt sävyttimet. Lisäksi logiikassa määritetään piirrettävät objektit ja funktio, joka piirtää objektit canvas-elementille. Ohjelmassa 2.18 on esitetty canvas-elementin alustaminen.

Ohjelma 2.18: Canvas-elementin alustaminen WebGL-kontekstilla

```
 try {
    canvas = document.getElementById('example');
  gl = canvas.getContext("experimental-webgl");
    gl.viewportWidth = canvas.width;
    gl.viewportHeight = canvas.height;
} catch (e) {
}
if (!gl) {
    alert("Could not initialise WebGL, sorry :-(");
}
```

WebGL-konteksti alustetaan globaaliin *gl*-muuttujaan, jonka jälkeen WebGL spesifikaation määrittämiä funktioita voidaan kutsua syntaksilla *gl.funktionNimi(parametrit)*. Alustuksen jälkeen konteksti asetetaan canvas-elementin kokoiseksi ja tarkistetaan onnistuiko alustus vai ei.

Sävyttimien alustukseen käytetään kahta funktiota, jotka on esitetty ohjelmissa 2.19 ja 2.20. Funktio *getShader* noutaa määritetyt sävytinelementit ja luo niistä WebGL:n funktioilla sävyttimet käytettäväksi.

Ohjelma 2.19: getShader sävyttimien luomiseen

```
function getShader(gl, id) {
    var shaderScript = document.getElementById(id);
    if (!shaderScript) {
        return null;
    }

    var str = "";
    var k = shaderScript.firstChild;
    while (k) {
        if (k.nodeType == 3) {
            str += k.textContent;
        }
        k = k.nextSibling;
    }

    var shader;
    if (shaderScript.type == "x-shader/x-fragment") {
        shader = gl.createShader(gl.FRAGMENT_SHADER);
    } else if (shaderScript.type == "x-shader/x-vertex") {
        shader = gl.createShader(gl.VERTEX_SHADER);
    } else {
        return null;
    }
```

```
    gl.shaderSource(shader, str);
    gl.compileShader(shader);

    if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
        alert(gl.getShaderInfoLog(shader));
        return null;
    }

    return shader;
}
```

Funktio *initShaders* käyttää *getShader*-funktiota, jonka noutamat sävyttimet tallennetaan *shaderProgram*-muuttujaan, joka on alustettu sävytinohjelmaksi (*WebGLProgram*). Jokainen sävytinohjelma voi säilöä kaksi sävytintä, verteksi- ja fragmenttisävyttimen, jotka ajetaan näytönohjaimella.

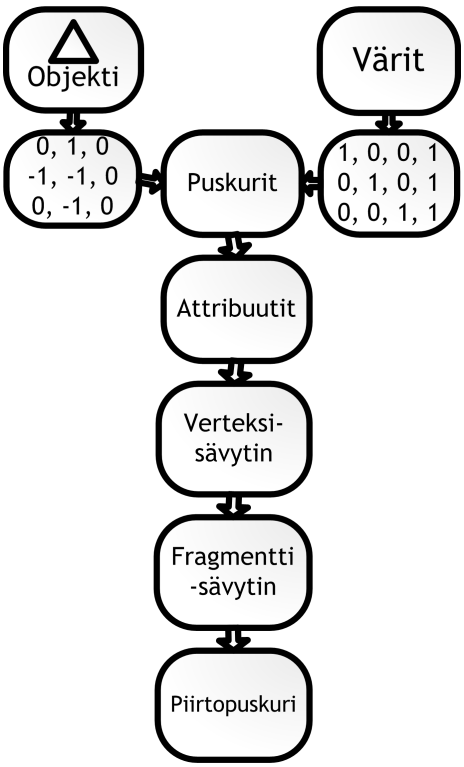
Ohjelma 2.20: initShaders sävytinohjelman luomiseen

```
function initShaders() {
  var fragmentShader = getShader(gl, "shader-fs");
  var vertexShader = getShader(gl, "shader-vs");

  shaderProgram = gl.createProgram();
  gl.attachShader(shaderProgram, vertexShader);
  gl.attachShader(shaderProgram, fragmentShader);
  gl.linkProgram(shaderProgram);

  if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS)) {
    alert("Could not initialise shaders");
  }

  gl.useProgram(shaderProgram);

  shaderProgram.vertexPositionAttribute =
    gl.getAttribLocation(shaderProgram, "aVertexPosition");
  gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);

  shaderProgram.vertexColorAttribute =
    gl.getAttribLocation(shaderProgram, "aVertexColor");
  gl.enableVertexAttribArray(shaderProgram.vertexColorAttribute);

  shaderProgram.pMatrixUniform = gl.getUniformLocation(shaderProgram, "uPMatrix");
  shaderProgram.mvMatrixUniform = gl.getUniformLocation(shaderProgram, "uMVMatrix");
}
```

Sävytinohjelman luomisen jälkeen se linkitetään ja siihen asetetaan syötteet *gl.getAttribLocation*-funktiolla. Lisäksi ohjelmalle määritellään, että halutaan käyttää kyseistä taulukkoa syötteen arvoiksi. Lopuksi tallennetaan verteksisävyttimessä luodut malli- ja projektiomatriisit sävytinohjelmaan.

Kontekstin ja sävyttimien luonnin jälkeen jäljellä on objektien luominen ja piirtäminen. Tämä piirtoprosessi on esitetty kuvassa 2.3. Prosessissa objektien määrittäminen on kehittäjän vastuulla, jotka rajapinnan kautta tallennetaan puskureihin, jonka jälkeen WebGL vastaa piirtämisestä. Esimerkin objektit ja värit määritetään omassa funktiossaan,

joka on esitetty ohjelmassa 2.21. Objektien piirtäminen on esitetty ohjelmassa 2.22, mikä muodostaa loppuosan prosessista puskureista eteenpäin.



Kuva 2.3: WebGL:n piirtoprosessi.

Ohjelma 2.21: Kolmion luominen piirrettäväksi

```
function initBuffers() {
    triangleVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
    var vertices = [
        0.0,  1.0,  0.0,
        -1.0, -1.0,  0.0,
        1.0, -1.0,  0.0
    ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.STATIC_DRAW);
    triangleVertexPositionBuffer.itemSize = 3;
    triangleVertexPositionBuffer.numItems = 3;

    triangleVertexColorBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexColorBuffer);
    var colors = [
        1.0, 0.0, 0.0, 1.0,
        0.0, 1.0, 0.0, 1.0,
        0.0, 0.0, 1.0, 1.0
    ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colors), gl.STATIC_DRAW);
    triangleVertexColorBuffer.itemSize = 4;
    triangleVertexColorBuffer.numItems = 3;
}
```

Funktio luo puskurin objektia varten, sitoo sen WebGL-kontekstiin ja lisää tarvittavat

verteksit, jotka muodostavat objektin. Verteksit annetaan koordinaatteina kolmiulotteisessa avaruudessa, jotka määrittävät objektin kulmapisteet. Lisäksi annetaan kaksi parametria, jotka määrittävät kuinka monta arvoa muodostaa yhden koordinaatin ja kuinka monta koordinaattia puskurissa on.

Ohjelma 2.22: WebGL-näkymän piirto

```
function drawScene() {
  gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

  mat4.perspective(45, gl.viewportWidth / gl.viewportHeight, 0.1, 100.0, pMatrix);
  mat4.identity(mvMatrix);

  mat4.translate(mvMatrix, [0.0, 0.0, -7.0]);
  gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
  gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
    triangleVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

  gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexColorBuffer);
  gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,
    triangleVertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);

  setMatrixUniforms();
  gl.drawArrays(gl.TRIANGLES, 0, triangleVertexPositionBuffer.numItems);
}
```

Piirtofunktiossa tyhjennetään konteksti edellisen piirron jäljiltä, jonka jälkeen määritellään perspektiivi, josta näkymää katsotaan. Tämän jälkeen alustetaan malli- ja identiteettimatriisit, joilla määritetään aloituspisteet. Objektin siirto toteutetaan myös matriisilaskulla, joka tässä esimerkissä siirtää objektia 5 askelta näkymän suuntaan. Oikean paikan löydyttyä objekti piirretään sitomalla objektin puskuri WebGL-kontekstiin, asettamalla verteksisävyttimeen verteksi- ja väripuskurit ja lopulta kutsutaan operaatiota *gl.drawArrays*, joka piirtää objektin canvas-elementtiin. Esimerkin matriisioperaatit on toteutettu erillisellä matriisikirjastolla. Lopputulos WebGL-esimerkistä on kuvassa 2.4. Esimerkin täydellinen listaus on annettu liitteessä A.

Kuva 2.4: WebGL esimerkki.

2.5. Tuki nykyselaimissa

Tuki HTML5:n tuomille ominaisuuksille tämän päivän selaimissa on vaihtelevaa. Suuri osa HTML5:n yleistasoisista ominaisuuksista on kattavasti tuettu suosituimmissa työpöytäselaimissa. Kaikkein kattavin tuki on Google Chrome 15 -selaimessa, jonka jälkeen tulevat Firefox 7, Apple Safari 5.1, Opera 11.50 ja viimeisenä Microsoft Internet Explorer 9. Selainten HTML5-tuki parantuu koko ajan, ja Internet Explorerin seuraava versio parantaa tukea huimasti. Tablet-tietokoneiden ja muiden mobiililaitteiden selainten HTML5-tuki seuraa työpöytäselainten tukea hieman jäljessä, ja niidenkin tuki kehittyy työpöytäselainten myötä. [33]

Sovelluksen kehittäjä voi toteuttaa tarkistuksia ominaisuuksien tuelle selaimessa neljällä eri tekniikalla [34]. Kehittäjä voi tarkistaa, onko tietty jäsenmuuttuja olemassa globaalissa JavaScript-objektissa, kuten *window* tai *navigator*. Ohjelmassa 2.23 tarkistetaan onko navigator-objektilla geolocation-jäsenmuuttuja.

Ohjelma 2.23: Paikannuksen tuen tarkistaminen

```
function supports_geolocation() {
  return !!navigator.geolocation;
}
```

Kehittäjä voi luoda elementin ja tarkistaa, onko tietty jäsenmuuttuja olemassa luodussa elementissä. Ohjelmassa 2.24 tarkistetaan onko canvas-elementillä *getContext*-jäsenfunktio.

Ohjelma 2.24: Canvas-elementin tuen tarkistaminen

```
function supports_canvas() {
  return !!document.createElement('canvas').getContext;
}
```

Kehittäjä voi luoda elementin, tarkistaa onko tietty funktio olemassa luodussa elementissä, kutsua kyseistä funktiota ja tarkistaa funktion palauttaman paluuarvon. Ohjelmassa 2.25 tarkistetaan, voiko tuettu video-elementti toistaa tietyn tyyppisen videon.

Ohjelma 2.25: Video-elementin toistokyvyn tarkistaminen

```
function supports_h264_baseline_video() {
  if (!supports_video()) { return false; }
  var v = document.createElement("video");
  return v.canPlayType('video/mp4; codecs="avc1.42E01E, mp4a.40.2"');
}
```

Kehittäjä voi luoda elementin, asettaa jäsenmuuttujan tiettyyn arvoon ja tarkistaa säilyttääkö jäsenmuuttuja kyseisen arvon. Ohjelmassa 2.26 tarkistetaan säilyttääkö input-elementti asetetun color-tyypin.

Ohjelma 2.26: Input-elementin color-tyypin tuen tarkistaminen

```
function supports_color_type(){
  var i = document.createElement("input");
  i.setAttribute("type", "color");
  return i.type !== "text";
}
```

Jokaisen ominaisuuden erillistä tarkistamista helpottamaan on kehitetty **Modernizr**-kirjasto[6], joka abstrahoi yksittäiset tarkistukset rajapinnan taakse. Ohjelman 2.26 tarkistus yksinkertaistuu ohjelmassa 2.27 esitettyyn muotoon.

Ohjelma 2.27: Input-elementin color-tyypin tuen tarkistaminen Modernizr-kirjastolla

```
if (!Modernizr.inputtypes.color) {
  // no support for element <input type="color"> :(
}
```

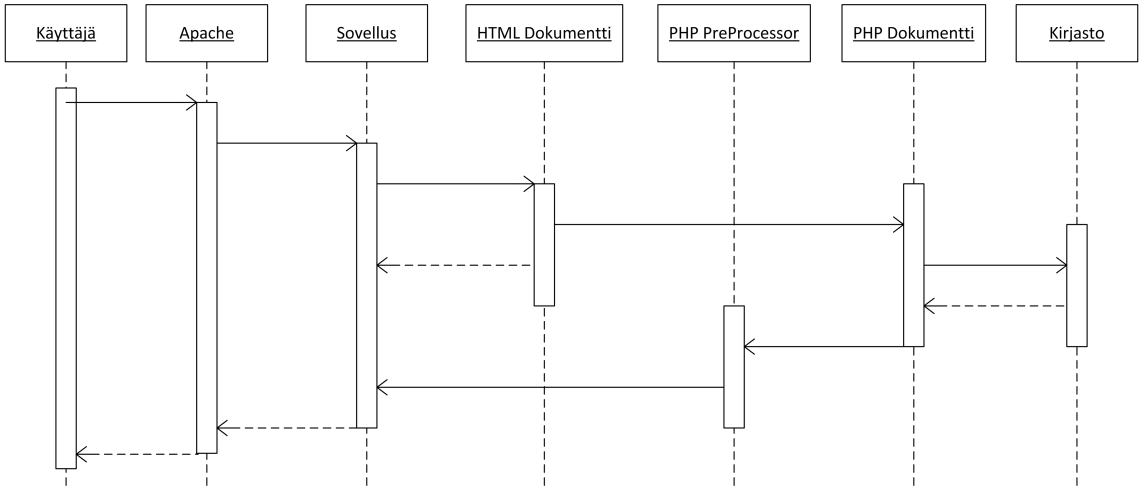
[6] http://www.modernizr.com/

3. KIRJASTOT

Web-sovellusten kehitys pohjautuu pitkälti olemassa oleviin kirjastoihin, joilla kirjaston käyttäjältä abstrahoidaan toteutusteknisiä yksityiskohtia piiloon. Kirjastot voidaan jakaa asiakas- ja palvelinpään kirjastoihin. Asiakaspäässä suurin osa kirjastoista on JavaScript-pohjaisia, palvelinpäässä valinnanvaraa on enemmän.

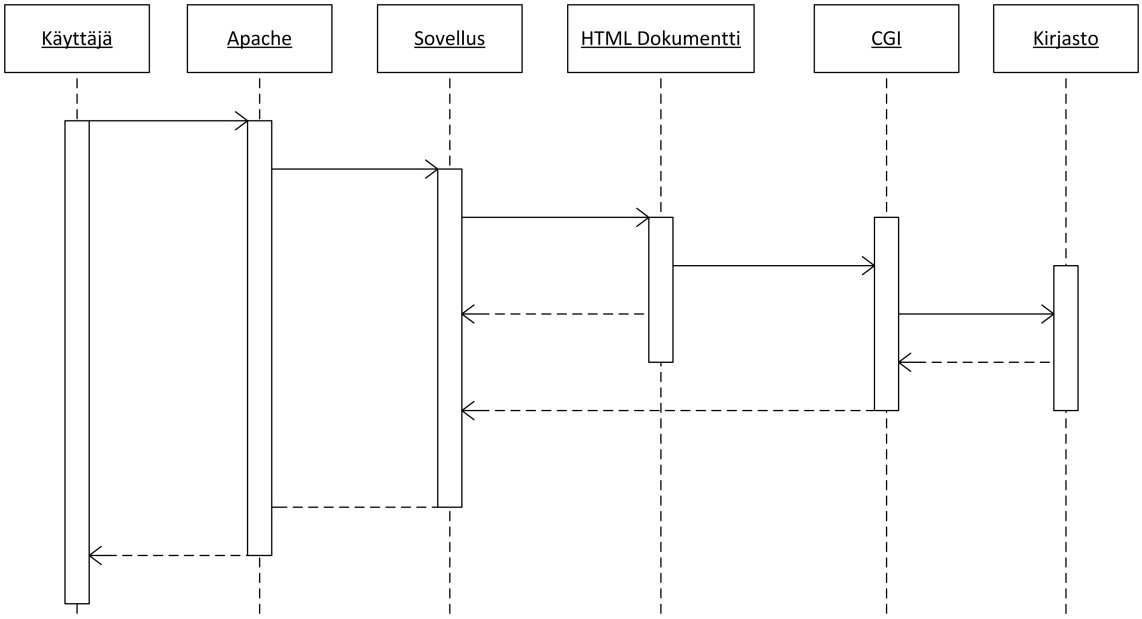
3.1. Palvelinpään kirjastot

Palvelinpään kirjastot voidaan käytännössä toteuttaa millä tahansa kielellä, kunhan sen saa vastaamaan verkkokyselyihin joko itsenäisesti tai jonkun muun komponentin välityksellä. Selainsovellukset ajetaan jollakin verkkopalvelimella, esimerkiksi Apache:lla[1] tai Microsoft IIS:llä[2]. Nämä palvelimet mahdollistavat HTML:n ulkopuolisen koodin suorittamisen joko palvelimeen asennettavien moduulien avulla, esimerkiksi PHP:n[3], tai Common Gateway Interfacen[4] (CGI) välityksellä. Kuvassa 3.1 on esitetty kuinka käyttäjän tekemä palvelupyyntö etenee Apache-verkkopalvelimelle asennetun PHP-komponentin läpi käytettyyn kirjastoon ja takaisin. Kuvassa 3.2 on esitetty saman palvelupyynnön suoritus kirjastoon CGI:n välityksellä.



Kuva 3.1: Suorituspolku käyttäjältä kirjastoon käyttäen PHP:ta.

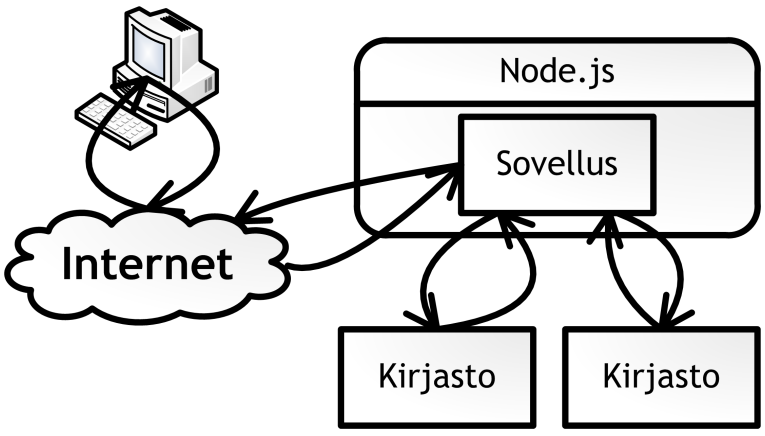
[1] http://httpd.apache.org/
[2] http://www.iis.net/
[3] http://www.php.net/
[4] http://tools.ietf.org/html/rfc3875



Kuva 3.2: Suorituspolku käyttäjältä kirjastoon käyttäen CGI:ta.

Selaimen Same Origin Policy rajoittaa missä ulkopuolista koodia saa ajaa. Tämä rajoittaa kirjastojen käyttämistä siten, että niitä voidaan käyttää vain verkkopalvelimen välityksellä, joka rajoittaa käytettävät kirjastot palvelimeen asennettujen moduulien tukeen.

Vaihtoehtoisesti sovellus voi itse toteuttaa verkkopalvelimen, jolloin sovellus voi käyttää mitä tahansa kirjastoa, johon palvelimella vain on pääsy. Tälläisiä sovelluksia voi rakentaa esimerkiksi Node.js:n[5] päälle, joka on palvelimella ajettava JavaScript-ympäristö, johon on tarjolla lukemattomia valmiita kirjastoja eri tarkoituksiin. Kuvassa 3.3 on esitetty kuinka suorituspolku etenee käyttäen Node.js:ää.

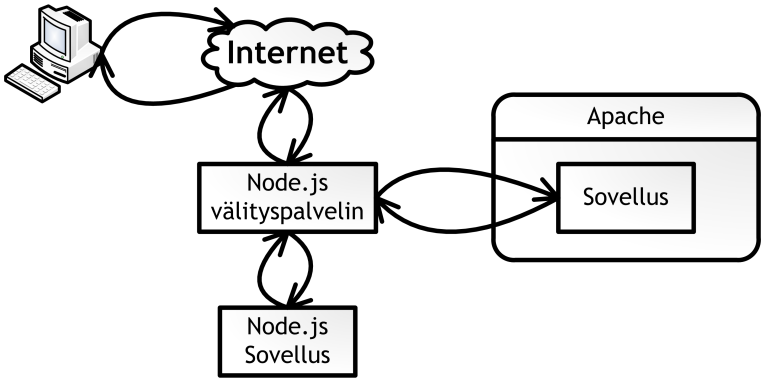


Kuva 3.3: Suorituspolku kirjastoihin Node.js sovelluksella.

Same origin policy rajoittaa kahden eri alustan päällä olevan sovelluksen yhdistämistä, sillä nämä sovellukset kuuntelevat eri portteja ja suoritettava JavaScript-koodi ei

[5] `http://www.nodejs.org/`

saa tehdä kyselyitä toiseen porttiin. Tämän rajoituksen voi kiertää käyttämällä HTTP-välityspalvelinta, joka ohjaa kyselyitä eri sovelluksille, tarjoten vastaukset aina samasta portista. Tämän voi toteuttaa esimerkiksi Node.js:n kirjastolla kuvan 3.4 mukaisesti. Kuvassa välityspalvelin ohjaa kyselyitä sekä Apache-verkkopalvelimelle että toiselle Node.js-sovellukselle, joka toteuttaa verkkopalvelimen.



Kuva 3.4: Node.js HTTP-välityspalvelin.

3.2. Asiakaspään kirjastot

Asiakaspään kirjastot voidaan jakaa käyttötarkoituksen perusteella monipuolisiin sovelluskehyksiin, tietyn kontekstin abstrahoimiseen tarkoitettuihin sekä sovellusaluekohtaisiin kirjastoihin.

3.2.1. Sovelluskehykset

JavaScript-sovelluskehykset mahdollistavat monipuolisia vaihtoehtoja verkkosovelluksen rakentamiseen. Sovelluskehyksiä löytyy aina yleiskäyttöisistä JavaScriptiä yksinkertaistavista kehyksistä widget-kehyksiin ja aina täysivertaisiin käyttöliittymäkehyksiin saakka. Seuraavassa on listattu JavaScript-sovelluskehyksiä eri tarkoituksiin.

jQuery[6] on suunniteltu yksinkertaistamaan selainsovelluksen skriptitoimintoja. Tärkeimpinä ominaisuuksina ovat mm. DOM-elementtien valitseminen, niiden läpikäynti ja muokkaus, tapahtumat, CSS-manipulointi, efektit ja animaatiot sekä Ajax-toiminnallisuus. jQuerya voi laajentaa lisäosilla, ja se abstrahoi selaimien virheellisien toiminnallisuuksien kiertämisen yksityiskohtia. Kehittäjän näkökulmasta tämä näkyy yhtenäisenä rajapintana.

Dojo Toolkit[7] on avoimen lähdekoodin modulaarinen sovelluskehys, joka on suunniteltu helpottamaan Ajax-pohjaisten verkkosovellusten nopeaa kehittämistä. Se tarjoaa

[6] `http://jquery.com/`
[7] `http://dojotoolkit.org/`

tuen usealle erilaiselle asynkroniselle kommunikaatiolle, paketointijärjestelmän modulaarisuutta silmällä pitäen ja vaihtoehtoja sekä asiakas- että palvelinpuolen tietosäilöjen toteuttamiseen. Lisäksi sovelluskehys tarjoaa kattavan tuen graafisille vimpaimille (engl. widget), joilla voi tehdä yhteneviä elementtejä kuten kalentereita, valikoita ja lomakkeita verkkosovellukseen. Kuvassa 3.5 on esitetty Dojo Toolkitin tuottamat kalenteri- ja dialogivimpaimet.

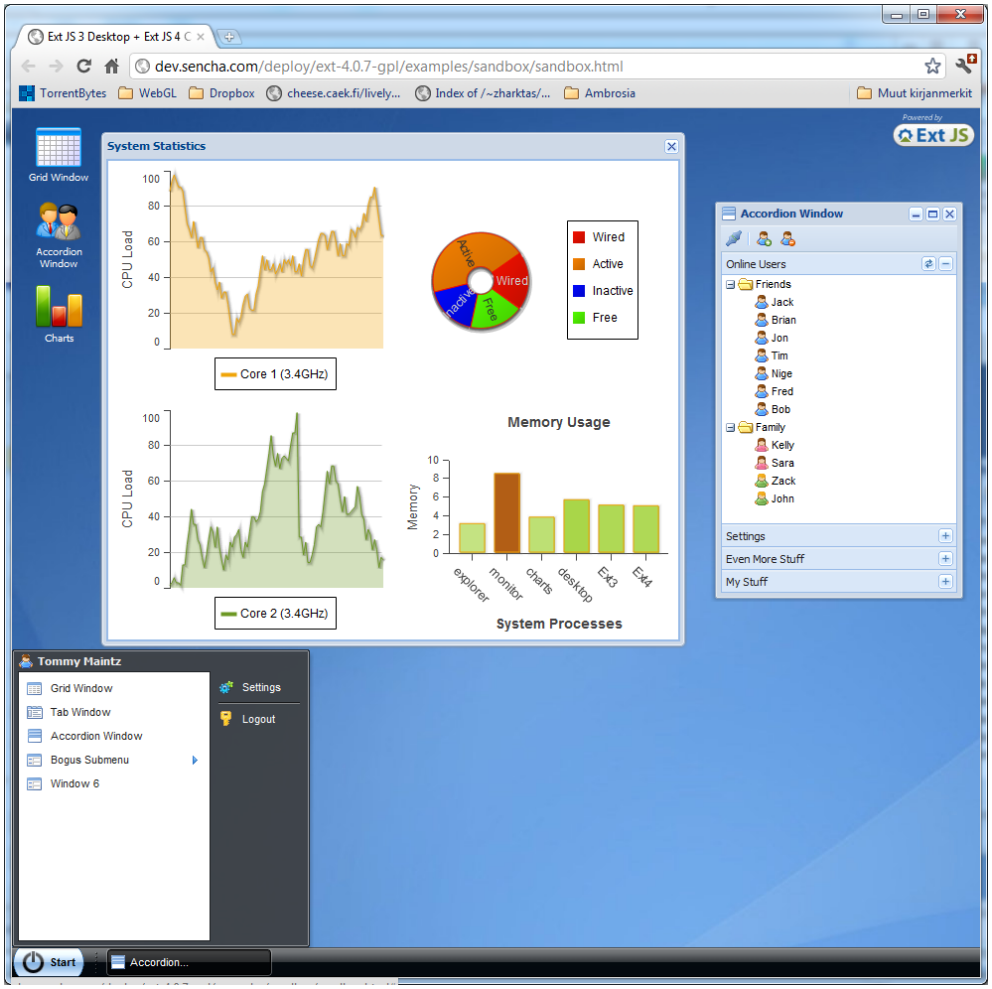
Kuva 3.5: Dojo Toolkitin tuottamia vimpaimia.

ExtJS[8] on kaupallinen sovelluskehys interaktiivisten verkkosovelluksien rakentamiseen. Sen tärkeimpinä ominaisuuksina ovat muokattavat widget-komponentit, joilla voi rakentaa koko käyttöliittymän. Useimmat komponentit pystyvät itsenäisesti kommunikoimaan asynkronisesti palvelimen kanssa. ExtJS tarjoaa myös verkkosovelluksille tuen modaalisiin ikkunoihin, käyttäjän syötteen validointiin ja tilan hallintaan. Lisäksi sovelluskehys tarjoaa mahdollisuuden DOM-hakuihin ja tietosäilöihin, joihin voi tallentaa sekä JSON että XML-muotoista dataa. Kuvassa 3.6 on esitetty ExtJS:llä toteutettu kokonainen käyttöliittymä.

3.2.2. Sovellusaluekohtaiset kirjastot

Toiminnallisuuksia toteuttavat kirjastot on kehitetty johonkin tiettyyn kontekstiin. Niillä voidaan toteuttaa yksityiskohtainen sovellus, kuten tässä diplomityössä on tehty, tai geneerisemmin jokin komponentti, joka voidaan liittää osaksi toista sovellusta helposti. Tässä diplomityössä käyttöliittymä on toteutettu kirjastona, joka voidaan lisätä palveluun ja se tuottaa samanlaisen käyttöliittymän kyseiseen palveluun. Seuraavassa on listattu joitakin geneerisiä käyttökohteita, joihin kirjastoja on toteutettu.

[8] `http://www.sencha.com/products/extjs/`



Kuva 3.6: ExtJS:llä toteutettu käyttöliittymä.

Matematiikka. JavaScript tarjoaa matemaattiset perusoperaatiot, trigonometrian ja pyöristykset, jotka riittävät melkein kaikkeen verkkosovelluksessa tarvittavaan laskentaan [14]. Kirjastoilla toteutetaan tunnettuja laskukaavoja, joita sovelluskohtaisesti usein tarvitaan. Matriisilaskentaan on tehty useita kirjastoja[9,10], joissa on toteutettu 3D-ohjelmoinnissa tarvittuja matriisilaskuja.

Fysiikkakirjastot ovat matemaattisten kirjastojen sovellus. Näissä laskennallisesti simuloidaan reaalimaailman fysiikkaa, joko tarkasti tai sopivasti oikoen nopeuden vuoksi. JavaScriptin fysiikkakirjastot toteuttavat reaaliaikaisen nopean simuloinnin, jotka voi sisällyttää omaan sovellukseen. Näitä kirjastoja ovat mm. JigLibJS[11] ja Box2DJS[12].

Visualisointiin on tarjolla kirjastoja, jotka piirtävät graafeja annetusta aineistosta. Edellä esitetyt visualisointiin tähtäävät kirjastot soveltuvat kolmiulotteisiin visualisoin-

[9] `http://code.google.com/p/glmatrix/`
[10] `http://sylvester.jcoglan.com/`
[11] `http://www.jiglibjs.org/`
[12] `http://box2d-js.sourceforge.net/`

teihin. Kaksiulotteiseen visualisointiin tarkoitettuja kirjastoja ovat mm. Processing.js[13] ja Paper.js[14].

3.2.3. WebGL-perustaiset kirjastot

JavaScriptin monimutkaisemmat toteutukset voidaan abstrahoida kirjaston taakse ja tähän onkin lukuisia valmiita toteutuksia eri toiminnoille. Useat sovelluskehyksetkin abstrahoivat yleisimpiä toimintoja, joista tyypillisimpiä esimerkkejä on Ajax-kutsujen yksinkertaistaminen. Ohjelmassa 3.1 on esitetty kuinka Ajax-kutsu tehdään puhtaalla JavaScriptillä, ja ohjelmassa 3.2 on esitetty kuinka sama tehdään käyttäen jQuery-sovelluskehystä.

Ohjelma 3.1: Puhtaalla JavaScriptillä toteutettu Ajax-pyyntö

```
var http = false;

if(navigator.appName == "Microsoft Internet Explorer") {
  http = new ActiveXObject("Microsoft.XMLHTTP");
} else {
  http = new XMLHttpRequest();
}

http.open("GET", "test.txt");
http.onreadystatechange=function() {
  if(http.readyState == 4) {
     document.getElementById('test').innerHTML = http.responseText;
  }
}
http.send(null);
```

Ohjelma 3.2: jQuery-sovelluskehyksellä toteutettu Ajax-pyyntö

```
$.get('test.txt', function(data) {
  $('test').html(data);
});
```

Puhtaan WebGL:n vaatima suhteellisen työläs toteutus helpottuu käyttämällä sopivaa WebGL-kirjastoa toteutukseen. Kirjastot keskittyvät eri tavoitteisiin ja kohdekäyttäjäryhmään. Seuraavassa on listattu tärkeimpiä WebGL-kirjastoja, joilla on kullakin omat tarkoituksensa.

GLGE[15] on suunniteltu helposti lähestyttäväksi kirjastoksi, joka mahdollistaa 3D-sovelluksen tuottamisen ilman kattavaa tuntemusta 3D-ohjelmoinnista. Kirjastosta löytyvät kaikki 3D-ohjelmoinnin peruskomponentit: objektit, materiaalit, tekstuurit, valot, kamerat ja niin edelleen.

Three.js[16] tarjoaa kevyen 3D-moottorin, jota voi käyttää ilman WebGL:n syvällistä tuntemusta. Se on alhaisemmalla tasolla kuin GLGE, jolloin 3D-näkymän muodostamiseen on enemmän vaihtoehtoja, kuten materiaalin valaistukseen on useita vaihtoehtoja.

[13] http://processingjs.org/
[14] http://paperjs.org/
[15] http://www.glge.org/
[16] https://github.com/mrdoob/three.js/

Alhaisempi taso tosin vaatii myös kattavamman tuntemuksen 3D-ohjelmoinnista. Three.js voi renderöidä canvas- ja svg-elementteihin käyttäen näiden rajapintoja sekä WebGL:ää.

TDL[17] on hyvin alhaisen abstraktiotason kirjasto, joka keskittyy nopeuteen helppokäyttöisyyden sijaan. Sävyttimet ja matriisilaskut täytyy toteuttaa itse, joten kirjaston käyttäminen vaatii syvällistä 3D-ohjelmoinnin tuntemusta.

C3DL[18] on kirjasto, jonka tavoitteena on 3D-sovellusten helppo toteuttaminen. Se tarjoaa luokat 3D-matematiikalle, näkymälle sekä objekteille, jotka tekevät WebGL:stä saavutettavamman kehittäjille, jotka eivät sisäistä 3D-ohjelmoinnin matemaattisia ominaisuuksia.

CopperLicht[19] on kaupallinen kirjasto, jonka ilmaisversiolla voi esittää maksullisella CobberCube-editorilla luodut 3D-näkymät. Editorilla voi luoda näkymiä ilman ohjelmointia ja kirjaston tarkoitus onkin enemmän 3D-visualisoinnissa kuin 3D-sovellusten toteuttamisessa.

O3D[20] oli alun perin selaimeen asennetteva lisäosa, joka WebGL:n myötä kehittyi WebGL-kirjastoksi. Kirjasto olettaa kehittäjältä syvällisen tuntemuksen 3D-ohjelmoinnista, jolloin sävyttimet ja matriisilaskut tarvitsee toteuttaa itse

SceneJS[21] tarjoaa, muista kirjastoista poiketen, JSON-rajapinnan JavaScript-rajapinnan sijaan. Kirjasto rakentuu Scene Graph API:n päälle, jossa määritellään näkymän rakenne JSON-solmuina. Solmuissa määritetään kaikki ominaisuudet sisältäen kamerat, valot ja objektit sijainteineen. SceneJS ei vaadi 3D-ohjelmoinnin tuntemusta muilta osin kuin peruskäsitteiltään.

SpiderGL[22] on 3D-objektien visualisointiin tarkoitettu kirjasto eikä sellaisenaan sovellu monimutkaisempiin 3D-sovelluksiin. Kirjasto tarjoaa matemaattiset rutiinit, geometriset rakenteet, WebGL:llä renderöinnin, mallien muokkaamisen ja käyttöliittymäelementtien interaktion.

X3DOM[23] tähtää X3D-mallien esittämiseen HTML5-ympäristössä. Ymmärrystä 3D-ohjelmoinnista ei vaadita, joten kirjastolla pystyy nopeasti esittämään yksinkertaisen 3D-sovelluksen. Kirjasto tukee kaikkia yleisiä 3D-näkymien perusominaisuuksia.

Jax[24] on suunniteltu WebGL sovellusten kehittämiseen ja se rakentuu MVC-suunnittelumallin päälle. Kirjasto tarjoaa monipuoliset rajapinnat 3D-näkymän muodostamiseen ja kattavat ominaisuudet näkymän muokkaamiseen. Perustuntemus 3D-ohjelmoinnista riittää ja kirjastolle löytyy kattava dokumentaatio, mikä on poikkeus WebGL-kirjastojen joukossa.

[17] `https://github.com/greggman/tdl`
[18] `http://www.c3dl.org/`
[19] `http://www.ambiera.com/copperlicht/`
[20] `http://code.google.com/p/o3d/`
[21] `http://scenejs.org/`
[22] `http://spidergl.org/`
[23] `http://www.x3dom.org/`
[24] `http://jaxgl.com/`

WebGL-kirjaston valinta riippuu tarpeista ja kehittäjän taitotasosta, osa kirjastoista soveltuu visualisointeihin ja osa hyvinkin kehittyneiden ja monipuolisten 3D-sovellusten kehittämiseen. Yhtä ainoaa oikeaa valintaa jokaiseen sovellukseen tuskin on nykyisellään olemassa.

3.3. GLGE

GLGE[35] on JavaScript-kirjasto, jolla toteutettiin diplomityön tekninen kontribuutio. Kirjasto tähtää WebGL:n piilottamiseen kehittäjältä, jolloin kehittäjä voi keskittyä sisällön tuottamiseen. Kirjastoa kehittää ja ylläpitää pääasiallisesti Paul Brunt, jonka lisäksi myös muut kehittäjät voivat lähettää korjausehdotuksia kirjaston versionhallintaan[25], jossa muutokset ovat viikoittaisia.

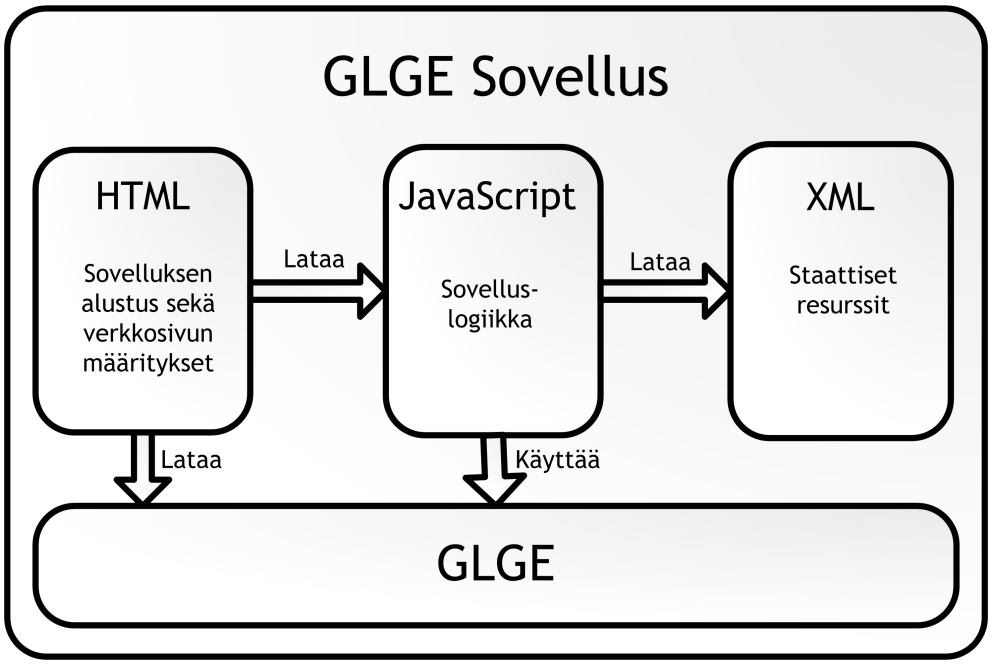
GLGE:n pääominaisuudet on esitelty seuraavassa:

- Keyframe-animaatiot, joissa määritetään animaatioiden alku- ja loppupiste, joista kirjasto interpoloi kaikki välissä olevat siirtymät.
- Valaistus, joka tarkoittaa valonlähteiden määritystä ja niiden vaikutuksen laskemista materiaaleihin.
- Animoidut materiaalit, jotka mahdollistavat materiaalinen muodostukseen vaikuttamisen ohjelmallisesti, esimerkiksi käyttäen materiaalina video- tai canvas-elementtejä.
- Luurankomallien animointi mahdollistaa luonnolliselta näyttävän liikkumisen hahmoille siten, että liikkumista ei ole etukäteen animoitu.
- Collada-tuki mahdollistaa olemassa olevien Collada-mallien käyttämisen objekteina.
- Tekstin renderöinti on mahdollistettu käyttäen canvas-elementtejä ja niille kirjoitettua tekstiä käyttäen.
- Sumu mahdollistetaan sumulaskennalla, joka laskee kameran etäisyyden piirrettävästä kohteesta ja kuinka mahdollinen sumu sen realistiseen piirtoon vaikuttaa.
- Varjot lasketaan valonlähteiden perusteella, jolloin valonlähteen tai objektin liikkuessa myös varjo muuttuu.
- Sävytinpohjainen poiminta, joka mahdollistaa objektien poimimisen näkymästä sävyttimien avulla.
- Heijastukset, jotka lasketaan materiaalikohtaisesti, kuvaavat kuinka jokin toinen objekti heijastuu kustakin materiaalista.

[25] `https://github.com/supereggbert/GLGE`

3.3.1. GLGE-sovelluksen rakenne

GLGE-sovellus koostuu kolmesta osasta: HTML-sivusta, joka alustaa GLGE-sovelluksen sekä toteuttaa sovelluksen ulkopuoliset käyttöliittymäelementit, JavaScript-tiedostoista, jotka sisältävät sovelluslogiikan sekä XML-tiedostosta, joka sisältää sovelluksen staattiset resurssit. Kaaviossa 3.7 on kuvattu sovelluksen rakenne. Seuraavassa osat esitellään yksi kerrallaan. [36]



Kuva 3.7: GLGE-sovelluksen rakenne.

HTML. HTML-sivulla alustetaan GLGE-sovellus, määritetään canvas-elementti, johon WebGL renderöidään, ladataan vaadittavat JavaScript-tiedostot, jotka sisältävät GLGE:n toiminnallisuuden ja sovelluslogiikan. Lisäksi HTML-sivulla määritetään 3D-maailman ulkopuoliset käyttöliittymäelementit. Ohjelma 3.3 näyttää kuinka näihin viitataan HTML:ssä.

Ohjelma 3.3: Sivu lataa vaadittavat JavaScript-resurssit, määrittää canvas-elementin WebGL:lle sekä 3D-maailman ulkopuoliset käyttöliittymäelementit

```
<script type="text/javascript" src="GLGE/glge-min.js"></script>
<script type="text/javascript" src="Lively3D/Lively3d.js"></script>

<canvas id="canvas" width="960" height="600"></canvas>
<button onclick="Reload()">Reload</button>
```

JavaScript. JavaScript-tiedostoissa ladataan XML-tiedosto, joka sisältää jäljellä olevat staattiset resurssit, määritetään kuinka näppäimistö- ja hiiritapahtumat käsitellään sekä alustetaan GLGE:n käyttämä canvas-elementti päivittymään tietyllä aikavälillä. Ohjelmassa 3.4 ladataan XML-tiedosto ja päivitetään canvas-elementtiä 10 millisekunnin välein.

Ohjelma 3.4: Javascript lataa staattiset resurssit sisältävän XML:n ja asettaa canvas-elementin renderöimään 3D-maailmaa 10 millisekunnin välein.

```
var doc = new GLGE.Document();
doc.onLoad = function(){
  var renderer = new GLGE.Renderer(document.getElementById("canvas"));
  var scene = doc.getElement("mainscene");
  renderer.setScene(scene);

  function render(){
    renderer.render();
  }

  setInterval(render, 10);
}
doc.load("lively3d.xml");
```

XML. XML-tiedosto sisältää GLGE-sovelluksen staattiset resurssit. Siinä määritellään mm. objektit, materiaalit, tekstuurit, animaatiot, kamerat, valot ja näkymä. Näkymän määrityksessä määritetään näkymässä olevat objektit, objektien teksturoinnit, näkymän valaistus ja kamerat sekä näiden asennot ja sijainnit. Ohjelmassa 3.5 määritetään yksinkertainen kuutio yhdellä valolla ja kameralla.

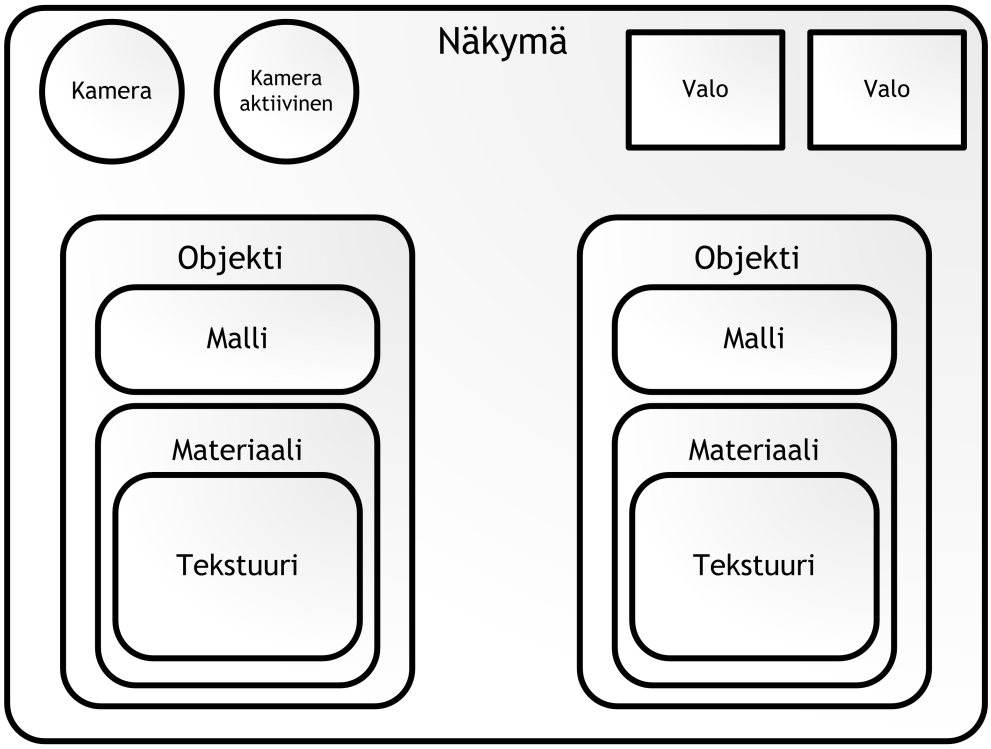
Ohjelma 3.5: Yksinkertainen näkymän sisältävä GLGE:n XML-tiedosto

```
<?xml version="1.0"?>
<glge>
  <mesh id="cube">
    <positions>-4.999998,5.000002,0.000000,5.000000,5.000000,0.000000, ...</positions>
    <normals>-0.000000,-1.000000,-0.000001, ...</normals>
    <uv1>0.000000,0.000000,1.000000, ...</uv1>
    <faces>0,1,2, ...</faces>
  </mesh>
  <material id="cubematerial" specular="0" >
    <texture id="cubetex" src="tex.png"/>
    <material_layer texture="#cubetex" mapinput="UV1" mapto="M_COLOR" />
  </material>
  <camera id="maincamera" loc_y="20" loc_x="1" loc_z="8" rot_z="0" rot_x="1.56"  />
  <scene id="mainscene" uid="scene" camera="#maincamera" >
    <object id="cubeobj" mesh="#cube" loc_z="0" material="#cubematerial" />
    <light id="mainlight" loc_x="0" loc_y="30" loc_z="20" />
  </scene>
</glge>
```

3.3.2. GLGE-näkymä

GLGE-sovelluksessa määritetään yksi tai useampia näkymiä, jotka muodostavat loogisen kokonaisuuden 3D-maailmasta. Näkymä voidaan muodostaa joko staattisesti XML:stä tai dynaamisesti JavaScriptillä. Staattisessa määrityksessä määritetään kaikki sovelluksen aloitustilanteessa oleva näkymän sisältö. Kuvassa 3.8 on tyypillisen perusnäkymän rakenne.

Staattisesti luotua näkymää voidaan täydentää JavaScriptillä, jossa esimerkiksi luodaan objekti staattisista resursseista ja lisätään se näkymään. JavaScriptillä voidaan



Kuva 3.8: Perusnäkymän rakenne.

myös muokata näkymässä olevia resursseja tai luoda kokonaan uusia. Ohjelmassa 3.6 luodaan dynaamisesti uusi objekti käyttäen staattisesti määritettyä mallia, jonka jälkeen se teksturoidaan dynaamisesti ja lisätään näkymään.

Ohjelma 3.6: Dynaamisesti luodun objektin teksturointi ja näkymään lisääminen

```
//Luodaan dynaaminen objekti
var obj = new GLGE.Object();

//objekti käyttää staattista 'cube'-resurssia
obj.setMesh(doc.getElement('cube'));

//luodaan dynaamisesti tekstuurit ja asetetaan se objektiin
var tex = new GLGE.Texture();
tex.setSrc('smiley.png');
var layer = new GLGE.MaterialLayer().setMapinput(GLGE.UV1).setMapto(GLGE.M_COLOR).
    setTexture(tex);
var material = new GLGE.Material().addTexture(tex).addMaterialLayer(layer);
obj.setMaterial(material);

//lisätään objekti näkymään
scene.addObject(obj);
```

Objektit. GLGE:n objekti toimii säiliönä renderöitävälle objektille. Jokaiseen renderöitävään objektiin liittyy malli, joka määrittää objektin kulmapisteet, viivat ja tasot. Lisäksi objektiin voidaan liittää materiaaleja ja animaatioita, joilla mallin ulkonäköön voidaan vaikuttaa. Jokaisella objektilla on sijainti ja asento, joiden mukaan malli sijoitetaan

3D-maailmaan.

Mallit. GLGE:n mallit määrittävät kappaleen muodon. Malli määrittää kappaleen kulmapisteet, viivat ja tasot, jotka voidaan tuottaa esimerkiksi mallinnusohjelmilla.

Materiaalit. GLGE:n materiaalit määrittävät mallin ulkonäön. Materiaalina voidaan käyttää yksinkertaisia värejä sekä tekstuureita, joissa voidaan käyttää kuvia, videoita ja canvas-elementtejä. Lisäksi materiaaleille voidaan määrittää käyttäytyminen valaistuksessa, johon voidaan määrittää erilaisia heijastustapoja kuten spekulaarin ja diffuusin heijastuksen. Materiaali voi myös toimia valon lähteenä, jolloin malli saadaan hohtamaan 3D-ympäristössä.

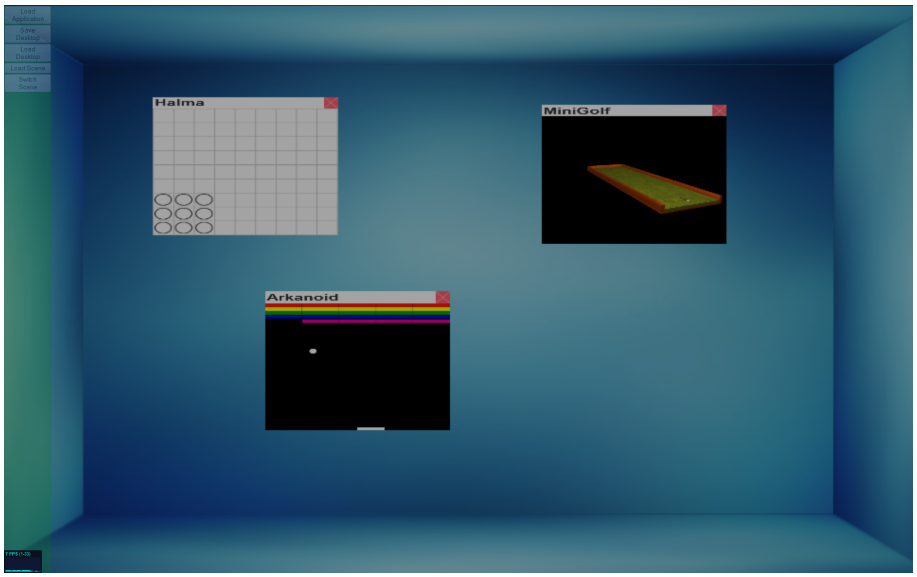
Tekstuurit. GLGE:n tekstuurit ovat kaksiulotteisia kuvia, videoita tai canvas-elementtejä, jotka määrittävät materiaalin värin. Videoiden ja canvas-elementtien sisältö voi muuttua, joka myös päivittyy käytettyyn tekstuuriin.

Valaistus. GLGE tukee neljää eri valaistusmallia: ambienttia valaistusta ja piste-, kohde- sekä suuntavaloja. Ambientissa valaistuksessa kaikkiin ympäristön pisteisiin osuu yhtä voimakas taustavalo, jolloin tuotetaan tasaisia, varjottomia väripintoja. Pistemäisissä valoissa valonlähde ajatellaan pisteeksi, joka säteilee ympäristöönsä. Kohdevaloissa tämä valo on rajattu kartioon, jolloin pinta ei valaistu, jos valonlähteen suunta pintaan nähden poikkeaa yli sallitun kulman verran. Suuntavalot ovat kuin kaukana olevia pistevaloja, jolloin kaikilla merkittävillä suunnilla valonlähteestä on sama suuntavektori.

Kamerat. GLGE:n kamera on siirreltävä ja käänneltävä objekti, joka määrittää katsojan sijainnin. Kameran siirto aiheuttaa näkymän kääntämisen, jonka lopputulos simuloi katsojan liikettä.

4. 3D-IKKUNOINTISOVELLUKSEN TOTEUTTAMINEN

Tämän diplomityön teknisenä kontribuutiona toteutettiin Lively3D-ikkunointisovellus käyttäen pääasiallisesti GLGE-kirjastoa. Lively3D on kolmiulotteinen sovelluskehys, johon on mahdollista upottaa canvas-elementteihin rakennettuja sovelluksia. Nämä canvas-elementit esitetään Lively3D:ssä perinteisinä sovellusikkunoina otsikkokenttineen ja sulkupainikkeineen. Canvas-sovelluksien upottaminen suunniteltiin siten, että sovelluksen kehittäjän tarvitsee kirjoittaa mahdollisimman vähän ylimääräisiä koodirivejä. Lisäksi Lively3D:ssä on mahdollista toteuttaa uusia 3D-ympäristöjä, joissa voidaan toteuttaa sovelluksien esittäminen kehittäjän valitsemalla tavalla. Nämä sovellukset ja 3D-ympäristöt on mahdollista asentaa Dropbox[1]-pilvipalveluun, jossa ne on saatavilla Lively3D:n käyttöön ilman, että Lively3D:n tietorakenteisiin tarvitsee tehdä muutoksia. Lopuksi Lively3D:n työpöydän ja sovelluksien tila voidaan tallentaa dokumenttitietokantaan Node.js-sovelluksen välityksellä. Kuvassa 4.1 on esitetty Lively3D kolmella auki olevalla sovelluksella.

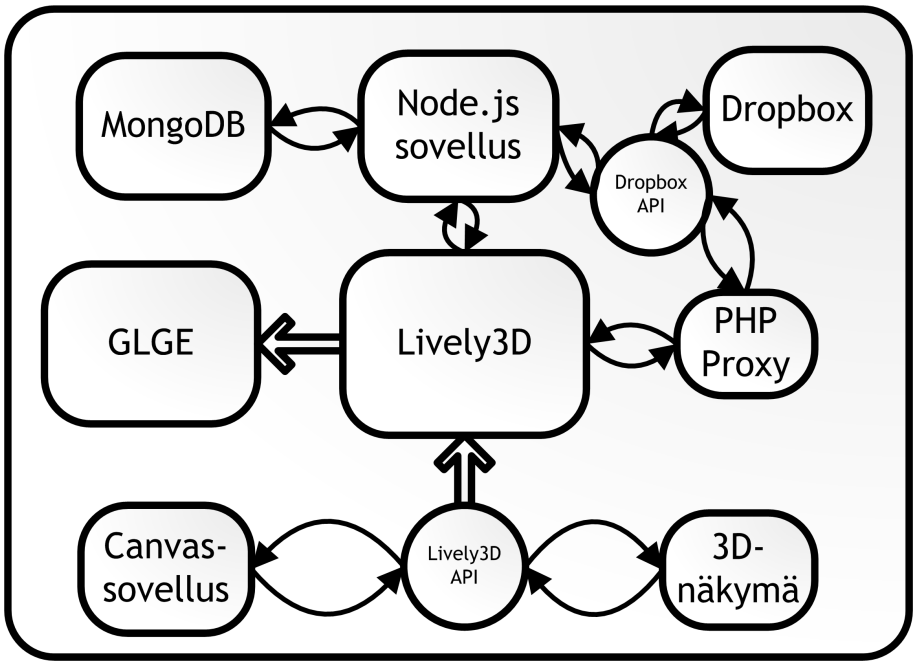


Kuva 4.1: Lively3D

[1] https://www.dropbox.com/

4.1. Lively3D

Lively3D:n tavoitteena oli tutkia mitä ongelmia ja rajoitteita uudet verkkoteknologiat ja selain asettavat työpöytämaisen sovelluksen toteuttamisella ja kuinka näitä voidaan ratkaista. Lively3D koostuu JavaScript-kirjastosta, josta näkyy kehittäjälle yksinkertainen rajapinta (Lively3D API), kolmannen osapuolen toteuttamista canvas-sovelluksista sekä 3D-näkymistä, joille on toteutettu Lively3D:n vaatimat funktiot, Dropbox-rajapinnasta, joka mahdollistaa canvas-sovelluksien käyttöönoton ilman pääsyä palvelimelle, jossa Lively3D sijaitsee, sekä Node.js-sovelluksesta, joka mahdollistaa tiedon persistoinnin MongoDB-tietokantaan[2]. Kuvassa 4.2 esitetään kuinka nämä komponentit asettuvat toisiinsa nähden.



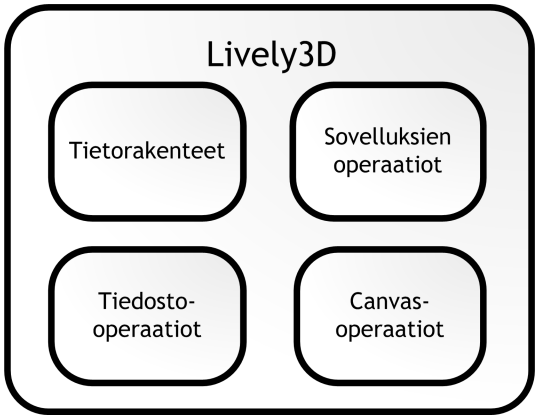
Kuva 4.2: Lively3D:n rakenne.

Lively3D perustuu vahvasti GLGE:hen, jota käytetään sisäisesti 3D-ympäristön toteutukseen. Tämä pyritään kuitenkin piilottamaan Lively3D:n käyttäjältä mahdollisuuksien mukaan. Lively3D alustetaan antamalla alustusfunktiolle HTML:n canvas-elementti, johon kirjasto renderöi oletus 3D-ympäristön. Ympäristön luonnin jälkeen kirjasto tarjoaa rajapinnat uusien ympäristöjen ja canvas-sovelluksien tuomiseen Lively3D:hen.

Lively3D:n sisäinen rakenne voidaan jakaa kuvan 4.3 mukaisesti tietorakenteisiin ja sovelluksien, tiedostojen sekä canvas-elementin operaatioihin. Nämä käsitellään seuraavaksi.

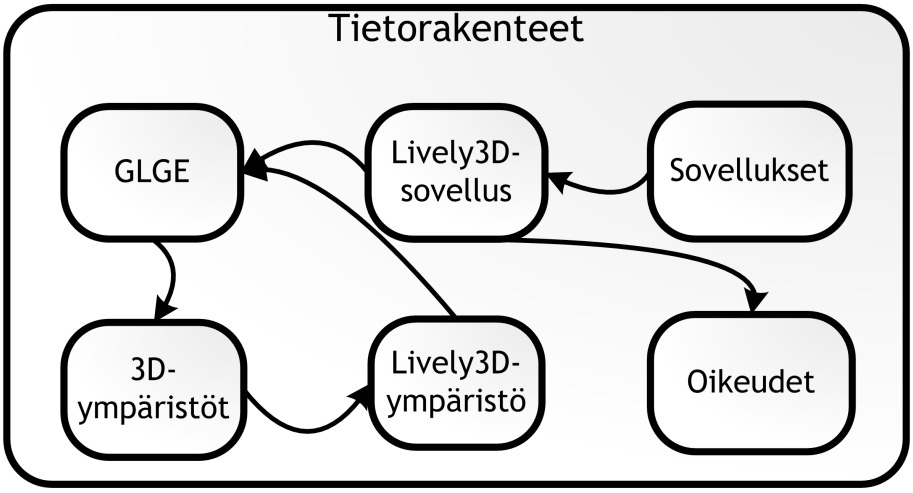
Tietorakenteet. Lively3D:n tietorakenteet muodostavat sovelluskehyksen rungon. Tietorakenteet voidaan jakaa kuvan 4.4 mukaisesti. Kuvan GLGE sisältää edellä esitellyn

[2] `http://www.mongodb.org/`



Kuva 4.3: Lively3D:n sisäinen rakenne.

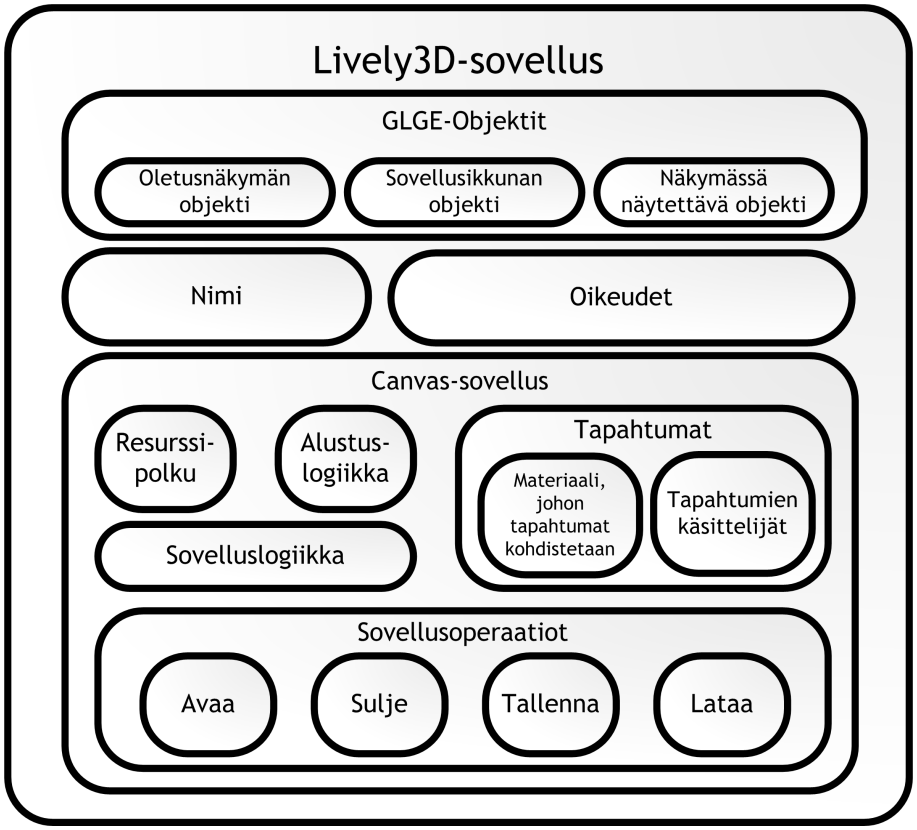
GLGE-kirjaston XML-dokumentin, renderöijän sekä käyttäjän lataamat 3D-ympäristöt, joita on oletuksena ladattuna vain yksi. 3D-ympäristöstä tietorakenteeseen tallennetaan edellä esitelty GLGE-näkymä sekä ympäristöön liittyvä sovelluslogiikka.



Kuva 4.4: Lively3D:n tietorakenteet.

Lively3D-sovellus. Lively3D:n sovellus koostuu canvas-elementtiin toteutetusta sovelluksesta ja sen Lively3D:ssä esittämiseen tarvittavasti tietorakenteista. Nämä tietorakenteet esitellään kuvassa 4.5. Tietorakenteeseen tallennetaan GLGE-objekteja, jotka sisältävät oletusnäkymän objektin, luodun sovellusikkunan, johon sovellus renderöidään, sekä viittauksen objektiin, joka kyseisellä hetkellä on renderöitynä. Lisäksi tietorakenteeseen tallennetaan sovelluksen nimi ja GLGE-objekteihin liittyvät oikeudet. Canvas-sovelluksesta tietorakenteeseen tallennetaan alustus- ja sovelluslogiikka, polku, josta sovelluksen käyttämät resurssit löytyvät, tapahtumienkäsittelijät ja viite sovellusikkunan teksturointimateriaalin käyttäjäinteraktiota varten sekä sovelluksen operaatiot, joilla sovellus voi reagoida Lively3D:ssä tehtyihin toimintoihin.

Lively3D:n oikeudet on määritetty merkkijonoliteraaleina tietorakenteessa. Literaaleja



Kuva 4.5: Lively3D-sovelluksen tietorakenne.

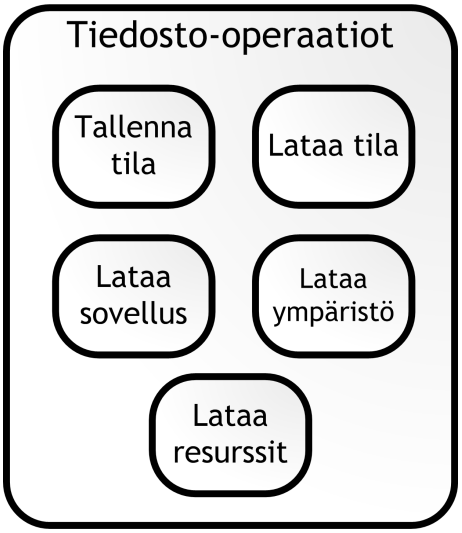
voidaan liittää Lively3D-sovelluksessa kyseisen sovelluksen objekteihin. Oikeuksia tarkistetaan sovellusoperaatioiden aikana, jolloin sovelluksen objekteihin liitettyjä literaaleja verrataan tietorakenteen literaaleihin. Sovellusoperaatio sallitaan, jos literaalit vastaavat toisiaan.

Sovellusoperaatiot. Lively3D:n sovellusoperaatiot koostuvat uusien Lively3D-sovelluksien lisäämis-, haku- ja Lively3D-sovelluksiin kohdistuvista toiminnoista. Nämä toiminnot on esitetty kuvassa 4.6. Kuvan operaatioille, lukuun ottamatta lisäämis- ja hakutoimintoja, annetaan Lively3D-sovellus, jolloin sovelluskehys toteuttaa halutun toiminnon. Toimintojen toteutus kohdistetaan käytössä olevalle 3D-ympäristölle, jolloin 3D-ympäristö voi reagoida toimintoon haluamallaan tavalla. Avaamis- ja sulkemistoiminnot vaihtavat sovellusta kuvaavan GLGE-objektin ja sovellusikkunan välillä, samalla kun sijainti- ja asentotiedot asetetaan kunkin ympäristön mukaisesti. Suurennus- ja pienennystoiminnot muuttavat sovellusikkunan kokoa siten, että se täyttää isomman alueen käytössä olevasta canvas-elementistä. Hakutoiminto hakee annetun GLGE-objektin sisältävän Lively3D-sovelluksen, jolle voidaan suorittaa muita toimintoja haun jälkeen. Lisäämisoperaatio luo Lively3D-sovelluksen tietorakenteet ja niihin liittyvän canvas-sovelluksen.

Tiedosto-operaatiot. Lively3D:n tiedosto-operaatiot kattavat toiminnot, joilla luetaan ja kirjoitetaan tiedostoja joko suoraan verkkopalvelimelta tai PHP- ja Node.js-

Kuva 4.6: Lively3D:n sovellusoperaatiot.

välityspalvelimien kautta. Kuvassa 4.7 on kuvattu toiminnot, joilla tiedostoja käsitellään.

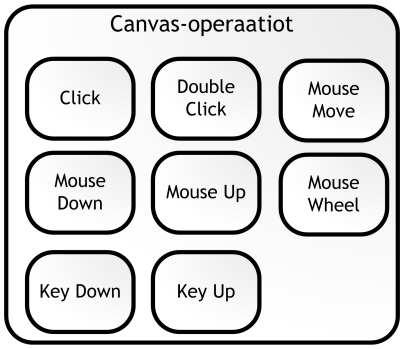


Kuva 4.7: Lively3D:n tiedosto-operaatiot.

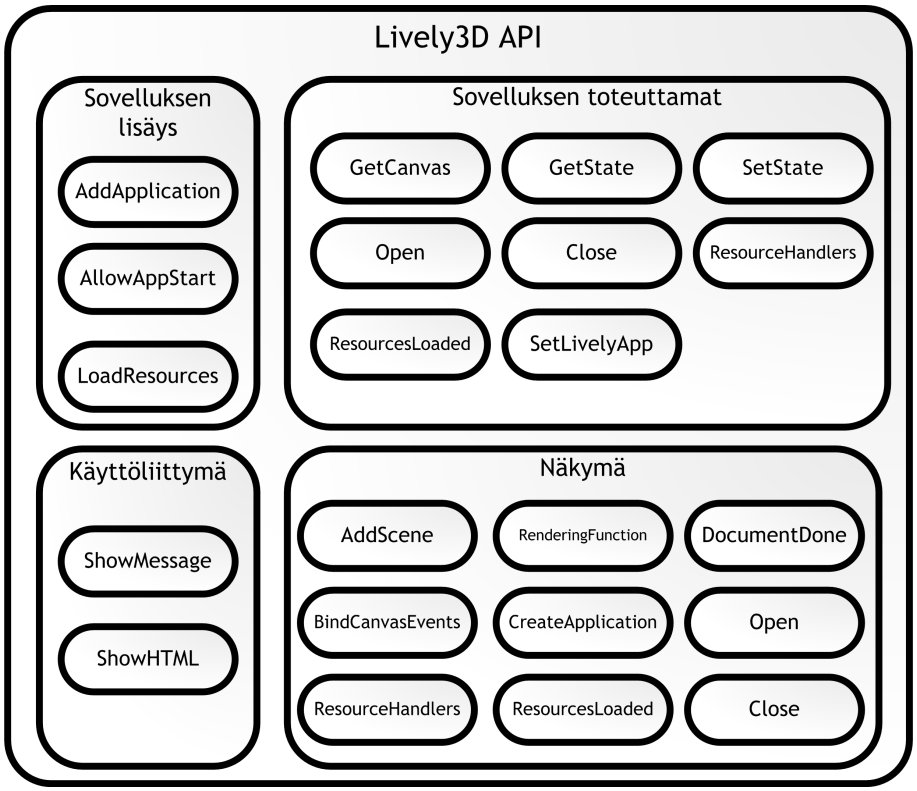
Tilan tallentamis- ja latausoperaatiot kommunikoivat joko PHP-välityspalvelimen tai Node.js-sovelluksen kanssa, jotka käyttävät JSON-objekteja tilan tallentamiseen. PHP-välityspalvelin tallentaa JSON-objektin tiedostoon ja Node.js-sovellus dokumenttitietokantaan. Sovelluksien ja ympäristöjen lataamistoiminnossa ladataan JavaScript-tiedostoja, joiden sisältämä koodi ajetaan lataamisen yhteydessä. Resurssien latauksessa resurssit välitetään kullekin resurssien käsittelijälle, jotka käsittelevät resurssit haluamallaan tavalla.

Canvas-elementin operaatiot. Lively3D:n esittämiseen käytetty canvas-elementti sisältää tapahtumia, jotka on kuvattu kuvassa 4.8. Näille tapahtumille on oletustoiminnalli-

suudet toteutettu oletusympäristöön, mutta ne voidaan uudelleen määrittää uuteen ympäristöön vaihtaessa. Oletustoiminnallisuuksissa canvas-elementtiin kohdistetuilla tapahtumilla vaikutetaan ympäristön tilaan. Hiiritapahtumilla voidaan siirrellä 3D-objekteja ympäristössä sekä avata, sulkea, suurentaa ja pienentää sovelluksia. Näiden lisäksi sovelluksiin kohdistetut hiiri- ja näppäimistötapahtumat välitetään sovelluksen tapahtumien käsittelijöille. Uudelleen määrityksessä ei tarvitse määrittää kuin halutut toiminnot uudelleen, määrittämättömät toiminnot säilyttävät edellisen toimintonsa.



Kuva 4.8: Lively3D:n canvas-operaatiot.



Kuva 4.9: Lively3D:n rajapinnat ulkoisiin sovelluksiin.

Lively3D API. Lively3D:n rajapinnat ulkoisiin sovelluksiin on esitetty kuvassa 4.9.

Taulukko 4.1: Lively3D rajapinta

Sovelluksen lisäys	
AddApplication	Lisää uuden sovelluksen ympäristöön.
AllowAppStart	Sallii sovelluksen käyttämisen ympäristössä, kutsutaan resurssien lataamisen jälkeen.
LoadResourses	Lataa annetun listan mukaiset resurssit ja palauttaa osoitteet ladatuille resursseille.

Sovelluksen toteuttamat	
GetCanvas	Palauttaa sovelluksen käyttämän canvas-elementin.
GetState	Palauttaa sovelluksen tilaobjektin.
SetState	Asettaa sovelluksen tilan annetulla objektilla.
Open	Kutsutaan, kun käyttäjä avaa sovelluksen ympäristössä.
Close	Kutsutaan, kun käyttäjä sulkee sovelluksen ympäristössä.
ResourseHandlers	Käsittelevät Lively3D:n lataamien resurssien osoitteet.
ResourcesLoaded	Kutsutaan, kun Lively3D on ladannut annetun resurssilistan. Ilmoittaa sovellukselle mikä resurssi on ladattu.
SetLivelyApp	Asettaa canvas-sovellukseen ympäristössä luodun Lively3D-sovelluksen objektin, jotta canvas-sovelluksesta voidaan käyttää Lively3D:n toimintoja.

Näkymä	
AddScene	Lisää uuden näkymän ympäristöön.
RenderingFunction	Kutsutaan animointisilmukan aikana, mahdollistaa näkymän päivittämisen jokaisella renderöinnillä.
DocumentDone	Kutsutaan, kun Lively3D on ladannut näkymään liittyvän XML-dokumentin.
BindCanvasEvents	Kutsutaan, kun näkymää vaihdetaan, mahdollistaa hiiritapahtumien uudelleenasettamisen.
CreateApplication	Kutsutaan uutta sovellusta lisätessä. Luo sovellusta kuvaavan GLGE-objektin ja lisää sen näkymään.
Open	Kutsutaan, kun käyttäjä avaa sovelluksen näkymässä.
Close	Kutsutaan, kun käyttäjä sulkee sovelluksen näkymässä.
ResourceHandlers	Käsittelevät Lively3D:n lataamien resurssien osoitteet.
ResourcesDone	Kutsutaan, kun Lively3D on ladannut annetun resurssilistan. Ilmoittaa sovellukselle mikä resurssi on ladattu.

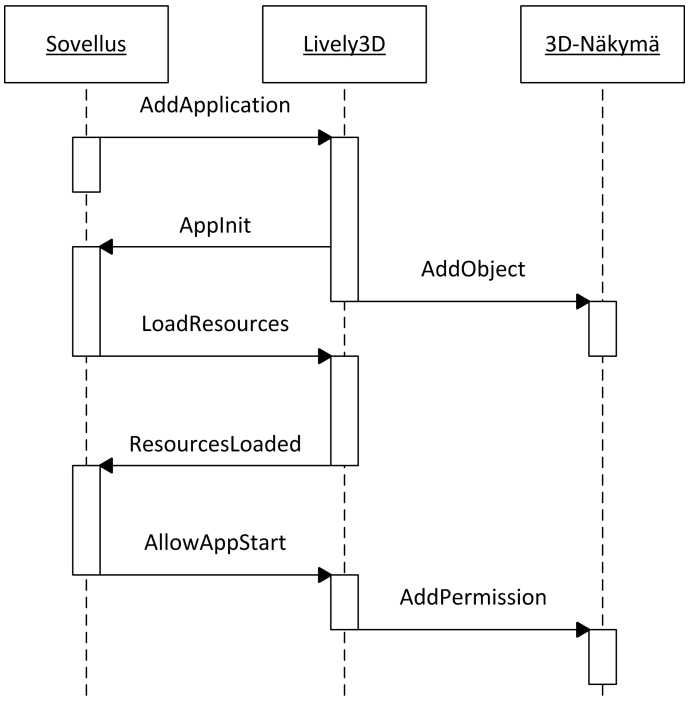
Käyttöliittymä	
ShowMessage	Näyttää annetun viestin käyttöliittymässä.
ShowHTML	Näyttää annetun HTML:n käyttöliittymässä

Rajapinnat on jaettu neljään kategoriaan: sovelluksen lisäämiseen vaadittaviin, sovelluksen toteuttamiin, 3D-näkymiin liittyviin sekä käyttöliittymään kohdistuviin rajapintoihin. Sovelluksen lisäämisessä rajapintoja kutsutaan lisättävästä canvas-sovelluksesta. Sovelluksen toteuttamia rajapintoja kutsutaan Lively3D:stä riippuen käyttäjän vuorovaikutuksesta. Näkymän rajapinnoista *AddScene*-funktio on ainoa, jota näkymää lisättäessä pitää kutsua, muut rajapinnat ovat näkymän toteuttamia, joita Lively3D kutsuu. Käyttöliittymärajapinnat mahdollistavat sovelluksien ja näkymien lähettävän viestejä sekä HTML:ää Lively3D:lle, joka näyttää sen käyttäjälle valmiiksi toteutetuilla dialogeilla. Taulukossa 4.1 on esitetty rajapintojen kuvaukset.

4.2. Sovellukset

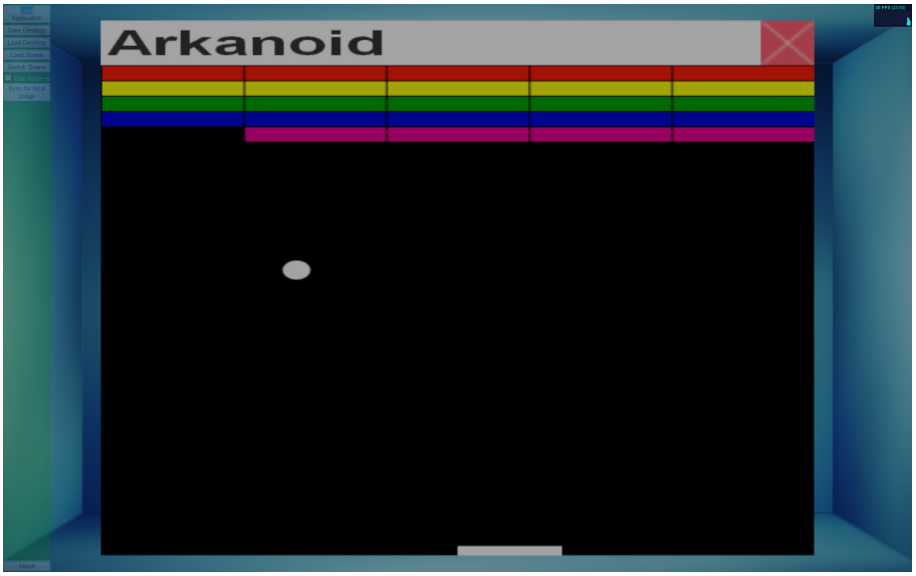
Canvas-sovellukset ovat canvas-elementtiin renderöityjä sovelluksia, jotka voidaan renderöidä sekä kaksi- että kolmiulotteisena. Sovellus toteutetaan kokonaisuudessaan JavaScriptillä omassa nimiavaruudessaan ja canvas-elementtiä käytetään sovelluksen renderöimiseen joko canvas-elementin 2D-rajapinnalla tai WebGL:llä. Lisäksi elementtiin määritetään tapahtumien kuuntelijoita, jotka reagoivat käyttäjän toimintoihin kuten hiiren liikkeisiin ja näppäimistön painalluksiin. Sovellukset voidaan integroida Lively3D:n kanssa siten, että canvas-elementtiä käytetään 3D-objektin tekstuurina ja objektiin kohdistetut hiiri- ja näppäimistötapahtumat välitetään kyseisen canvas-elementin kuuntelijoille. Integrointi on suunniteltu siten, että sovelluksen kehittäjän tarvitsee integroidessaan toteuttaa mahdollisimman vähän ylimääräisiä koodirivejä.

Integrointi aloitetaan antamalla *AddApplication*-funktiolle sovelluksen koodi, sen alustusfunktio sekä nimi. Lively3D alustaa sovelluksen, jonka alustusfunktio kutsuu Lively3D:n tarjoamaa resurssienlataajaa sovelluksen kehittäjän määrittämillä resursseilla. Kun resurssit ovat käytettävissä, Lively3D kutsuu sovelluksen kehittäjän toteuttamaa resurssien käsittelijää, joka voi käsitellä resurssit kehittäjän haluamalla tavalla. Kun kaikki resurssit on käsitelty, sovellus on kokonaan alustettu, jolloin asiasta tiedotetaan Lively3D:tä, joka sallii sovelluksen käyttämisen. Tämä prosessi on esitetty kuvassa 4.10. Rinnakkain sovelluksen alustamisen kanssa, Lively3D luo 3D-objektit jokaiseen ladattuun 3D-ympäristöön kutsumalla ympäristöjen objektin luontifunktioita.



Kuva 4.10: Lively3D:n ja sovelluksen väliset takaisinkutsut.

Liitteessä B on listattu Bill Millin esimerkeissä esitetty Arkanoid-peli[3] integroituna Lively3D:n ympäristöön. Integroinnin lopputulos on esitetty kuvassa 4.11.



Kuva 4.11: Arkanoid-peli integroituna Lively3D:n ympäristöön.

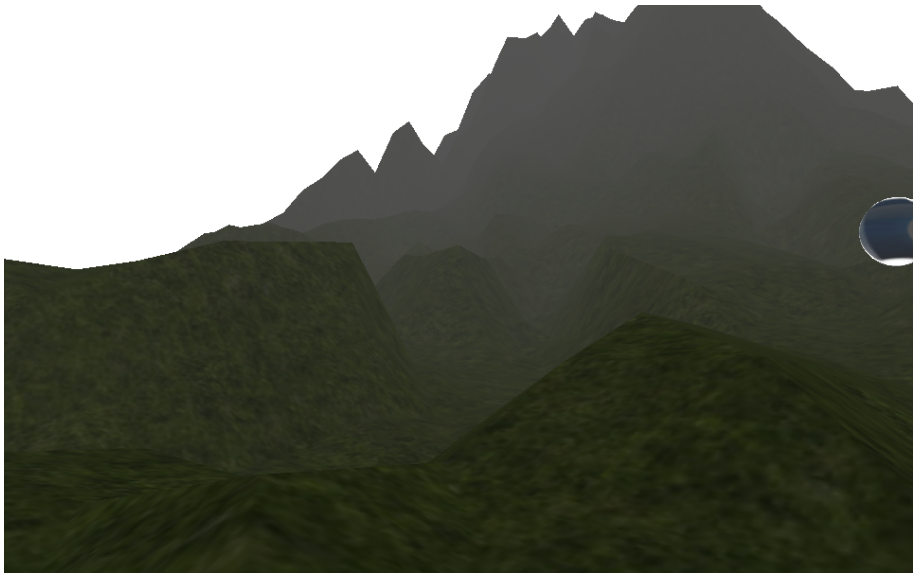
4.3. 3D-ympäristöt

Lively3D:ssä 3D-ympäristöt ovat kolmiulotteisia näkymiä, jotka koostuvat kamerasta, rajoittamattomasta määrästä objekteja ja sovelluslogiikasta, joka määrittää käyttäjän interaktion ja automaattiset prosessit. 3D-ympäristöt toteutetaan käyttäen GLGE:tä, joka tarjoaa JavaScript rajapinnan, jolla voi vaikuttaa ympäristön eri osiin. GLGE:n pääkomponentti on animointisilmukka, joka on toteutettu Lively3D:ssä käyttäen selaimen requestAnimationFrame-funktiota. Funktio ajoittaa selaimen piirtämään canvas-elementin uudestaan seuraavan ruudun päivityksen yhteydessä. Tämä estää animaation päivittämisen, kun selainikkuna on piilotettu taustalla olevaan välilehteen, säästäen siten tietokoneen resursseja. Animointisilmukka Lively3D:ssä laskee edellisestä renderöinnistä kuluneen ajan, päivittää ympäristön näkymän määrityksien mukaan ja kutsuu GLGE:n renderöintifunktiota.

Ulkopuolinen kehittäjä voi tuottaa uusia 3D-ympäristöjä GLGE:llä. Ympäristön alkuperäinen tila määritetään XML-tiedostossa, joka voidaan tuottaa mallinnusohjelmalla kuten Blenderillä[4]. XML määrittää kaikki kamerat, valot, objektit, mallit, materiaalit ja tekstuurit joita käytetään ympäristössä. Määrittääkseen uuden ympäristön, kehittäjän täytyy tuottaa JavaScript-tiedosto, jonka muuttujaan on tallennettu XML-tiedoston nimi. Tämän lisäksi kehittäjän täytyy tuottaa identifioivat nimet ympäristölle, kameroille ja ympäristön

[3] `http://billmill.org/static/canvastutorial/`
[4] `http://www.blender.org/`

sisäisesti käytetyille objekteille. Kuvassa 4.12 on esitetty vaihtoehtoinen 3D-ympäristö, jossa 3D-maailma koostuu maastosta ja sovellukset ovat palloja, jotka liikkuvat sattumanvaraisiin suuntiin maastossa.



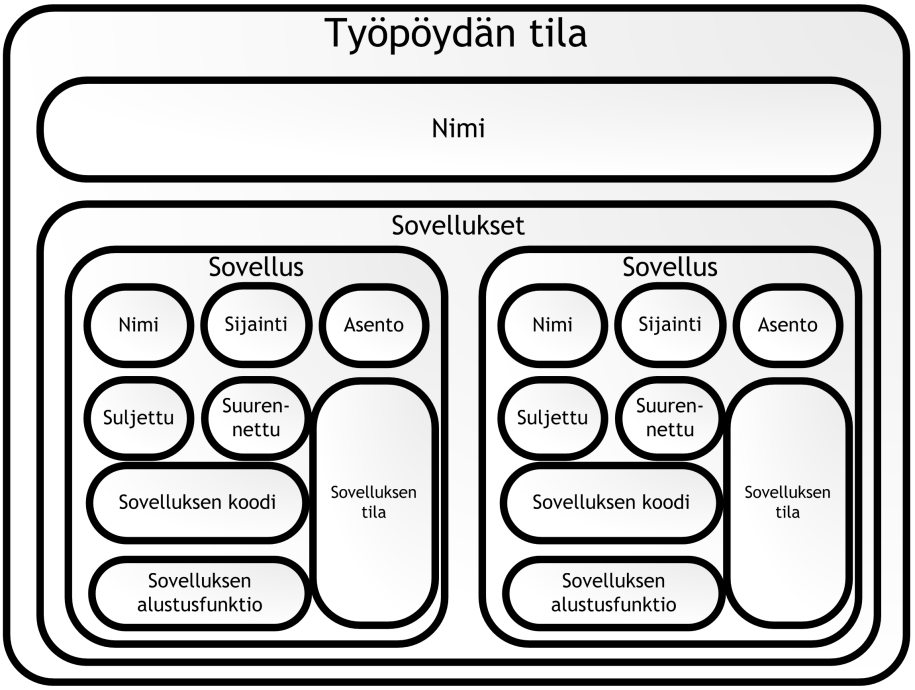
Kuva 4.12: Rajapintoja käyttäen toteutettu 3D-ympäristö, jossa näkyvissä yksi sovellus.

Jokaisessa 3D-ympäristössä on määritetty kolme funktiota: hiiritapahtumien kuuntelijoiden uudelleen asettaminen, objektin lisääminen ympäristöön ja ympäristön päivitys animointisilmukan aikana. Hiiren tapahtumien kuuntelijat asetetaan uudestaan oletuksiinsa jokaisella ympäristön vaihdolla, joten ympäristön kehittäjän ei tarvitse asettaa niitä uudestaan jos oletuskuuntelijat ovat riittävät. Objektin lisäämisfunktiota kutsutaan sovelluksen integroinnin ja uuden ympäristön tuonnin aikana, jos Lively3D:ssä on ladattuja sovelluksia. Funktio luo GLGE-objekteja, joko hakemalla ne näkymän XML-tiedostosta tai luomalla ne dynaamisesti käyttäen GLGE:n JavaScript rajapintakutsuja. Luonnin jälkeen ne lisätään nykyiseen näkymään ja palautetaan Lively3D:lle, joka tallentaa ne sisäisesti käytössä oleviin sovellusobjekteihin. Näkymän päivitysfunktiota kutsutaan animointisilmukan aikana. Se mahdollistaa ympäristön muokkaamisen ruudun päivitysten välillä. Funktiolle annetaan aika-arvot, joita voidaan käyttää laskemaan kuinka paljon aikaa on kulunut sitten edellisen ruudunpäivityksen. Funktiota voidaan käyttää käsittelemään näppäinkomentoja, luomaan uusia objekteja, siirtämään olemassa olevia objekteja ja muihin samankaltaisiin toimintoihin, jotka muodostavat animaation kun ruutua päivitetään. Funktiota kutsutaan vain, jos kyseinen ympäristö on esitettynä, joten se ei hidasta muiden ympäristöjen renderöimistä.

4.4. Persistointi

Persistointia käytetään Lively3D:ssä tallentamaan 3D-ympäristön ja sen sovelluksien tila, jotta niitä voidaan jatkaa myöhemmin. Tämä saavutetaan luomalla JSON-objekti, joka

tallennetaan joko MongoDB-tietokantaan tai tekstitiedostoon. Tallennuspyynnössä Lively3D luo JSON-objektin, joka koostuu ympäristön tilan nimestä ja sovellustaulukosta, johon tallennetaan kaikki tallennushetkellä ladatut ohjelmat. Tämä objekti on esitetty kuvassa 4.13.



Kuva 4.13: Lively3D:n tilaobjekti.

Jokaisesta sovelluksesta tallennetaan nimi, sovelluksen nykyinen sijainti ja asento oletusympäristössä, totuusarvo sovelluksen aukaisu- tai maksimointitilasta, sovelluksen koodi kokonaisuudessaan sekä alustusfunktio sovelluksen palautusta varten, joiden lisäksi tallennetaan objekti sovelluksen sisäisestä tilasta. Sisäistä tilaa varten sovelluksen kehittäjän pitää toteuttaa funktiot, jotka luovat ja jäsentävät vapaavalintaisen JavaScript objektin. Tämä objekti tallennetaan tietokantaan tai tiedostoon sovelluksen kanssa ja kehittäjän vastuulla on, kuinka tätä objektia käytetään tilan palautuksessa. Jos näitä funktioita ei ole toteutettu, Lively3D käyttää tyhjiä toteutuksia, jolloin sisäistä tilaa ei tallenneta.

Luotu JSON-objekti lähetetään joko PHP-välityspalvelimelle, joka tallentaa sen Dropboxiin tai Node.js-sovellukselle, joka hoitaa interaktion tietokannan kanssa. Node.js-sovellus koostuu kahdesta osasta. Ensimmäinen osa on HTTP-välityspalvelin, joka päättää minne pyynnöt välitetään. Node.js-spesifiset verkko-osoitteet välitetään samalle Node.js-sovellukselle ja kaikki muut Apache-verkkopalvelimelle, joka tarjoaa Lively3D:n muut osat. Node.js-sovelluksen toinen osa on tietokantainteraktio, joka jäsentää pyynnöt tietokantakyselyiksi. MongoDB valittiin tietokannaksi, koska se mahdollistaa tuntemattomien JSON-objektien tallennuksen. Tämä oli elintärkeää kehittäjän määrittämille JavaScript-objekteille, joita käytetään sovelluksien sisäisen tilan esittämiseen. Node.js-sovellus tarjoaa pyynnönkäsittelijät tilan tallennukselle, joka tallentaa tilan luomalla uu-

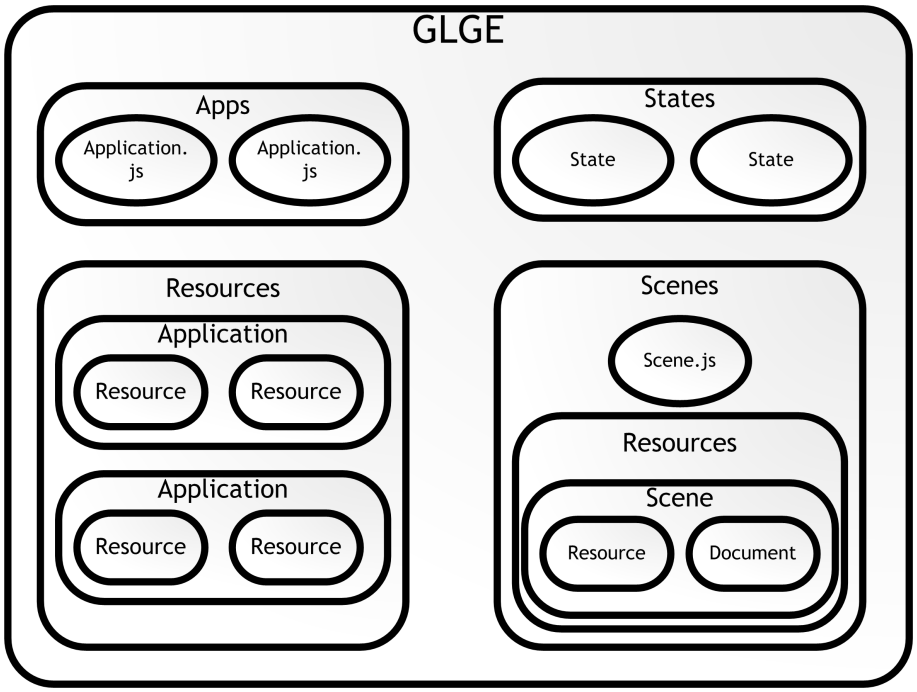
den arvon kantaan tai korvaamalla vanhan nimen perusteella. Lisäksi sovellus tarjoaa käsittelijät tilojen nimien listan hakuun sekä tilan JSON-objektin hakuun nimen perusteella.

Kun aikaisemmin tallennettu tila ladataan Lively3D:hen, nykyiset ladatut sovellukset poistetaan, ja tilaobjektin sovellukset luodaan käyttäen samoja funktioita kuin yksitellen ladatessa käytetään. Tämä mahdollistaa ympäristön palautuksen samaan tilaan kuin missä se oli tallennuksen yhteydessä, jopa eri selaimessa tai tietokoneessa.

4.5. Asennusympäristö

Lively3D:n sovelluksien, ympäristöjen ja näiden resurssien asennusympäristöksi valittiin Dropbox, koska Lively3D:n sovellukset ja ympäristöt on toteutettavissa kolmannen osapuolen kehittäjillä ja Lively3D:n verkkopalvelimesta erillinen asennusympäristö tarjoaa mahdollisuuden ottaa uusia sovelluksia ja ympäristöjä käyttöön ilman, että tiedostoja tarvitsee erikseen siirtää Lively3D:n verkkopalvelimelle.

Kommunikaatio Dropboxin ja Lively3D:n välillä hoidetaan joko PHP-kirjastolla tai Node.js-sovelluksella, jotka abstrahoivat Dropbox rajapinnan yksinkertaisiksi funktiokutsuiksi. PHP-kirjastoa ja Node.js-sovellusta käytetään palvelimella olevassa välityspalvelinsovelluksessa, joka tarjoaa käsittelijät tiedostolistan ja yksittäisen tiedoston hakemiseen. Välityspalvelin perustuu tiukkaan hakemistorakenteeseen, joka mahdollistaa Dropboxin käytön sovelluksien asennusympäristönä. Tämä hakemistorakenne on esitetty kuvassa 4.14.



Kuva 4.14: Lively3D:n hakemistorakenne Dropboxissa.

Lively3D havaitsee uudet sovellukset hakemalla apps-hakemiston tiedostolistauksen,

josta sovellus ladataan pyytämällä yksittäinen tiedosto. Tämän pyynnön perusteella tiedosto haetaan palvelimelle ja palvelin vastaa pyyntöön tarjoamalla URL:n haettuun tiedostoon. Sama pätee sovelluksen resursseihin: sovellus pyytää taulukollisen tiedostonimiä ja välityspalvelin vastaa taulukollisella URL:ja, joiden käyttötapa jää sovelluksen kehittäjän vastuulle. Koska välityspalvelin tarjoaa vain URL:ja Lively3D:n käyttöön, kaistan ja muistin tarve vähenee, sillä selain käsittelee URL:t sen sijaan, että kaikki resurssit ladattaisiin muistiin ennen resurssin käyttöä. Välityspalvelin on myös tietoinen tiedostojen muutoksista Dropbox rajapinnan välityksellä, jolloin tiedostoa ei haeta Dropboxista uudelleen, jos palvelimella on jo sama tiedosto aikaleiman mukaan. Koska tiedoston siirtopyynnöt pysäytetään välityspalvelimelle, tämä nopeuttaa Dropbox-vuorovaikutusta.

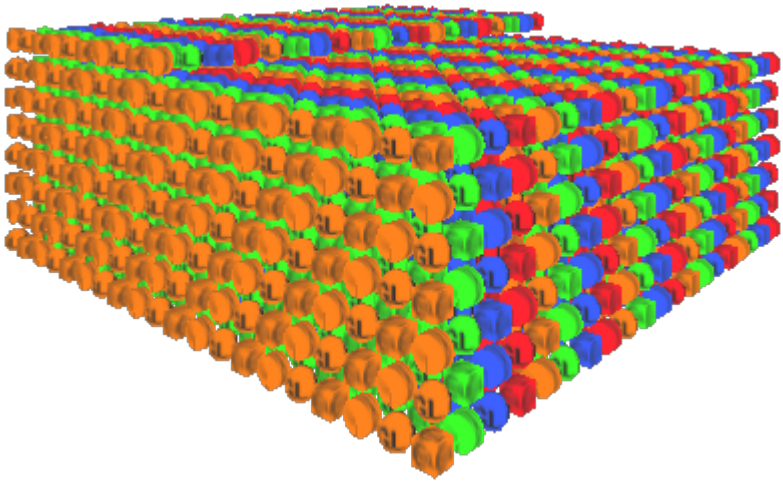
5. ARVIOINTI

Lively3D:tä toteuttaessa havaittiin teknologioiden, kirjastojen ja selaimen tietoturvan asettamia rajoituksia. Näitä rajoituksia ovat mm. renderöintinopeus, ominaisuudet, joita rajapinnat eivät tarjoa, JavaScriptin Same Origin Policy sekä eri selaimien väliset toteutuserot.

5.1. Renderöintinopeus

Lively3D:tä toteuttaessa havainnoitiin 3D-objektien lukumäärän ja käytetyn selaimen vaikuttavan näkymän renderöintinopeuteen. Tätä vaikutusta tutkittiin toteuttamalla yksinkertaisia ohjelmia eri WebGL-kirjastoilla ja mittaamalla tuloksia käyttäen Stats.js[1]- sekä WebGL Inspector[2] -työkaluja .

3D-objektien lukumäärän vaikutusta arvioitiin toteuttamalla puhtaalla WebGL:llä, GLGE:llä sekä Three.js:llä näkymä, jossa kolmea mallia ja neljää tekstuuria vuorotellen luotiin objekteja haluttu määrä. Kuvassa 5.1 on esitetty Three.js toteutuksen näkymä 3000 objektilla.

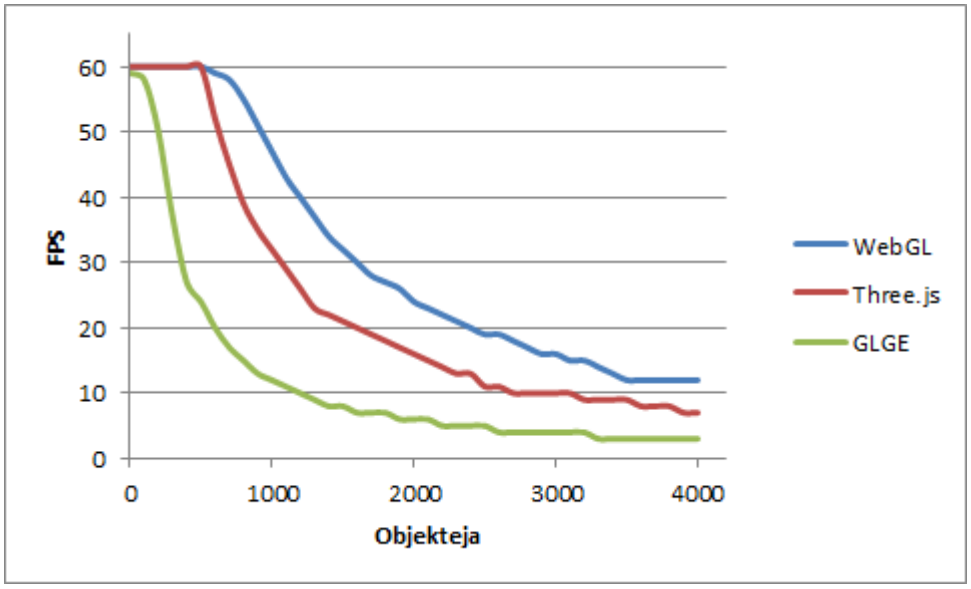


Kuva 5.1: Näkymä, jolla mitattiin objektien lukumäärän vaikutusta renderöintinopeuteen.

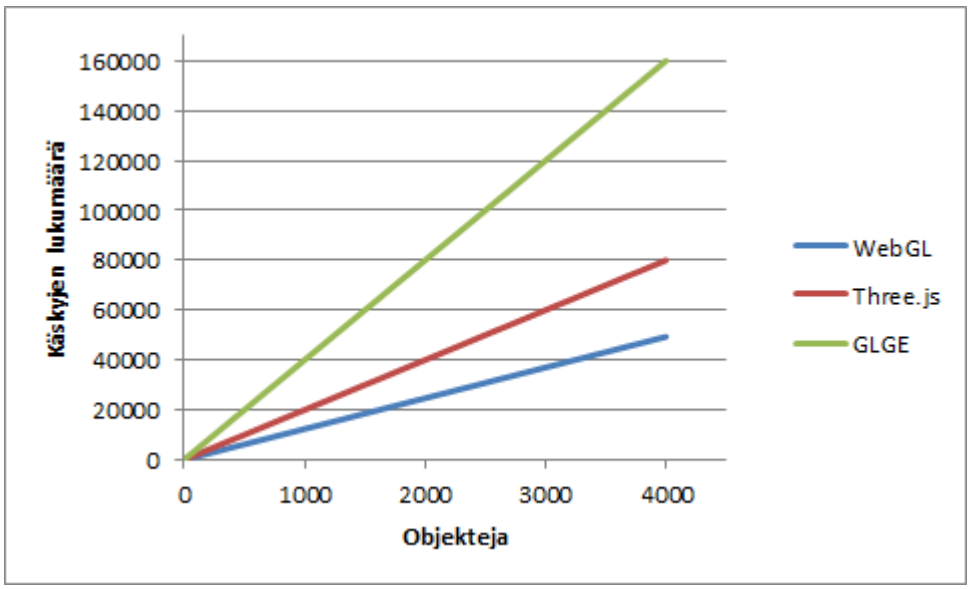
Mittaukset suoritettiin koneella, jossa on Intel Core i5 2.53 GHz:n prosessori, Nvidia

[1] https://github.com/mrdoob/stats.js
[2] http://benvanik.github.com/WebGL-Inspector/

Quadro NVS 3100M -näytönohjain, 4 gigatavua keskusmuistia, käyttöjärjestelmänä Windows 7 ja selaimena Google Chrome 14.0.835. Mittauksien tulokset on esitetty kuvien 5.2 ja 5.3 graafeissa. Tuloksista nähdään, että kirjastojen käyttö lisää käskyjä näytönohjaimelle yksinkertaisissakin näkymissä, mikä hidastaa renderöintiä.



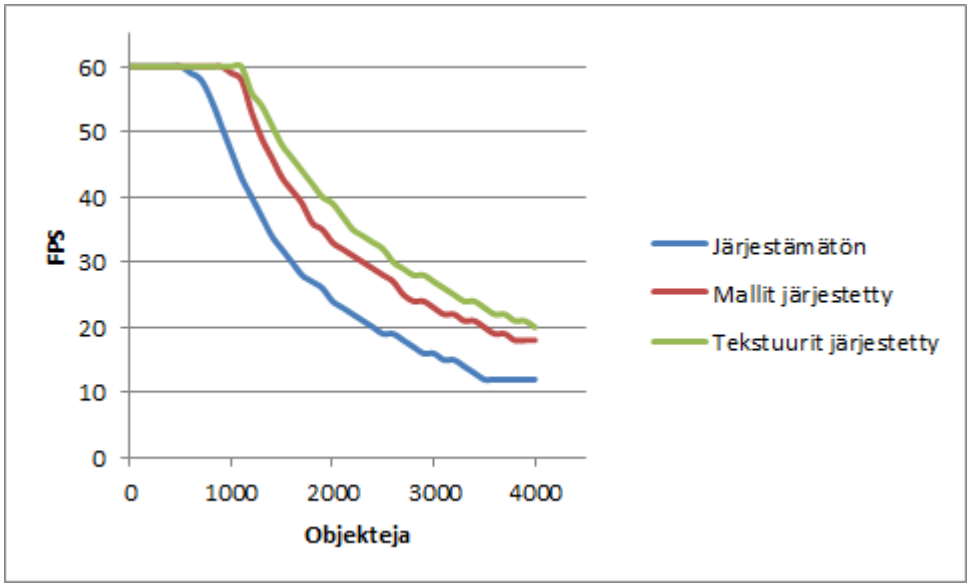
Kuva 5.2: Objektien lukumäärän vaikutus renderöintinopeuteen.



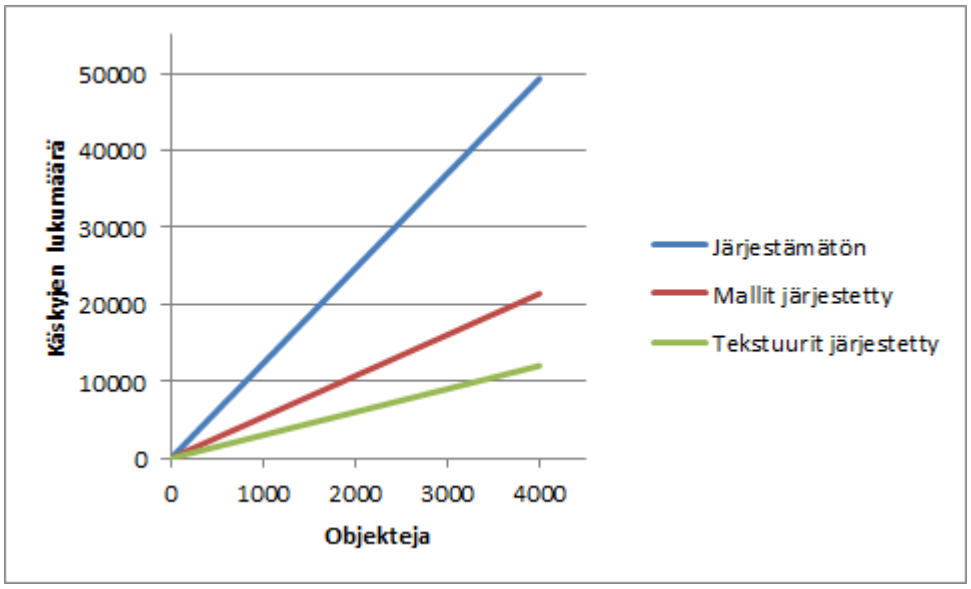
Kuva 5.3: Objektien lukumäärän vaikutus näytönohjaimelle menevien käskyjen lukumäärään.

Tuloksista myös nähdään, että mitä korkeammalla abstraktiotasolla kirjasto on, sitä vähemmän ohjelmoijalla on mahdollisuuksia vaikuttaa näkymän muodostamiseen. GLGE:n tapa aiheuttaa suuren määrän käskyjä näytönohjaimelle, kun taas Three.js:n optionaaliset valaistus- ja teksturointimallit ovat huomattavasti tehokkaampia.

Näkymän muodostusta voidaan optimoida mm. järjestämällä mallit ja tekstuurit, jolloin käskyjä näytönohjaimelle voidaan vähentää, kun ei tarvitse vaihtaa mallia tai tekstuuria jokaisen objektin kohdalla. Järjestämisen jälkeen näkymä renderöidään siten, että ensin renderöidään kaikki siniset kuutiot, joiden jälkeen renderöidään vihreät, oranssit ja punaiset kuutiot. Kuutioiden jälkeen sama tekstuurijärjestys toteutetaan pyramideille ja palloille. Näiden optimointien vaikutukset puhtaassa WebGL-toteutuksessa on esitetty kuvien 5.4 ja 5.5 graafeissa.



Kuva 5.4: Mallien ja tekstuurien järjestämisen vaikutus renderöintinopeuteen.

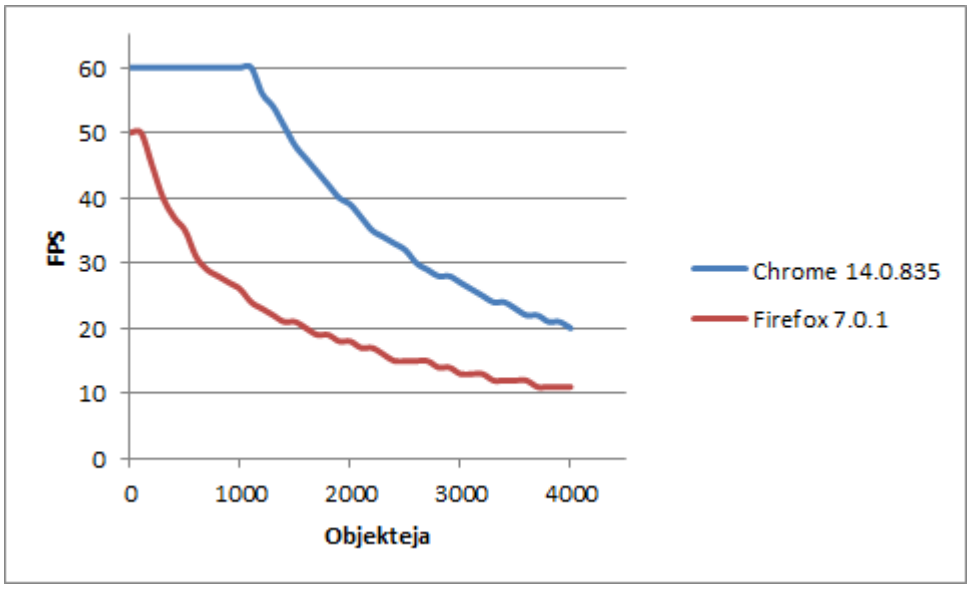


Kuva 5.5: Mallien ja tekstuurien järjestämisen vaikutus näytönohjaimelle menevien käskyjen määrään.

Edellä mainitun optimoinnin hyödyt rajoittuvat näkymiin, joissa käytetään paljon samoja malleja ja tekstuureita. Yleiskäyttöisempää optimointia on toteutettavissa esimerkiksi yhdistämällä kaikki näkymän objektit yhdeksi isoksi malliksi, joka voidaan piirtää yhdellä käskyllä. Lisäksi renderöintisilmukassa kannattaa välttää raskaita näytönohjainkäskyjä, kuten ohjelman tilan noutaminen näytönohjaimelta tai renderöintipuskurien asettaminen.

5.2. Selainten WebGL-toteutuksien erot

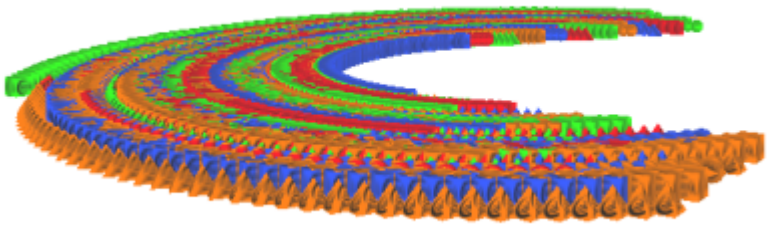
Lively3D:tä toteuttaessa selainten WebGL-toteutuksia testattiin Google Chromen versiosta 9 ja Mozilla Firefoxin versiosta 4 alkaen. Versionumeroiden edetessä WebGL-renderöinti on nopeutunut ja havaittavia virheellisiä toiminnallisuuksia on esiintynyt ja niitä myös on korjattu uudemmissa versioissa. Edellä ollutta optimoitua WebGL-näkymää testattiin Google Chromen versiolla 14.0.835.202, ja Mozilla Firefoxin versiolla 7.0.1. Renderöinnin mittaustulokset on esitetty kuvan 5.6 graafissa. Tuloksista nähdään, että tällä hetkellä Chromen WebGL-toteutus on Firefoxin toteutusta huomattavasti nopeampi.



Kuva 5.6: Optimoidun WebGL-näkymän renderöintinopeus Google Chromella ja Mozilla Firefoxilla.

Mittauksia tehtäessä havaittiin myös, että Chromen ja Firefoxin välillä on selvästi havaittavia toteutuseroja. Jos WebGL-kontekstia ei tyhjennetä ennen renderöintiä, se tuottaa selaimissa erilaisen lopputuloksen. Firefoxissa uusi näkymä piirretään vanhan päälle, jonka lopputulos on esitetty kuvassa 5.7, Chromessa lopputulos on edelleen kuvan 5.1 mukainen riippumatta siitä, että tyhjennetäänkö kontekstia vai ei.

Tämä hankaloittaa sovellusten kehittämistä, joissa nimenomaan halutaan piirtää vanhan kuvan päälle esimerkiksi tietynlaisia efektejä luodessa. Chromea varten joudutaan



Kuva 5.7: Kontekstin tyhjentämättömyyden vaikutus Firefox-selaimessa.

toteuttamaan rakenteet, jotka säilövät kontekstin tilaa. Säilötty tila voidaan yhdistää sovellustasolla uuteen renderöintiin, joka sitten piirretään kontekstiin kokonaan uudestaan.

5.3. Havaitut ongelmat

Lively3D:n toteutuksen aikana havaittiin ongelmia ja rajoitteita sekä testattujen selaimien tietoturvassa, että niiden WebGL-toteutuksissa. Tietoturvassa rajoitteita asettaa Same Origin Policy, joka WebGL-kontekstissa pätee myös näkymän käyttämiin resursseihin. Teksturoidessa objekteja käytetyt tekstuurit ovat Same Origin Policyn alaisia, joten ne on rajoitettu samalle koneelle kuin verkkopalvelu. Tätä rajoitusta ratkaisemaan on kehitetty Cross-Origin Resource Sharing (CORS), jossa määritetään kuinka verkkopalvelimen pitää tarjota resurssinsa, jotta niitä voidaan käyttää toisesta verkkolähteestä [37]. CORS:ia määrittäessä voidaan rajata pääsy resursseihin vain tietyille verkkopalveluille tai kenen tahansa käytettäviksi. CORS-tuki WebGL-kontekstissa on toteutettuna Google Chromen versiosta 13 ja Mozilla Firefoxin versiosta 8 alkaen.

Vaikka selaimen tietoturvaominaisuudet huomioitiin Lively3D:tä toteuttaessa, itse Lively3D:n ominaisuudet ovat hyvin tietoturvattomia. Kolmannen osapuolen toteuttamaa koodia ajetaan sovelluksissa ja 3D-ympäristöissä ilman minkäänlaisia validointeja. Ulkopuolista koodia ei pitäisi koskaan suorittaa, jos millään tavalla halutaan käyttäjää suojata [38]. Lively3D:n tilan tallennuksen avulla myös saadaan levitettyä selaimen työkaluilla muokatut haitalliset sovelluskoodit muille käyttäjille, vaikka muokkaajalla ei olisi pääsyä verkkopalvelimelle tai Dropboxiin.

WebGL-toteutuksien ongelmia havaittiin lähinnä Google Chromesta, sillä Lively3D:n pääasiallinen testaus tehtiin Chromella ja Firefoxia testattiin vain, jos Chrome toimi virheellisesti. Canvas-elementtien käyttäminen tekstuureina oli viikkojen ajan hankalaa, sillä Chromessa esiintyi tekstuurien vaihtumista objektien välillä, jos tekstuureina käytettiin kahta tai useampaa suurin piirtein samankokoista canvas-elementtiä. Tämä virheellinen toiminta on sittemmin korjattu. Chromen versiosta 13 alkaen GLGE-kirjaston funktioilla toteutettu hiiren liikkeen seuranta 3D-näkymässä aiheuttaa vilkkumista näkymässä, jo-

ka johtuu näkymän tyhjentämisen ja piirtämisen välisen ajan pitenemisestä. Firefoxissa vastaavaa toiminnallisuutta ei ole.

WebGL-spesifikaatio myös rajoittaa käytettävää sisältöä. Spesifikaatio sallii vain kuvat, videot ja canvas-elementit käytettäväksi tekstuureina. Lively3D:n sovelluksia toteuttaessa valmiin verkkosisällön hyödyntäminen jää melko rajatuksi, koska tämä sisältö pitää pohjautua canvas-elementtiin ja Lively3D:n rajapinta pitää toteuttaa. Koska WebGL-spesifikaatio ei salli IFrame-elementtien käyttämistä, niin olemassa olevia verkkosivuja ei voida sisällyttää näkymään tekstuureina. Jos haluttaisiin toteuttaa Lively3D-sovellus, joka toimisi kuten selain, pitäisi toteuttaa canvas-elementille HTML-jäsennin, CSS-jäsennin, noutaa verkkosivut Lively3D:n käyttämälle palvelimelle Same Origin Policyn kiertämiseksi ja renderöidä verkkosivut canvas-elementtiin.

WebGL-kirjastojen puutteellinen tai olematon dokumentaatio koettiin suurimpana ongelmana WebGL-sovelluksia toteuttaessa. Vaikka kirjastot väittävät yksinkertaistavan WebGL-sovelluksien kehitystä, niin kehittäjä joutuu kuluttamaan paljon aikaa kirjastoon tutustumiseen lähdekooditasolla. Useimpien kirjastojen dokumentaationa toimivat esimerkkisovellukset, jotka eivät aina kata kaikkea mihin kirjasto pystyy. Riskinä dokumentoimattoman rajapinnan käytössä on, että rajapinta muuttuu ilmoittamatta ja kehittäjän WebGL-sovelluksen toiminnallisuus hajoaa, jos kirjaston päivittää uudempaan.

6. JOHTOPÄÄTÖKSET

Tulevat HTML5, sen oheisteknologiat ja WebGL mahdollistavat alustariippumattomat modernit rikkaat verkkosovellukset ilman lisäosien asentamista selaimeen. Kokonaisuutena HTML5 ja sen oheisteknologiat muodostavat yhtenäisen kokonaisuuden, jonka käyttö on yksinkertaista. Se myös tarjoaa paljon uusia mahdollisuuksia verkkosovelluksiin, jotka aikaisemmin vaativat binääritoteutuksen, jossa ei ollut yhtenäistä tapaa mm. mediatoiston, paikannuksen ja tietokanta-abstraktion toteuttamiseen.

WebGL tarjoaa standardoidun tavan esittää 3D-sisältöä selaimessa ja selainten WebGL-tuki paranee koko ajan sekä työpöytä- että mobiilikäytössä. WebGL itsessään on alhaisen tason spesifikaatio, jonka päälle on rakennettu useita kirjastoja abstrahoimaan yksityiskohtia. Näillä kirjastoilla on kullakin oma käyttökohteensa ja vaatimukset kehittäjän 3D-ohjelmoinnin tuntemukselle. Kirjastojen huonona puolena on niiden keskeneräisyys, varsinkin dokumentaation osalta. Ne eivät ole vielä niin kehittyneitä, että kuka tahansa niitä voisi käyttää ilman, että tutustuisi kattavasti esimerkkiohjelmiin, eikä kirjastot myöskään ole niin samankaltaisia, että sovelluksen kehittäjä voisi helposti vaihtaa kirjastosta toiseen.

WebGL asettaa rajoituksia renderöintinopeuteen ja yleisesti haluttuihin ominaisuuksiin, joita rajapinnat eivät tarjoa. Renderöintinopeutta voi parantaa koodin optimoinnilla 3D-ohjelmoinnista tutuilla tekniikoilla, joilla vältetään näytönohjaimen turhaa käskyttämistä. Lisäksi nopeutta voi parantaa sovellusalueeseen sopivilla kirjastovalinnoilla. Valmiin verkkosisällön sisällyttäminen ei onnistu WebGL-kontekstiin, sillä konteksti nykyisessä standardissa hyväksyy vain canvas-, kuva- ja video-elementtejä. Tämä rajoittaa Mashup-sovelluksien kehittämistä, koska olemassa olevia verkkosivuja on suoraan voi kontekstiin lisätä. Selaimien väliset toteutuserot aiheuttavat ongelmia, jotka WebGL-sovelluskehittäjän on huomioitava. JavaScriptin tietoturva rajoittaa mitä verkon sisältöä WebGL-sovellus voi hyödyntää. Näihin rajoituksiin voi varautua käyttämällä CORS-tekniikkaa tai välityspalvelimia osana WebGL-sovellusta.

Jatkossa WebGL-toteutukset ja JavaScript-moottorit nopeutuvat entisestään ja verkkosovelluksien kehitys johtaa entistä enemmän Mashup-sovelluksia kohti. Valmiit komponentit kehitetään julkisten rajapintojen taakse, jolloin niiden käyttö on helppoa ja se mahdollistaa ohjelmistokehittäjien yhteistyön ilman että kehittäjien tarvitsee olla toisiinsa yhteydessä. Tämä sallii sovelluksien, datan, visualisoinnin tai minkä tahansa muun resurssin jakamisen ja uudelleenkäytön missä tahansa verkkosovelluksessa. Sovellukset olivat to-

dellisia verkkosovelluksia, jotka koostuvat dynaamisesti ladatuista komponenteista, jotka soveltuvat kyseisen sovelluksen tarpeisiin. Tällöin ohjelmistokehityksen tuottavuus voisi dramaattisesti kasvaa, sillä komponenttien uudelleen käytöllä sovelluksen kehittäjän tarvitsee vain yhdistellä resurssit.

Lively3D on sovelluskehys, joka mahdollistaa olemassa olevien canvas-sovelluksien integroimisen yhteen kolmiulotteiseen ikkunointiympäristöön. Rajapintaa noudattamalla suuria muutoksia sovelluksiin ei tarvitse tehdä ja Lively3D:n resurssien lataaja mahdollistaa monipuolisten resurssien käyttämisen. Lively3D:ssä on myös mahdollista toteuttaa uusia esitystapoja 3D-ympäristölle, joissa kaikki ympäristön toiminnot ovat kustomoitavissa.

Mashup-sovellukset WebGL-kontekstissa mahdollistavat kolmiulotteisen esityksen sisällölle, esimerkiksi Google Maps[1] ja Nokia Maps[2] -karttasovelluksissa on mallinnettu rakennukset kolmiulotteisina, jotka voi tulevaisuudessa sisällyttää omiin Mashup-sovelluksiin. CORS-tekniikan yleistyessä kolmannen osapuolen sisällön hyödyntäminen on vaivattomampaa, joka mahdollistaa entistä monipuolisemmat Mashup-sovellukset.

[1] `http://maps.google.fi/`
[2] `http://maps3d.svc.nokia.com/webgl/`

LÄHTEET

[1] Antero Taivalsaari. Mashware: the Future of Web Applications. Technical report, Mountain View, CA, USA, 2009.

[2] Stat Owl. Flash Player Version Support. `http://statowl.com/flash.php`.

[3] D. Crane, D.C.E. Pascarello, E. Pascarello, and D. James. *Ajax in action*. A1bazaar, 2005.

[4] Charles Freedman, Keith Peters, Clint Modien, Ben Lucyk, and Ryan Manning. *Professional AIR: Application Development for the Adobe Integrated Runtime*. Wrox Press Ltd., Birmingham, UK, UK, 2009.

[5] R. Hanson and A. Tacy. *GWT in Action*. Manning, 2007.

[6] J. Weaver and J.L. Weaver. *JavaFX Script: Dynamic Java Scripting for Rich Internet/Client-Side Applications*. Apress, 2007.

[7] L. Moroney. *Introducing Microsoft Silverlight 2.0*. Microsoft Press, 2008.

[8] B. Tate and C. Hibbs. *Ruby on Rails: Up and Running*. O'Reilly Media, Inc., 2006.

[9] Antero Taivalsaari, Tommi Mikkonen, Dan Ingalls, and Krzysztof Palacz. Web Browser as an Application Platform: The Lively Kernel Experience. Technical report, Mountain View, CA, USA, 2008.

[10] Tommi Mikkonen and Antero Taivalsaari. Web Applications - Spaghetti Code for the 21st Century. In *Proceedings of the 2008 Sixth International Conference on Software Engineering Research, Management and Applications*, pages 319–328, Washington, DC, USA, 2008. IEEE Computer Society.

[11] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. Technical report, 1999.

[12] Dave Raggett, Arnaud Le Hors, and Ian Jacobs. HTML 4.01 Specification. Technical report, W3C, 1999. `http://www.w3.org/TR/html401/`.

[13] Bert Bos, Tantek Çelik, Ian Hickson, and Håkon Wium Lie. Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. Technical report, W3C, 2011. `http://www.w3.org/TR/CSS2/`.

[14] ECMA-262 ECMAScript Language Specification, 2011. `http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf`.

[15] Adele Goldberg and David Robson. *Smalltalk-80: the language and its implementation*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1983.

[16] John McCarthy. *LISP 1.5 Programmer's Manual*. The MIT Press, 1962.

[17] David Ungar and Randall B. Smith. Self: The power of simplicity. In *Conference proceedings on Object-oriented programming systems, languages and applications*, OOPSLA '87, pages 227–242, New York, NY, USA, 1987. ACM.

[18] Arnaud Le Hors, Philippe Le Hégaret, Lauren Wood, Gavin Nicol, Jonathan Robie, Mike Champion, and Steve Byrne. Document Object Model (DOM) Level 3 Core Specification. Technical report, W3C, 2004. `http://www.w3.org/TR/DOM-Level-3-Core/`.

[19] Douglas Crockford. The application/json Media Type for JavaScript Object Notation (JSON). Technical report, IETF, 2006.

[20] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau, and John Cowan. Extensible Markup Language (XML) 1.1 (Second Edition). Technical report, W3C, 2006. `http://www.w3.org/TR/xml11/`.

[21] Jesse James Garrett. Ajax: A New Approach to Web Applications, 2005. `http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications`.

[22] Ian Hickson. HTML5, A vocabulary and associated APIs for HTML and XHTML. Technical report, W3C, WHATWG, 2011. `http://www.w3.org/TR/html5/spec.html`.

[23] Ian Hickson. Web Workers. Technical report, W3C, 2011. `http://www.w3.org/TR/workers/`.

[24] Andrei Popescu. Geolocation API Specification. Technical report, W3C, 2010. `http://www.w3.org/TR/geolocation-API/`.

[25] Andrei Popescu, Jeremy Orlow, Eliot Graff, Jonas Sicking, and Nikunj Mehta. Indexed Database API. Technical report, W3C, 2011. `http://www.w3.org/TR/IndexedDB/`.

[26] Ian Hickson. The WebSocket API. Technical report, W3C, 2011. `http://www.w3.org/TR/websockets/`.

[27] Eric Uhrhane. File API: Directories and System. Technical report, W3C, 2011. `http://www.w3.org/TR/file-system-api/`.

[28] Arun Ranganathan and Jonas Sicking. File API. Technical report, W3C, 2010. http://www.w3.org/TR/FileAPI/.

[29] Chris Marrin. WebGL Specification. Technical report, Khronos Group, 2011. http://www.khronos.org/registry/webgl/specs/1.0/.

[30] Aaftab Munshi and Jo Leech. OpenGL ES 2.0 Specification. Technical report, Khronos Group, 2010. http://www.khronos.org/registry/gles/specs/2.0/es_full_spec_2.0.25.pdf.

[31] Robert J. Simpson. OpenGL ES Shading Language. Technical report, Khronos Group, 2009. http://www.khronos.org/registry/gles/specs/2.0/GLSL_ES_Specification_1.0.17.pdf.

[32] Antti Puhakka. *3D-Grafiikka*. Talentum Media, 2008.

[33] Niels Leenheer. The HTML5 test. http://www.html5test.com/.

[34] Mark Pilgrim. Dive Into HTML5. http://diveintohtml5.org/.

[35] Paul Brunt. GLGE. http://www.glge.org/.

[36] Matti Anttonen and Arto Salminen. Building 3D WebGL Applications. Technical report, Tampereen Teknillinen Yliopisto, 2011.

[37] Anne van Kesteren. Cross-Origin Resource Sharing. Technical report, W3C, 2010. http://www.w3.org/TR/cors/.

[38] Jari-Pekka Voutilainen. Tietoturva Ajaxia hyödyntävissä verkkosovelluksissa. Kandidaatintyö, Tampereen Teknillinen Yliopisto, 2010.

A. WEBGL ESIMERKKI

Ohjelma A.1: WebGL esimerkin HTML.

```
<!DOCTYPE html>
<html>
  <head>
  <title>WebGL Example</title>
  <meta http-equiv="content-type" content="text/html; charset=ISO-8859-1">
  <script type="text/javascript" src="glMatrix-0.9.5.min.js"></script>
  <script type="text/javascript" src="webgl-utils.js"></script>
  <script id="shader-fs" type="x-shader/x-fragment">
    #ifdef GL_ES
    precision highp float;
    #endif

    varying vec4 vColor;

    void main(void) {
      gl_FragColor = vColor;
    }
  </script>

  <script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    attribute vec4 aVertexColor;

    uniform mat4 uMVMatrix;
    uniform mat4 uPMatrix;

    varying vec4 vColor;

    void main(void) {
      gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
      vColor = aVertexColor;
    }
  </script>

  <script src="webgl.js"></script>
  </head>
  <body onload="webGLStart();">
    <canvas id="example" style="border: none;" width="300" height="300"></canvas>
  </body>
</html>
```

Ohjelma A.2: WebGL esimerkin JavaScript.

```
var gl;
function initGL(canvas) {
  try {
    gl = canvas.getContext("experimental-webgl");
    gl.viewportWidth = canvas.width;
    gl.viewportHeight = canvas.height;
```

```
7     } catch (e) {
8     }
9     if (!gl) {
10      alert("Could not initialise WebGL, sorry :-(");
11    }
12  }
13
14  function getShader(gl, id) {
15    var shaderScript = document.getElementById(id);
16    if (!shaderScript) {
17      return null;
18    }
19
20    var str = "";
21    var k = shaderScript.firstChild;
22    while (k) {
23      if (k.nodeType == 3) {
24        str += k.textContent;
25      }
26      k = k.nextSibling;
27    }
28
29    var shader;
30    if (shaderScript.type == "x-shader/x-fragment") {
31      shader = gl.createShader(gl.FRAGMENT_SHADER);
32    } else if (shaderScript.type == "x-shader/x-vertex") {
33      shader = gl.createShader(gl.VERTEX_SHADER);
34    } else {
35      return null;
36    }
37
38    gl.shaderSource(shader, str);
39    gl.compileShader(shader);
40
41    if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
42      alert(gl.getShaderInfoLog(shader));
43      return null;
44    }
45
46    return shader;
47  }
48
49  var shaderProgram;
50  function initShaders() {
51    var fragmentShader = getShader(gl, "shader-fs");
52    var vertexShader = getShader(gl, "shader-vs");
53
54    shaderProgram = gl.createProgram();
55    gl.attachShader(shaderProgram, vertexShader);
56    gl.attachShader(shaderProgram, fragmentShader);
57    gl.linkProgram(shaderProgram);
58
59    if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS)) {
60      alert("Could not initialise shaders");
61    }
62
63    gl.useProgram(shaderProgram);
64    shaderProgram.vertexPositionAttribute = gl.getAttribLocation(shaderProgram, "
        aVertexPosition");
```

```
65    gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);
66    shaderProgram.vertexColorAttribute = gl.getAttribLocation(shaderProgram, "aVertexColor
          ");
67    gl.enableVertexAttribArray(shaderProgram.vertexColorAttribute);
68    shaderProgram.pMatrixUniform = gl.getUniformLocation(shaderProgram, "uPMatrix");
69    shaderProgram.mvMatrixUniform = gl.getUniformLocation(shaderProgram, "uMVMatrix");
70  }
71
72  var mvMatrix = mat4.create();
73  var pMatrix = mat4.create();
74  function setMatrixUniforms() {
75    gl.uniformMatrix4fv(shaderProgram.pMatrixUniform, false, pMatrix);
76    gl.uniformMatrix4fv(shaderProgram.mvMatrixUniform, false, mvMatrix);
77  }
78
79  var triangleVertexPositionBuffer;
80  var triangleVertexColorBuffer;
81  function initBuffers() {
82    triangleVertexPositionBuffer = gl.createBuffer();
83    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
84    var vertices = [
85      0.0,  1.0,  0.0,
86      -1.0, -1.0,  0.0,
87      1.0, -1.0,  0.0
88    ];
89    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.STATIC_DRAW);
90    triangleVertexPositionBuffer.itemSize = 3;
91    triangleVertexPositionBuffer.numItems = 3;
92
93    triangleVertexColorBuffer = gl.createBuffer();
94    gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexColorBuffer);
95    var colors = [
96      1.0, 0.0, 0.0, 1.0,
97      0.0, 1.0, 0.0, 1.0,
98      0.0, 0.0, 1.0, 1.0
99    ];
100   gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colors), gl.STATIC_DRAW);
101   triangleVertexColorBuffer.itemSize = 4;
102   triangleVertexColorBuffer.numItems = 3;
103 }
104
105 function drawScene() {
106   gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
107   gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
108
109   mat4.perspective(45, gl.viewportWidth / gl.viewportHeight, 0.1, 100.0, pMatrix);
110   mat4.identity(mvMatrix);
111   mat4.translate(mvMatrix, [0.0, 0.0, -5.0]);
112
113   gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexPositionBuffer);
114   gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
          triangleVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
115   gl.bindBuffer(gl.ARRAY_BUFFER, triangleVertexColorBuffer);
116   gl.vertexAttribPointer(shaderProgram.vertexColorAttribute, triangleVertexColorBuffer.
          itemSize, gl.FLOAT, false, 0, 0);
117   setMatrixUniforms();
118   gl.drawArrays(gl.TRIANGLES, 0, triangleVertexPositionBuffer.numItems);
119 }
120
```

```
121 function tick() {
122   requestAnimFrame(tick);
123   drawScene();
124 }
125
126 function webGLStart() {
127   var canvas = document.getElementById("example");
128   initGL(canvas);
129   initShaders()
130   initBuffers();
131   gl.clearColor(0.0, 0.0, 0.0, 1.0);
132   gl.enable(gl.DEPTH_TEST);
133   tick();
134 }
```

B. LIVELY3D-SOVELLUS

Ohjelma B.1: Lively3D-sovellus.

```
1  // Original game from
2  // http://billmill.org/static/canvastutorial/
3
4  var Arkanoid = function(){
5    var x = 25;
6    var y = 250;
7    var dx = 1.5;
8    var dy = -4;
9    var ce;
10   var ctx;
11   var WIDTH;
12   var HEIGHT;
13   var paddlex;
14   var paddleh = 10;
15   var paddlew = 75;
16   var rightDown = false;
17   var leftDown = false;
18   var canvasMinX = 0;
19   var canvasMaxX = 0;
20   var intervalId = 0;
21   var bricks;
22   var NROWS = 5;
23   var NCOLS = 5;
24   var BRICKWIDTH;
25   var BRICKHEIGHT = 15;
26   var PADDING = 1;
27
28   this.init = function(canvasElement) {
29     if (!canvasElement) {
30       canvasElement = document.createElement("canvas");
31       canvasElement.setAttribute("style", "display: none");
32       canvasElement.id = "arkanoid_canvas";
33     }
34     ce = canvasElement;
35     ce.width = 512;
36     ce.height = 512;
37     ctx = ce.getContext("2d");
38     WIDTH = ce.width;
39     HEIGHT = ce.height;
40     paddlex = WIDTH / 2;
41     BRICKWIDTH = (WIDTH/NCOLS) - 1;
42     canvasMinX = ce.offsetLeft;
43     canvasMaxX = canvasMinX + WIDTH;
44   }
45
46   this.Open = function(){
47     intervalId = setInterval(draw, 10);
```

```
48      return intervalId;
49    }
50
51    function circle(x,y,r) {
52      ctx.beginPath();
53      ctx.arc(x, y, r, 0, Math.PI*2, true);
54      ctx.closePath();
55      ctx.fill();
56    }
57
58    function rect(x,y,w,h) {
59      ctx.beginPath();
60      ctx.rect(x,y,w,h);
61      ctx.closePath();
62      ctx.fill();
63    }
64
65    function clear() {
66      ctx.clearRect(0, 0, WIDTH, HEIGHT);
67      rect(0,0,WIDTH,HEIGHT);
68    }
69
70    this.onKeyDown = function(evt) {
71      if (evt.keyCode == 39) rightDown = true;
72      else if (evt.keyCode == 37) leftDown = true;
73    }
74
75    this.onKeyUp = function(evt) {
76      if (evt.keyCode == 39) rightDown = false;
77      else if (evt.keyCode == 37) leftDown = false;
78    }
79
80    this.initbricks = function() {
81      bricks = new Array(NROWS);
82      for (i=0; i < NROWS; i++) {
83        bricks[i] = new Array(NCOLS);
84        for (j=0; j < NCOLS; j++) {
85          bricks[i][j] = 1;
86        }
87      }
88    }
89
90    function drawbricks() {
91      for (i=0; i < NROWS; i++) {
92        ctx.fillStyle = rowcolors[i];
93        for (j=0; j < NCOLS; j++) {
94          if (bricks[i][j] == 1) {
95            rect((j * (BRICKWIDTH + PADDING)) + PADDING,
96              (i * (BRICKHEIGHT + PADDING)) + PADDING,
97              BRICKWIDTH, BRICKHEIGHT);
98          }
99        }
100     }
101   }
102
103   var ballr = 10;
104   var rowcolors = ["#FF1C0A", "#FFFD0A", "#00A308", "#0008DB", "#EB0093"];
105   var paddlecolor = "#FFFFFF";
106   var ballcolor = "#FFFFFF";
```

```
107   var backcolor = "#000000";
108
109   function draw() {
110     ctx.fillStyle = backcolor;
111     clear();
112     ctx.fillStyle = ballcolor;
113     circle(x, y, ballr);
114
115     if (rightDown) paddlex += 5;
116     else if (leftDown) paddlex -= 5;
117     ctx.fillStyle = paddlecolor;
118     rect(paddlex, HEIGHT-paddleh, paddlew, paddleh);
119
120     drawbricks();
121     rowheight = BRICKHEIGHT + PADDING;
122     colwidth = BRICKWIDTH + PADDING;
123     row = Math.floor(y/rowheight);
124     col = Math.floor(x/colwidth);
125     if (y < NROWS * rowheight && row >= 0 && col >= 0 && bricks[row][col] == 1) {
126       dy = -dy;
127       bricks[row][col] = 0;
128     }
129
130     if (x + dx + ballr > WIDTH || x + dx - ballr < 0)
131       dx = -dx;
132
133     if (y + dy - ballr < 0)
134       dy = -dy;
135     else if (y + dy + ballr > HEIGHT - paddleh) {
136       if (x > paddlex && x < paddlex + paddlew) {
137         dx = 8 * ((x-(paddlex+paddlew/2))/paddlew);
138         dy = -dy;
139       }
140       else if (y + dy + ballr > HEIGHT)
141       clearInterval(intervalId);
142     }
143
144     x += dx;
145     y += dy;
146   }
147
148   this.GetCanvas = function(){
149     return ce;
150   }
151
152   var LivelyApp;
153   this.SetLivelyApp = function(app){
154     LivelyApp = app;
155   }
156
157   this.StartApp = function(){
158     Lively3D.AllowAppStart(LivelyApp);
159   }
160 }
161
162 var ArkanoidInit = function(Arkanoid){
163   var arkanoid = new Arkanoid();
164   arkanoid.init();
165   arkanoid.initbricks();
```

```
166     arkanoid.EventListeners = { "keydown": arkanoid.onKeyDown, "keyup": arkanoid.onKeyUp
            };
167     return arkanoid;
168   }
169
170   var app = Lively3D.AddApplication('Arkanoid', Arkanoid, ArkanoidInit);
171   app.StartApp();
```