Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan koulutusohjelma

WebGL-rajapinta HTML5-pelinkehittäjän näkökulmasta

Kandidaatintyö

14. helmikuuta 2014

Atte Isopuro

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan koulutusohjelma

KANDIDAATINTYÖN
TIIVISTELMÄ

Tekijä:	Atte Isopuro
Työn nimi:	WebGL-rajapinta HTML5-pelinkehittäjän näkökulmasta
Päiväys:	14. helmikuuta 2014
Sivumäärä:	TODO: Kirjoita tähän lopuksi oikea määrä, tässä esimerkissä 23
Pääaine:	Ohjelmistotekniikka
Koodi:	T3001
Vastuuopettaja:	Ma professori Tomi Janhunen
Työn ohjaaja(t):	Professori Jukka Nurminen (Tietoliikenneohjelmistot)
TODO: tiivistelmä	
Avainsanat:	webgl, html5, peli, pelinkehitys, suorituskyky, ongelmia, edut
Kieli:	Suomi

Aalto-universitetet
Högskolan för teknikvetenskaper
Examensprogrammet för datateknik

SAMMANDRAG AV
KANDIDATARBETET

Utfört av:	Atte Isopuro
Arbetets namn:	WebGL-rajapinta HTML5-pelinkehittäjän näkökulmasta
Datum:	14. helmikuuta 2014
Sidoantal:	TODO: Kirjoita tähän lopuksi oikea määrä, tässä esimerkissä 23
Huvudämne:	Ohjelmistotekniikka
Kod:	T3001
Övervakare:	Ma professori Tomi Janhunen
Handledare:	Professori Jukka Nurminen (Tietoliikenneohjelmistot)
Ett abstrakt hit	
Nyckelord:	webgl, html5, peli, pelinkehitys, suorituskyky, ongelmia, edut
Språk:	Svenska

Sisältö

1 Johdanto

Tämä kandidaatintyö käsittelee HTML5:n WebGL-ohjelmointirajapintaa pelinkehittäjän näkökulmasta.

HTML5-spesifikaatio on mahdollistanut kaikenlaisen median esittämisen verkossa ilman, että käyttäjän tarvitsee asentaa erillisiä lisäosia. Yksi näistä uusista rajapinnoista on WebGL, jonka kautta verkkosivun on mahdollista käyttää tietokoneen näytönohjainta teoriassa alustasta riippumatta.

Erityisesti monimutkaista ja tarkkaa interaktiota vaativat pelit tarvitsevat nopeaa grafiikan piirtämistä ruudulle. Jokaiselle alustalle erikseen räätälöidyt ratkaisut ovat kuitenkin kalliita ja hankalia ylläpitää, joten alustasta riippumaton grafiikkaohjelmointi on erittäin kiinnostavaa tästä näkökulmasta.

Kirjallisuudessa on käsitelty WebGL:n käyttömahdollisuuksia erinäisillä aloilla. Useimmat tällaiset käsittelyt ovat kuitenkin yksittäisiä sovelluksia. Yleistä arviota WebGL:n eduista ja haitoista pelinkehityksessä ei ole tehty.

Tämän kandidaatintyön tavoitteena on kerätä pelinkehittäjän kannalta hyödyllisiä havaintoja WebGL:stä. Työssä pyritään tunnistamaan sellaiset WebGL:n ominaispiirteet jotka pelinkehittäjän tulisi ottaa huomioon. Tämä työ ei tule käsittelemään muita selaimen grafiikan esitykseen liittyviä teknologioita, kuten Canvas 2D context tai Flash. Teknologioita verrataan WebGL-rajapintaan, mutta niiden erityispiirteitä ei tutkita tässä työssä tarkemmin.

Työ toteutetaan kirjallisuuskatsauksena. Aineisto koostuu yliopistojen julkaisuista, tieteellisistä artikkeleista ja kirjoista jotka käsittelevät WebGL-rajapintaa. Erityisesti viitataan teksteihin joissa on konkreettisten toteutusten yhteydessä havaittu WebGL:n ominaispiirteitä. Aiheesta ei ole löytynyt suuria määriä kirjallisuutta, joten tieteelliset lähteet on tulkittu hyväksyttäviksi kunhan ne on julkaistu jonkin yliopiston toimesta tai ne löytyvät Scopus-tietokannasta.

Ensiksi alustetaan aiheen taustaa: esitellään HTML5, JavaScript, grafiikan piirtämisen rajapintoja. Seuraavaksi esitellään lyhyesti WebGL-rajapinnalla tehtyjä sovelluksia. Tämän jälkeen eritellään pelinkehittäjän tarpeita ja käydään läpi WebGL-rajapinnasta tehdyt havainnot hänen näkökulmasta. Lopuksi havainnoista tehdään lyhyt yhteenveto, jossa esitetään suositus minkälaisiin projekteihin WebGL-rajapinnan käyttö soveltuu.

2 Taustaa

Takaisin sisällysluetteloon

2.1 Selainteknologiat

Seuraavaksi esitellään tämän työn kannalta keskeisimmät verkkoselaimissa käytetyt teknologiat.

2.1.1 HTML5

HTML5 on uusin versio **H**yper**T**ext **M**arkup **L**anguage-kielestä. Yksi HTML5-kielen tuomia uudistuksia on erinäisten media-formaattien sisäistäminen itse spesifikaatioon erinäisten rajapintojen kautta[1]. Näin ollen sisällöntuottajien ja käyttäjien ei tarvitse huolehtia kolmansien osapuolten liitännäisistä kuten Flash. Tällöin sisällöntuottaja voi olla varma siitä että kaikki ne joilla on spesifikaatiota noudattava selain pääsevät käsiksi hänen tuottamaan sisältöön. Täten alustojen erilaisuuksien ei pitäisi ainakaan teoriassa vaikuttaa sovellukseen, jolloin sisällöntuottaja voi keskittyä luomaan sisältöä ainoastaan yhdelle alustalle.

2.1.2 JavaScript

JavaScript-kieli on laajasti käytössä oleva ohjelmointikieli. Kieli on toteutus ECMAScript-standardista[2] jota käytetään pääasiassa verkkosivujen ohjelmointiin[3]. Kieli mahdollistaa verkkosivujen reaaliaikaisen muuttamisen, animoinnit sekä käyttäjän interaktiot. Useimmat HTML5-kielen määrittämät rajapinnat ovat juuri JavaScript-ohjelmia varten tarkoitettuja.

2.1.3 Canvas-elementti

`<canvas>`-elementti on HTML5-spesifikaation määrittelemä elementti jonka voi sisällyttää verkkosivulleen HTML5-koodissa. Elementin tarkoitus on toimia piirtoalustana: ohjelmoija voi rajapintojen kautta piirtää `<canvas>`-elementille grafiikkaa. Yksi näistä rajapinnoista on tämän työn käsittelemä WebGL-rajapinta.[4]

2.2 Intensiivinen grafiikka ja näytönohjaimen rajapinnat

Intensiivisellä grafiikalla tarkoitetaan tässä yhteydessä grafiikkaa jonka piirtäminen vaatii huomattavan määrän laskentatehoa. Esimerkiksi yksittäisen digitaalisen valokuvan

piirtäminen ruudulle ei ole intensiivistä grafiikkaa. Fysikaalisen ilmiön (kuten veden) simuloiminen ja piirtäminen ruudulle taas on hyvin intensiivistä ja voi veden liikkeistä ja koostumuksesta riippuen vaatia suuria määriä laskentatehoa.

2.2.1 Peligrafiikka

Pelialalla intensiivinen grafiikka yhdistyy useimmiten fysiikan simulointiin, kolmiuloitteisiin ympäristöihin ja monimutkaisiin esineisiin pelimaailmassa. Vaikka tällaisia asioita voidaan periaatteessa simuloida tietokoneen pääsuorittimella, teho jää nopeasti riittämättömäksi. Liian intensiivinen grafiikka hidastaisi konetta sen verran että käyttökokemus kärsii. Graafisesti monimutkaisten pelien laskenta on näin ollen siirrettävä yleispätevästä suorittimesta näytönohjaimeen, joka on graafiseen laskentaan erikoistunut laitekomponentti.

2.2.2 OpenGL-rajapinta

OpenGL (**Open G**raphics **L**ibrary) on alusta- ja kieliriippumaton ohjelmointirajapinta tietokoneiden näytönohjaimille[5]. Ohjelmoija voi käyttää OpenGL-kirjastoa tehdäkseen kutsuja näytönohjaimelle, ilman että hänen tarvitsee käyttää näytönohjaimen omaa, matalan tason käskykantaa. Vastaava teknologia on DirectX[6], Microsoftin ylläpitämä lisensioitava rajapinta-kirjasto, joka on tarkoitettu yksinomaan Windows-käyttöjärjestelmälle.

2.2.3 OpenGL ES-rajapinta

OpenGL ES (**E**mbedded **S**ystems) on OpenGL-kirjaston versio joka on tarkoitettu käytettäväksi sulautetuissa järjestelmissä, kuten pelikonsoleissa, puhelimissa ja tablettitietokoneissa. OpenGL ES on rajatumpi versio täydestä OpenGL-kirjastosta, jotta se olisi mahdollisimman laajasti käytettävä.[7]

2.2.4 WebGL-rajapinta

WebGL (**Web G**raphics **L**ibrary) on JavaScript-rajapinta `<canvas>`-elementtiä varten. WebGL perustuu pitkälti OpenGL ES:ään[8]. Rajapinta sallii näytönohjaimen käskyttämisen JavaScript-koodista. Näin ollen JavaScript-ohjelmoija voi piirtää `<canvas>`-elementille kuvia WebGL-rajapinnan kautta samalla tavoin kuin natiivisovellus tekisi esimerkiksi C++-kieltä ja OpenGL-rajapintaa käyttäen.

3 WebGL-rajapinnan Sovelluksia

Takaisin sisällysluetteloon

WebGL-
rajapinnalla on toteutettu monia erilaisia projekteja. Jimenez tutkimusryhmineen[9] sekä Mobeen ja Feng[10] ovat toteuttaneet kolmiuloitteisen tiedon visualisointimenetelmiä. Kyseiset toteutukset mahdollistavat kolmiuloitteisten mallien tutkimisen selaimessa, erityisesti lääketieteellisissä tutkimuksissa käytettäviä malleja. Näin lääkärit voivat tutkia esimerkiksi magneettikuvantamisessa tuotettuja malleja potilaan aivoista.

WebGL-rajapinnalla on myös tehty tuotteiden esikatseluun tarkoitettuja sovelluksia. Esimerkiksi [11] on autojen esikatselua varten tarkoitettu sovellus. Priyopradono ja Perdana[12] ovat tehneet asuntojen virtuaalista esikatselua varten tarkoitetun sovelluksen. Vaikka kummatkaan sovellukset eivät ole kaupallisessa käytössä ne esimerkillistävät WebGL-rajapinnan mahdollisuuksia.

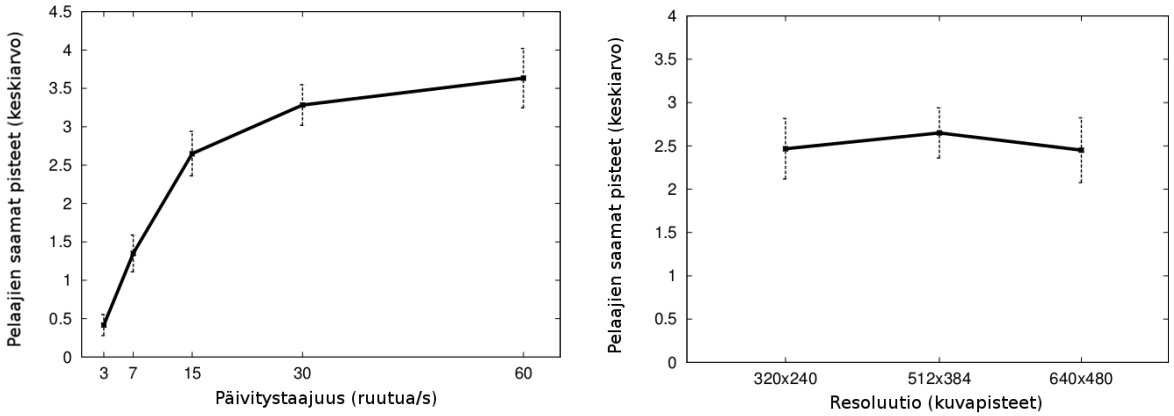
Tämän työn kannalta kiinnostavimmat sovellukset ovat kuitenkin WebGL-rajapinnalla tehdyt pelit, joita on monia[13]. Tunnetuimpia lienee Google:n itse teettämä WebGL-versio Quake 2-pelistä[14]. Pelin iästä huolimatta kyseessä on pelikokemus joka vaatii nopeaa ja tasaista grafiikan piirtoa, joten pelin toteuttaminen ilman näytönohjainta ei luultavasti olisi mahdollista suurimmassa osassa nykyisiä tietokoneita.

Kuitenkaan WebGL-versioiden tekeminen peleistä ei vaikuta olevan suuressa suosiossa. Aiheesta ei löydy kirjallisuutta, mutta WebGL.com-sivuston katsotuimpien projektien listasta[13] voi vetää jonkinlaisia johtopäätöksiä: viiden katsotuimman projektin joukossa on yksi peli, joka on Unreal Engine 3-pelimoottorin esittelyyn tarkoitettu demo[15]: loput ovat jollain muulla tapaa WebGL-rajapintaa käyttäviä projekteja.

4 Pelinkehittäjän tarpeet

Takaisin sisällysluetteloon

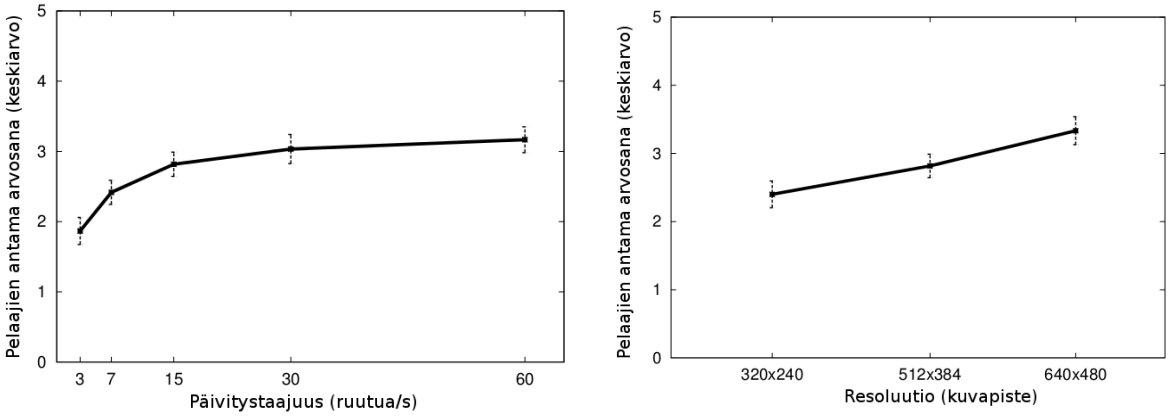
Erityisesti nopeita reaktioita vaativien pelien pelattavuus kärsii huomattavasti liian pienillä ruudun päivitysnopeuksilla. Ero pelaajan pelaamiskyvyssä korkeiden ja matalien piirtotaajuuksien välillä on huomattava: kuvat 1 ja 2 havainnollistavat Claypoolin tutkimusryhmän[16] saamia tuloksia tutkittaessa kuvanlaadun ja piirtotiheyden vaikutusta pelaajiin.



(a) Pelaajien saamat pisteet suhteutettuna päivitystiheyteen. Resoluutio 512x384 kuvapistettä.

(b) Pelaajien saamat pisteet suhteutettuna resoluutioon. Päivitystiheys on 15 ruutua sekunnissa.

Kuva 1: Claypoolin tutkimusryhmän[16] havaintoja grafiikan laadun vaikutuksista pelaamiskykyyn. Suurempi pistemäärä kuvastaa parempaa pelimenestystä.



(a) Pelaajien antamat arvosanat suhteutettuna päivitystiheyteen. Resoluutio 512x384 kuvapistettä.

(b) Pelaajien antamat arvosanat suhteutettuna resoluutioon. Päivitystiheys on 15 ruutua sekunnissa.

Kuva 2: Claypoolin tutkimusryhmän[16] saamia kyselytuloksia. Kyselyissä pyydettiin koehenkilöitä antamaan pelin visuaaliselle laadulle arvosana asteikolla 0-5.

Kuvan 1a mukaan piirtotaajudella on suuri vaikutus pelaajiin jotka pelaavat intensiivistä, nopeaa reagointikykyä vaativaa peliä. Claypoolin tutkimusryhmän löydökset[16] osoittavat näin ollen yhden pelinkehittäjän konkreettisen tarpeen. Liian vähäinen piirtotaajuus voi pahimmassa tapauksessa tehdä pelistä pelaamattoman.

Kuvien 1a ja 2a perusteella voidaan lisäksi päätellä että päivitystiheydellä on suuri vaikutus pelaajan kokemuksiin pelistä erityisesti tiheyden ollessa matala. Toisaalla vertaamalla kuvaa 2b muihin voimme nähdä että päivitystiheyden noustessa yli 30 ruutuun sekunnissa resoluutiolla on suurempi vaikutus pelaajien mielipiteeseen pelin visuaalisesta laadusta[16].

Näin ollen pelin sunnittelijan kannattaa rajata pelinsä sisältö sellaiseksi että se pystytään esittämään tarvittavan sujuvassa muodossa. Tällöin joidenkin pelityyppien näyttävyys saattaa rajautua huomattavasti jos ne ovat ollenkaan toteutettavissa. Jos WebGL on toimiva työkalu, se lisäisi pelinkehittäjien ilmaisuvoimaa selaimessa pelattavilla peleillä. Paljon laskentatehoa vaativat pelityypit ovat tähän mennessä olleet pakostakin natiivisovelluksia: jos WebGL toimii hyvin, se mahdollistaisi helpomman ja halvemman kehitystyön tällaisille peleille.

5 WebGL-rajapinnan ominaisuuksia pelinkehittäjän näkökulmasta

Takaisin sisällysluetteloon

5.1 Suorituskyky

Selkein WebGL:n tarjoama etu on suorituskyky. WebGL-rajapinta sallii laskennan suorittamisen näytönohjaimessa suorittimen sijaan, mikä on huomattavasti nopeampaa. Kuvasta 3 nähdään että Canvas 2D Context ja Flash ovat vertailussa selvästi hitaampia. C++-pohjainen yksinkertainen OpenGL-sovellus on yllättäen WebGL:ää hitaampi: natiivi toteutus täytyy optimoida ennen kuin se on tehokkaampi kuin WebGL-toteutus.[17] Ero ei suurimmaksi osaksi kuitenkaan johdu WebGL:n ja natiivin OpenGL:n välisistä eroista. Koska WebGL:ää kutsutaan verkkosivun koodista, selaimen on suoritettava JavaScript-koodia jokaista piirtoa varten. JavaScript-koodi on muutamia erikoistapauksia lukuunottamatta paljon hitaampaa kuin C++-koodi[18]. Täten suurin osa WebGL:n piirtämisajasta kuluu JavaScriptin kääntämiseen ja ajamiseen[17].

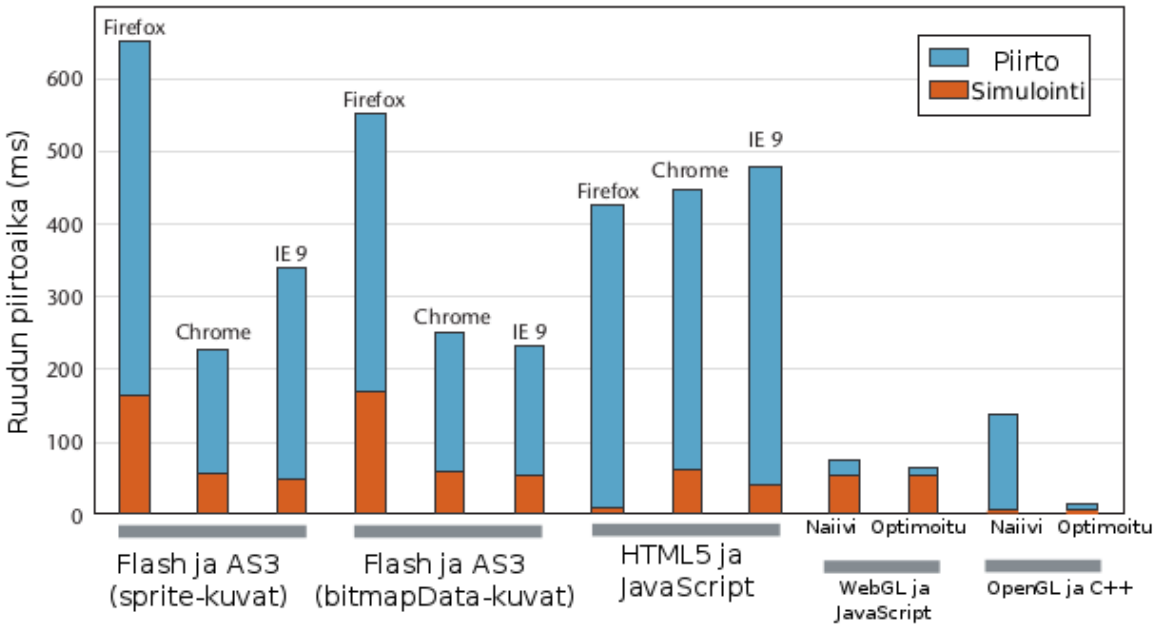
Pelinkehittäjän kannattaa siis ottaa huomioon, että WebGL on huomattavasti hitaampi kuin optimoitu natiivi sovellus, mikä oli odotettavissa. Huomionarvoisempaa on kuitenkin, että WebGL ei itse ole pääsyyllinen kyseiseen eroon: Taulukosta 1 ilmenee, että vaikka WebGL-koodin optimointi lyhensi piirtämisaikaa noin 80 %, kokonaisaika pieneni ainoastaan noin 23 %[1].

Taulukko 1: Hoetzleinin[17][19] mittaustulokset erilaisista toteutuksista piirtäessä 100000 objektia.

Implementaatio	**Simulointi (ms)**	**Piirto (ms)**	**Kokonaisaika (ms)**	**30Hz maksimi**[a]
HTML5 ja JavaScript, Firefox	9,7	417,0	426,7	7000
HTML5 ja JavaScript, Chrome	59,3	391,0	450,3	4000
HTML5 ja JavaScript, IE 9	39,0	443,3	482,3	6000
WebGL, naiivi, Chrome	54,0	23,0	77,0	70000
WebGL, optimoitu, Chrome	54,7	4,3	59,0	80000
OpenGL, naiivi	3,3	137,9	141,2	20000
OpenGL, optimoitu	3,3	7,5	10,8	400000

[a]Suurin määrä objekteja joita voitiin piirtää ja myös yltää 30 ruudun/sekunti päivitystiheyteen.

[1]WebGL, sprites, Chrome: piirtämisaika 4,3 ms, kokonaisaika 59,0 ms. WebGL, VBOs, Chrome: piirtämisaika 23,0 ms, kokonaisaika 77,0 ms. 1 - (4,3 / 23,0) = **0,813**. 1 - (59 / 77) = **23,4**.



Kuva 3: Hoetzleinin[17] havainnot eri teknologioiden nopeuksista. Oranssit palkit kuvastavat simulointiaikaa, siis aikaa joka vaadittiin piirrettävien objektien uusien sijaintien laskennassa (Simulation). Siniset palkit kuvaavat itse piirtämiseen (Rendering) kulunutta aikaa. Matalammat palkit vastaavat nopeampaa laskentaa. Mittauksessa piirrettiin 100000 objektia.

Pelinkehittäjälle olennaista on huomata, että kuvan 3 ”WebGL ja JavaScript” -osion kummassakin toteutuksessa WebGL-koodin osuus kuluneesta ajasta on paljon pienempi kuin JavaScript-koodin. Kehittäjä saattaa siis päästä paljon parempiin tuloksiin, jos hän keskittyy WebGL-koodin sijasta JavaScript-koodin optimointiin.

JavaScriptin toimintaa voi nopeuttaa niin sanotulla esikääntämisellä. Esikääntämisessä JavaScript-koodi muunnetaan niin että toiminnallisuus ei muutu, mutta koodin struktuuri on tehokkaampaa. Tunnetuin esikääntäjä lienee Google:n Closure-kääntäjä[20]. Myös muut parannukset ovat mahdollisia. Hoetzleinin[17] toteutuksessa esimerkiksi fysiikan laskenta suoritetaan JavaScript-koodissa. Tällaisen laskennan sirtäminen näytönohjaimeen saattaisi nopeuttaa piirtämistä huomattavasti.

5.2 WebGL-kirjaston piirteitä

Pelinkehittäjän on aina hyvä tietää, mihin valittu teknologia pystyy ja mitä puutteita sillä on. Di Benetto[21] on tunnistanut puutteita WebGL-kirjastosta.

5.2.1 Lataaminen

WebGL on matalan tason kirjasto joka oletusarvoisesti toimii sekä tietokoneilla että mobiililaitteilla. Tästä syystä siitä saattaa puuttua toiminallisuutta joka löytyy oletusarvona suuremmista kirjastoista. Yksi tärkeä ominaisuus nykyaikaisissa grafiikkakirjastoissa on asynkroninen lataaminen. Kuvaa piirrettäessä tietokoneen on haettava muistista erinäistä tietoa: tekstuureita, malleja ynnä muuta. Jos tällaiset tiedostot pitää ladata yksi kerrallaan ohjelma saattaa lakata reagoimasta käyttäjään kunnes kaikki tarvittava tieto on ladattu. Asynkroninen lataaminen sallii eri osien lataamisen samanaikaisesti, jolloin suoritin voi samalla reagoida käyttäjään. JavaScript salli tällaisen asynkronisuuden ainoastaan Web Workers rajapinnan kautta[22], joten ominaisuutta ei löydy oletuksena WebGL-kirjastoa käytettäessä.

5.2.2 HTML-elementtien rajapinnat

Yksi HTML5-kielen hyödyllisistä ominaisuuksista ovat elementti-kohtaiset rajapinnat. Esimerkiksi `<img>`-elemetin JavaScript-rajapinnasta löytyy onload-funktio, joka sallii kehittäjän määritellä mitä tehdään kun kyseisen elementin määrittelemä kuva on ladattu valmiiksi. JavaScriptissä tai WebGL:ssä ei ole tällaista toiminnallisuutta 3D-objekteille: HTML5 ei määrittele 3D-objekteille erityisiä elementtejä tai rajapintoja.

5.2.3 Eroavaisuudet täysimittaiseen kirjastoon

Tärkeä asia pitää mielessä on, että WebGL perustuu OpenGL ES-rajapintaan, joka vuorostaan sisältää vain osan täysimittaisen OpenGL-kirjaston toiminnallisuudesta (kuten OpenGL 4.0)[23]. Näin ollen kehittäjä joka on kokenut OpenGL-koodaaja ei välttämättä ole niin tehokas kuin voisi toivoa: WebGL-kirjaston rajallisuus OpenGL-kirjastoon verrattuna saattaa vaatia sopeutumista.

Kuitenkin perustyökalut muotojen piirtämiseen ovat rajapinnoissa samat. Taulukosta 2 nähdään, että melkein kaikki OpenGL-rajapinnan primitiivit löytyvät myös WebGL-rajapinnasta. Ainoa poikkeus on muoto (Patch) joka on tarkoitettu mosaiikkien luomiseen erilaisista primitiiveistä[24]. Näin ollen pelinkehittäjällä on käytössään samat perusprimittivit kummassakin rajapinnassa.

5.2.4 Laskentakirjastot

Korkeammalla tasolla WebGL-rajapinnassa on kuitenkin puutteita verratessa OpenGL-rajapintaan. 3D-grafiikan piirtämisessä käytetään huomattavia määriä erilaista matriisilaskentaa. Näitä funktioita ei löydy WebGL-rajapinnan kirjastosta, joten

Taulukko 2: OpenGL-[24] ja WebGL-rajapinnan[8, 5.14] primitiivit.

Muoto	**OpenGL 4.0**	**WebGL 1.0**
Pisteitä	`GL_POINT`	`POINTS`
Viivoja	`GL_LINES`	`LINES`
Jatkuva viiva	`GL_LINE_STRIP`	`LINE_STRIP`
Jatkuva, sulkeutuva viiva	`GL_LINE_LOOP`	`LINE_LOOP`
Kolmioita	`GL_TRIANGLES`	`TRIANGLES`
Sarja kolmioita	`GL_TRIANGLE_STRIP`	`TRIANGLE_STRIP`
Kolmioita joilla yhteinen kulma	`GL_TRIANGLE_FAN`	`TRIANGLE_FAN`
Muoto (Patch)	`GL_PATCH`	Ei

kehittäjän on joko toteutettava ne itse tai käytettävä kolmannen osapuolen kehittämää kirjastoa. Jos kehittäjän tarkoituksiin sopivaa, laadukasta kirjastoa ei ole tiedossa se merkitsee mahdollisesti huomattavaa riskiä kehityksessä.

5.3 WebGL-koodin Helppokäyttöisyys

Tässä osiossa helppokäyttöisyyttä katsastetaan vertailemalla WebGL-koodin monimutkaisuutta muihin toteutuksiin. Kirjaston helppokäyttöisyyttä voi myös katsastella esimerkiksi dokumentaation laadun ja koodin struktuurin näkökulmista. Tällainen tarkka katsastus ei kuitenkaan kuulu tämän työn puitteisiin, joten tässä keskitytään ainoastaan lähdekoodin monimutkaisuuden vertailuun.

Verrattavat toteutukset ovat erittäin tiiviisti kirjoitettu, joten niiden toiminnallisuus selitetään alustavasti koodin yhteydessä. Lähdekoodit ovat erinäisten toteutusten piirtosilmukoita, kaikki Hoetzleinin[17] tutkimuksesta. Piirtosilmukka yksinkertaisesti tarkoittaa sitä osaa koodista joka piirtää kuvan ruudulle. Ennen piirtosilmukoiden ajoa on jokaisessa toteutuksessa laskettu piirrettävien esineiden uudet sijainnit jollakin metodilla joka on kaikissa toteutuksissa hyvin samanlainen[17].

Koodi 1 kuvastaa Canvas 2D Context -rajapintaa hyödyntävää JavaScript-kielen toteutusta. Tämä on hyvin yksinkertainen piirtosilmukka joka ajetaan kokonaan suorittimessa. Vertaamalla tätä koodeihin 2 ja 3 nähdään että koodin määrä moninkertaistuu, jota voi pitää selvänä merkkinä monimutkaistumisesta.

Toisin kuin Canvas 2D Context-rajapinnan tapauksessa laskentaa ei suoriteta samassa muistissa. Kaikkien piirrettävien objektien uudet sijainnit on siirrettävä näytönohjaimen muistiin. OpenGL- ja WebGL-rajapintojen monimutkaisemmassa koodissa täytyy ensin sitoa objektien kuvat laskettuihin uusiin koordinaatteihin, tämä tieto on siirrettävä näytönohjaimelle ja lopulta näytönohjainta on pyydettävä piirtämään kuva.

Koodi 1: 2D Context-rajapintaa käyttävä, JavaScript-kielellä toteutettu piirtosilmukka[17]

```
for( var i = 0, j = particles.length; i < j; i++ )
    context.drawImage ( ball_img, particles[i].posX, particles[i].posY );
```

Koodi 2: OpenGL-rajapintaa käyttävä, C++-kilellä toteutettu piirtosilmukka[17]

```
float* dat = bufdat;
for (int n=0; n < num_p; n++ ) {
   *dat++ = pos[n].x; *dat++ = pos[n].y;
   *dat++ = pos[n].x+32; *dat++ = pos[n].y;
   *dat++ = pos[n].x+32; *dat++ = pos[n].y+32;
   *dat++ = pos[n].x; *dat++ = pos[n].y+32;
}
glEnable ( GL_TEXTURE_2D );
glBindTexture ( GL_TEXTURE_2D, img.getGLID() );
glBindBufferARB ( GL_ARRAY_BUFFER_ARB, vbo );
glBufferDataARB ( GL_ARRAY_BUFFER_ARB, sizeof(float)*2*4*num_p, bufdat,
    GL_DYNAMIC_DRAW_ARB );
glEnableClientState ( GL_VERTEX_ARRAY );
glVertexPointer ( 2, GL_FLOAT, 0, 0 );
glBindBufferARB ( GL_ARRAY_BUFFER_ARB, vbotex );
glEnableClientState ( GL_TEXTURE_COORD_ARRAY );
glTexCoordPointer(2, GL_FLOAT, 0, 0 );
glDrawArrays ( GL_QUADS, 0, num_p );
```

Canvas 2D Context-rajapinta on helpompi käyttää. Jokaista objektia kohden kerrotaan objektin uusi sijainti sekä kuva joka kyseiseen sijaintiin tulisi piirtää. Tietoa ei tarvitse siirtää muistien välillä, sillä se löytyy valmiiksi samasta välimuistista mistä koodi ajetaan. Näin Canvas 2D Context-rajapinnan koodi pysyy paljon yksinkertaisempana.

Koodi 2 ja Koodi 3 eivät kuitenkaan eroa toisistaan yhtä paljon kuin koodista 1: C++-koodin rivimäärä ei eroa WebGL-koodin rivimäärästä yhtä dramaattisesti kuin JavaScript-koodin rivimäärästä. OpenGL- ja WebGL-rajapinnat tosin eroavat toisistaan kattavuuden suhteen, kuten kappaleessa 5.2 eriteltiin. Kyseiset eroavaisuudet kuitenkin koskivat pääasiassa sellaisia henkilöitä joille jompi kumpi rajapinta oli ennestään tuttu. Lähdekoodin tasolla voidaan olettaa eroja olevan vähemmän, joten kummatkin rajapinnat ovat oletettavasti yhtä helppoja käyttää. Näin ollen pelinkehittäjän ei luultavasti tarvitse huolehtia itse koodin monimutkaistumisesta OpenGL-rajapintaan verrattuna.

5.4 Alustariippumattomuus

Web-sovellusten suurin etu on alustariippumattomuus: saman sovelluksen pitäisi toimia kaikilla alustoilla jotka tukevat selainta, jolla sovellus toimii. Teoriassa kehittäjän tarvitsee ainoastaan varmistaa pelin toimivan niillä selaimilla joilla on surimmat käyttäjämäärät saadakseen pelilleen mahdollisimman suuren potentiaalisen yleisön.

Koodi 3: WebGL-rajapintaa käyttävä, JavaScript-kielellä toteutettu piirtosilmukka[17]

```
for( var i=0, j=0, i < num_particles; i++ ) {
   vert[j++] = particle[i].posX; vert[j++] = particle[i].posY;
   vert[j++] = particle[i].posX+32; vert[j++] = particle[i].posY;
   vert[j++] = particle[i].posX+32; vert[j++] = particle[i].posY+32;
   vert[j++] = particle[i].posX; vert[j++] = particle[i].posY;
   vert[j++] = particle[i].posX+32; vert[j++] = particle[i].posY+32;
   vert[j++] = particle[i].posX; vert[j++] = particle[i].posY+32;
}
gl.bindBuffer(gl.ARRAY_BUFFER, geomVB);
gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.DYNAMIC_DRAW);
gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
mat4.identity(pMatrix);
mat4.scale ( pMatrix, [2.0/SCREEN_WIDTH, -2.0/SCREEN_HEIGHT, 1] );
mat4.translate ( pMatrix, [-(SCREEN_WIDTH)/2.0, -(SCREEN_HEIGHT)/2.0, 0] );
gl.bindBuffer(gl.ARRAY_BUFFER, geomVB);
gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute, geomVB.itemSize,
    gl.FLOAT, false, 0, 0);
gl.bindBuffer(gl.ARRAY_BUFFER, geomTB);
gl.vertexAttribPointer(shaderProgram.textureCoordAttribute, geomTB.itemSize, gl.FLOAT,
    false, 0, 0);
gl.activeTexture(gl.TEXTURE0);
gl.bindTexture(gl.TEXTURE_2D, neheTexture);
gl.uniform1i(shaderProgram.samplerUniform, 0);
gl.drawArrays(gl.TRIANGLES, 0, 6*num_particles );
```

Parhaassa tapauksessa ainoastaan pieniä osia koodista tarvitsee muuttaa, jotta koodi saadaan toimimaan eri selaimissa. Natiivit sovellukset taas saattavat pahimmillaan vaatia täysin eri kielet eri alustoja varten, jolloin samasta toteutuksesta voi olla useita ylläpidettäviä kopioita. Tällöin selaimessa toimiva sovellus olisi huomattavasti helpompi ylläpitää.

Käytännössä selainten välillä saattaa kuitenkin olla suuriakin eroja. Varsinkin JavaScriptin kääntönopeudet vaihtelevat huomattavasti käytetyimpien selainten välillä[17]. Suurempi ongelma on kuitenkin WebGL-rajapinnan toteutusten erot: jotkin piirto-ominaisuudet saattavat toimia aivan eri lailla riippuen selaimesta[25]. Näin ollen kehittäjän ei välttämättä ole helppoa saada peliään toimimaan kaikilla selaimilla: jotkin piirtoratkaisut saattaa olla pakollista muuttaa perustavanlaatuisesti jotta tuote saadaan toimimaan toisella selaimella.

Suurin ongelma WebGL:n alustariippumattomuudelle on kuitenkin laitteistotuki. Koska WebGL-rajapinta toimii laitteen näytönohjaimen kanssa, ovat mahdolliset selain-laite yhdistelmät hyvin moninaiset. Näin ollen selainten ja laitteiston yhteensopivuudet eivät ole taattuja, vaan laitteen toimivuus riippuu käytetystä selaimesta[26][27]. Kehittäjä ei näin ollen voi olla varma, että kaikki selaimen käyttäjät voivat pelata WebGL-pohjaisia pelejä.

Toisaalta on pidettävä mielessä, että selainvalmistajat pyrkivät tukemaan suurinta määrää käyttäjiä. Varsinkin jos WebGL:stä tulee suosittu teknologia voidaan olettaa, että selainvalmistajat varmistavat tuen ainakin kaikkein suosituimmille näytönohjainmalleille. Kehittäjä ei siis saa peliään toimimaan kaikkien potentiaalisten käyttäjien koneilla, mutta ero ei luultavasti ole dramaattinen.

6 Yhteenveto

Takaisin sisällysluetteloon

Olemassaoleva kirjallisuus ei ole erityisen kattavaa, mutta materiaalista on mahdollista johtaa pelinkehittäjälle suuntaa-antavia johtopäätöksiä.

Ensinnäkin pelinkehittäjän on huomioitava WebGL-rajapinnan suorituskykyyn liittyvät ominaispiirteet. Huomionarvoista on että WebGL-koodin suorituskyky ei vaikuta lopputulokseen yhtä vahvasti kuin se JavaScript-koodi josta WebGL-koodia ajetaan, joten suorituskykyä parantaessa kehittäjän kannattaa ehdottomasti ensin keskittyä JavaScript-koodin nopeuttamiseen.

Toiseksi WebGL-kirjastossa saattaa olla puutteita: rajapinta on tarkoitettu toimimaan niin päätelaitteilla kuin mobiililaitteilla, joten sen on tarkoitus olla kevyt ja monikäyttöinen. Kehittäjän täytyy varmistaa että osaaminen riittää tai käyttää jotakin kehitystä tukevaa, WebGL-kirjaston päälle kehitettyä kirjastoa.

Kolmanneksi WebGL ei ole puhtaasti alusta-riippumaton: erot suorituskyvyssä ja lopputuloksissa sekä rajoitteet tuetuissa näytönohjaimissa tarkoittavat, että kehittäjä ei voi olettaa pelin toimivan yhtä laajalti kuin esimerkiksi Canvas 2D-Context-rajapintaan pohjautuva peli. Kuitenkin koodin ylläpidettävyydessä saavutetut hyödyt voivat olla mittavia.

Yleisvaikutelma WebGL-rajapinnasta kielii keskeneräisyyttä. Rajapinta on verrattain nuori, eivätkä selainvalmistajat ole kaikesta päätellen saaneet toteutuksiaan toimimaan täysin yhteneväisesti. Tulevaisuudessa nämä eroavaisuudet saattavat kadota selainvalmistajien parantaessa toteutuksiaan. Pelinkehittäjän kannalta on oleellisinta on että WebGL on muiden toteutuksien välimaastossa. WebGL-koodi on huomattavasti monimutkaisempaa kuin Canvas 2D-Context-rajapintaa käyttävä koodi, mutta pystyy paljon tehokkaampaan laskentaan. Kuitenkaan se ei yllä optimoidun, natiivin sovelluksen tasolle. WebGL-rajapinta on huomattavasti alustariippumattomampi kuin natiivi sovellus, mutta ei kuitenkaan ole yhtä universaali kuin yksinkertaisemmat sovellukset.

WebGL-rajapinta soveltunee parhaiten juuri peleihin jotka jäävät näiden kahden ääripään väliin. Pelit jotka vaativat sujuvaa piirtoa johon Canvas 2D Context-rajapinta ei yksinkertaisesti riitä, mutta jotka eivät kuitenkaan vaadi keskivertoa tehokkaampia tietokoneita. WebGL tarjoaa parempaa alustariippumattomuutta kuin natiivi OpenGL-koodi, joten WebGL-rajapinta soveltuu pienemmille kehittäjäryhmille joilla ei ole resursseja työstää montaa eri versiota samasta pelistä.

WebGL-rajapinnan ominaisuuksista ei tätä työtä varten löytynyt erityisen paljon aiheen kannalta relevanttia kirjallisuutta. Tulevaisuudessa WebGL-rajapinnan

tutkimukset voisivat keskittyä esimerkiksi muistin käyttäytymiseen ja laskennan jakoon näytönohjaimen ja JavaScript-koodin välillä. Esimerkiksi voitaisiin tutkia kuinka paljon JavaScript-koodin esikääntäminen tai monien erilaisten kuvien piirtäminen vaikuttaa surituskykyyn.

Lähteet

[1] W3C and WHATWG. HTML5 differences from HTML4: Introduction. Saatavissa http://www.w3.org/TR/2011/WD-html5-diff-20110405/#introduction. Viitattu 15.02.2014.

[2] ECMA International. ECMA-262 ECMAScript Language Definition. Saatavissa http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf. Viitattu 11.03.2014.

[3] Mozilla Developer Network. JavaScript. Saatavissa https://developer.mozilla.org/en-US/docs/Web/JavaScript. Viitattu 11.03.2014.

[4] WHATWG. HTML - 4.12.4 - The canvas element. Saatavissa http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html. Viitattu 21.03.2014.

[5] Khronos Group. The OpenGL Graphics System: A Specification, 2011. Saatavissa http://www.opengl.org/registry/doc/glspec40.core.20100311.pdf. Viitattu 28.02.2014.

[6] Microsoft. DirectX Graphics and Gaming. Saatavissa http://msdn.microsoft.com/en-us/library/windows/desktop/ee663274. Viitattu 28.02.2014.

[7] Khronos Group. OpenGL ES. Saatavissa http://www.khronos.org/opengles/. Viitattu 28.02.2014.

[8] Khronos Group. WebGL Specification, 2013. Saatavissa http://www.khronos.org/registry/webgl/specs/latest/1.0/. Viitattu 15.02.2014.

[9] Jesus Jimenez, Jaime Cruz, and Juan Ruiz de Miras. High performance 3D visualization on the Web: a biomedical case study. Saatavissa http://iwbbio.ugr.es/papers/iwbbio_078.pdf. Viitattu 07.03.2014.

[10] Mobeen Movania Mobeen and Lin Feng. High-Performance Volume Rendering on the Ubiquitous WebGL Platform. *High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on*, pages 381–388, 2012. Saatavissa http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6332197. DOI: 10.1109/HPCC.2012.58. Viitattu 07.03.2014.

[11] +360°. Car Visualizer. Saatavissa http://carvisualizer.plus360degrees.com/threejs/. Viitattu 10.03.2014.

[12] Bentar Priyopradono and Fajar A. Perdana. Web3D Publishing Interior Design and Residential Collection based on WebGL Technology. Saatavissa http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.278.5819&rep=rep1&type=pdf. Viitattu 07.03.2014.

[13] WebGL.com. WebGL - Games. Saatavissa http://www.webgl.com/category/webgl-games/. Viitattu 07.03.2014.

[14] Stefan Haustein, Joel Webber, Matthias Büchner, Ray Cromwell, and moogle(käyttäjänimi). quake2-gwt-port. Saatavissa http://quake2playn.appspot.com/. Viitattu 10.03.2014.

[15] WebGL.com. WebGL Game Demo - Unreal Engine 3 (Epic Citadel). Saatavissa http://www.webgl.com/2013/05/webgl-game-demo-unreal-engine-3-epic-citadel/. Viitattu 10.03.2014.

[16] Mark Claypool, Kajal Claypool, and Feissal Damaa. The effects of frame rate and resolution on users playing First Person Shooter games. In *Electronic Imaging 2006*, pages 607101–607101–11. International Society for Optics and Photonics, 2006. Saatavissa http://web.cs.wpi.edu/~claypool/papers/fr-rez/paper.pdf. Viitattu 15.02.2014. DOI: 10.1117/12.648609.

[17] Rama C. Hoetzlein. Graphics Performance in Rich Internet Applications. *Computer Graphics and Applications, IEEE*, 32(5):98–104, 2012. Saatavissa IEEE Explore, DOI: 10.1109/MCG.2012.102. Viitattu 21.02.2014.

[18] Fredrik Smedberg. Performance analysis of JavaScript, 17.05.2010 2010. Saatavissa http://www.diva-portal.org/smash/get/diva2:321661/FULLTEXT01.pdf. Viitattu 21.02.2014.

[19] Rama C. Hoetzlein. Sprite Testing for Rich Internet Applications - Raw Data.

[20] Google Developers. Closure Tools - Compiler. Saatavissa https://developers.google.com/closure/compiler/. Viitattu 11.03.2014.

[21] Marco Di Benedetto, Federico Ponchio, Fabio Ganovelli, and Roberto Scopigno. SpiderGL: a JavaScript 3D graphics library for next-generation WWW. In *Proceedings of the 15th International Conference on Web 3D Technology*, pages 165–174. ACM, 2010. Saatavissa ACM Digital Library, DOI: 10.1145/1836049.1836075. Viitattu 18.02.2014.

[22] WHATWG. HTML: 10 - Web workers, 21.02.2014. Saatavissa http://www.whatwg.org/specs/web-apps/current-work/multipage/workers.html. Viitattu 21.02.2014.

[23] Khronos Group. WebGL and OpenGL Differences. Saatavissa http://www.khronos.org/webgl/wiki_1_15/index.php/WebGL_and_OpenGL_Differences. Viitattu 21.02.2014.

[24] Khronos Group. Saatavissa https://www.opengl.org/wiki/Primitive. Viitattu 21.03.2014.

[25] Jari-Pekka Voutilainen. Vuorovaikutteisten Web-sovellusten kehittäminen, 2011. Saatavissa http://dspace.cc.tut.fi/dpub/bitstream/handle/123456789/20804/voutilainen.pdf?sequence=3. Viitattu 25.02.2014.

[26] Mozilla.org. Blocklisting/Blocked Graphics Drivers. Saatavissa https://wiki.mozilla.org/Blocklisting/Blocked_Graphics_Drivers. Viitattu 18.02.2014.

[27] Khronos Group. BlacklistsAndWhitelists, 2014. Saatavissa http://www.khronos.org/webgl/wiki/BlacklistsAndWhitelists. Viitattu 15.02.2014.