Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan koulutusohjelma

WebGL HTML5-pelinkehittäjän näkökulmasta

Kandidaatintyö

14. helmikuuta 2014

Atte Isopuro

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan koulutusohjelma

KANDIDAATINTYÖN
TIIVISTELMÄ

Tekijä:	Atte Isopuro
Työn nimi:	WebGL 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 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 piirtämiseen.

Erityisesti monimutkaista ja tarkkaa interaktiota vaativat pelit tarvitsevat nopeaa grafiikan piirtämistä ruudulle. 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 grafiikka-esitykseen liittyviä teknologioita, kuten Canvas 2D context tai Flash. Teknologioita verrataan WebGL:ään, mutta niiden erityispiirteitä ei tutkita tässä työssä tarkemmin. Työssä ei myöskään pyritä toteamaan WebGL:n paremmuudesta mitään konkreettista. Löytöjen perusteella peinkehittäjän on yhä itse päätettävä mikä teknologia soveltuu hänen projektiinsa parhaiten.

Työ toteutetaan kirjallisuuskatsauksena. Aineisto koostuu tieteellisistä artikkeleista ja kirjoista jotka käsittelevät WebGL:ää. Erityisesti viitataan teksteihin joissa on konkreettisten toteutusten yhteydessä havaittu WebGL:n ominaispiirteitä.

Ensiksi alustetaan aiheen taustaa: esitellään HTML5 ja grafiikan piirtäminen. Seuraavaksi selvitetään tarkemmin pelinkehittäjän tarpeet sekä WebGL-spesifikaation tarjoamat edut. Tämän jälkeen eritellään tutkimusmenetelmät: lähteiden rajauskriteerit selvitetään ja kerrotaan kuinka ne on määrätty luotettaviksi. Lopuksi aineistosta kerätty tieto kootaan eheäksi kokonaisuudeksi, josta viimein tullaan lopullisiin johtopäätöksiin.

2 Taustaa

Takaisin sisällysluetteloon

2.1 HTML5

HTML5 on uusin versio HyperText Markup Language-kielestä. Yksi HTML5:n tuomia uudistuksia on erinäisten media-formaattien sisäistäminen itse spesifikaatioon erinäisten rajapintojen kautta[13]. Näin ollen sisällöntuottajien ja käyttäjien ei tarvitse huolehtia kolmansien osapuolten liitännäisistä kuten Flash. Tällöin sisällöntuottaja voi olla varma siitä että kaikke 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.

2.2 Intensiivinen grafiikka

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 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 graafiseen laskentaan erikoistuneeseen yksikköön: näytönohjaajaan.

2.2.2 OpenGL

OpenGL on alusta- ja kieliriippumaton ohjelmointirajapinta tietokoneiden näytönohjaimille[4]. Ohjelmoija voi käyttää OpenGL kirjastoa tehdäkseen kutsuja näytönohjaimelle, ilman että hänen tarvitsee käyttää näytönohjaimen omaa, matalan tason käskykantaa. Vastaava teknologia on DirectX[9], Microsoftin ylläpitämä lisensioitava rajapinta-kirjasto.

2.2.3 OpenGL ES

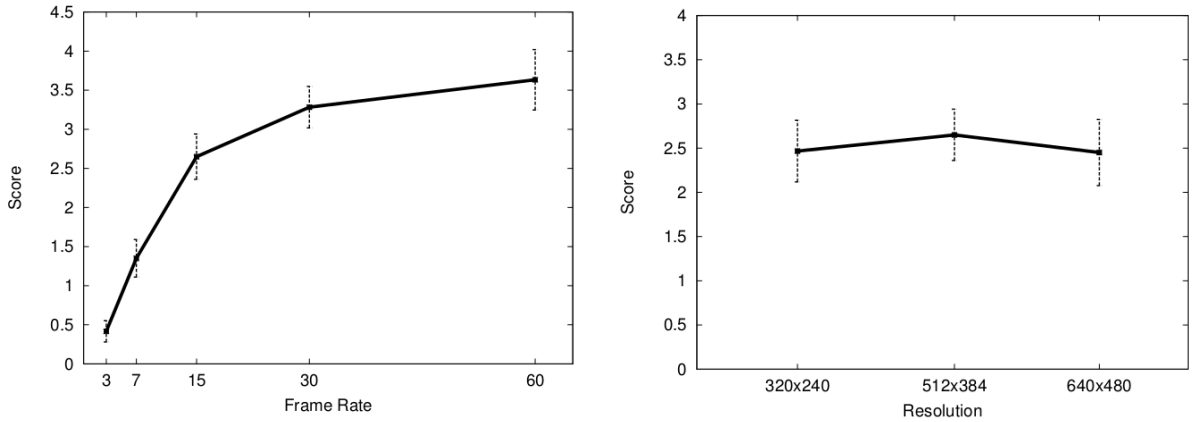
OpenGL ES (Embedded Systems) on OpenGL-kirjaston versio joka on tarkoitettu käytettäväksi sulautetuissa järjestelmissä, kuten pelikonsoleissa, puhelimissa ja tablettitietokoneissa[5].

2.2.4 WebGL

WebGL on JavaScript-rajapinta HTML5:n canvas-elementtiä varten. WebGL perustuu pitkälti OpenGL ES:ään[6]. Rajapinta sallii näytönohjaimen käskyttämisen JavaScript-koodista. Näin ollen JavaScript-ohjelmoija voi piirtää <canvas>-elementille kuvia WebGL:llä samalla tavoin kuin natiivisovellus tekisi esimerkiksi C++:aa ja OpenGL:ää käyttäen.

2.3 Pelinkehittäjän tarpeet

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 saamia tuloksia tutkittaessa kuvanlaadun ja piirtotiheyden vaikutusta pelaajiin.

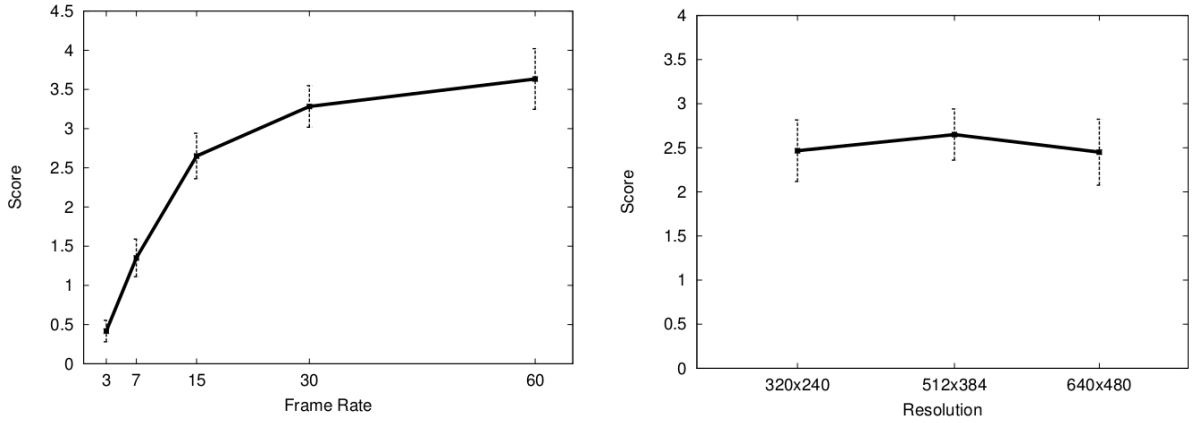


(a) Pelaajien saamat pisteet suhteutettuna päivitystiheyteen. Resoluutio 512x384 kuvapistettä.

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

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

Kuvien 1a ja 2a perusteella voidaan 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[1].



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

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

Kuva 2: Claypoolin tutkimusryhmän[1] saamia kyselytuloksia. Kyselyissä pyydettiin koehenkilöitä antamaan pelin visuaaliselle laadulle arvosana asteikolla 0-5.

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. Paljon laskentatehoa vaativat pelityypit ovat tähän mennessä olleet pakostakin natiivisovelluksia: jos WebGL toimii hyvin, se mahdollistaisi helpomman ja halvemman kehitystyön tällaisille peleille.

3 Sovelluksia

Takaisin sisällysluetteloon

3.1 Tuoteesittelyä

3.2 Lääketieteellistä kuvantamista

3.3 Pelejä

4 Tutkimusmateriaali

Takaisin sisällysluetteloon

4.1 Suorituskyky

Selkein WebGL:n tarjoama etu on suorituskyky: mahdollisuus suorittaa laskentaa näytönohjaimessa suorittimen sijasta on huomattavasti nopeampaa: 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.[8] 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[11]. Täten suurin osa WebGL:n piirtämisajasta kuluu JavaScriptin kääntämiseen ja ajamiseen[8].

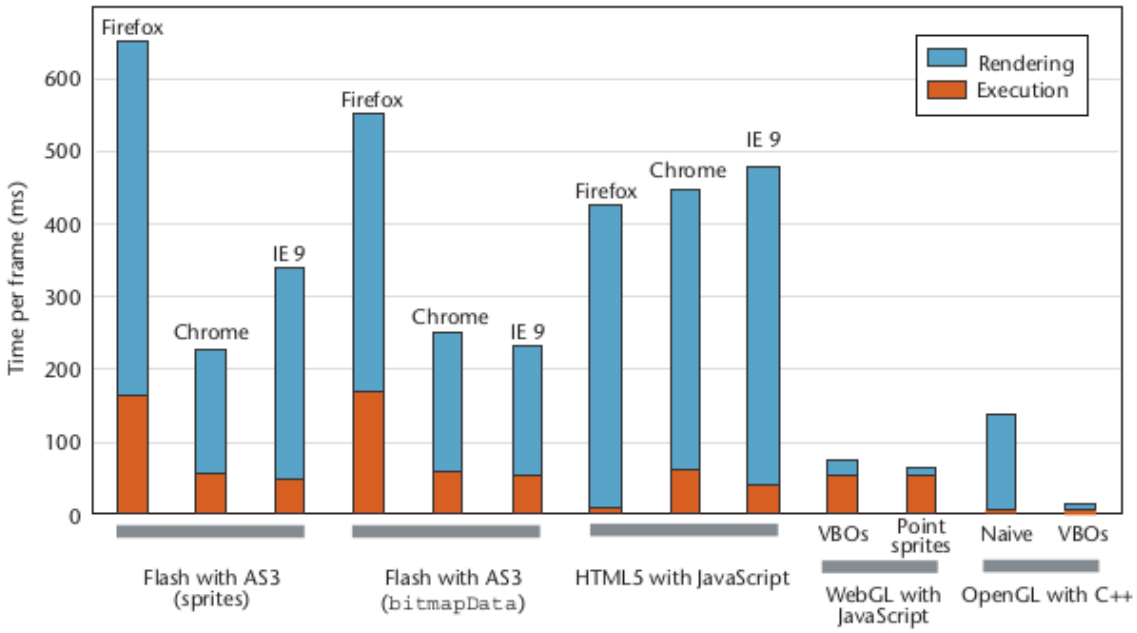
Pelinkehittäjän kannattaa siis ottaa huomioon että WebGL on huomattavasti hitaampi kuin optimoitu natiivi sovellus, mikä oli odotettavissa. Huomionarvoisempaa on kuitenkin että WebGL ei itse ole pääsyyllinen kyseiseen eroon: Taulusta 1 ilmenee että vaikka WebGL-koodin optimointi lyhensi piirtämisaikaa noin 80%, kokonaispiirtämisaika pieneni ainoastaan noin 23%[1]. Figure 3 esittelee havainnot graafisessa muodossa.

Taulukko 1: Hoetzleinin[8] mittaustulokset erilaisista toteutuksista.

Implementaatio	Simulointi (ms)	Piirto (ms)	Kokonaisaika (ms)
WebGL, ei-optimoitu, Chrome	54,0	23,0	77,0
WebGL, optimoitu, Chrome	54,7	4,3	59,0

Pelinkehittäjälle olennaista on huomata että ”WebGL with JavaScript”:in nopeammassa toteutuksessa WebGL:n osuus kuluneesta ajasta on paljon pienempi kuin JavaScriptin: kehittäjä saattaa päästä paljon parempiin tuloksiin jos hän keskittyy WebGL:n sijasta JavaScriptin optimointiin. Hoetzleinin[8] toteutuksessa esimerkiksi fysiikan laskenta suoritetaan JavaScript-koodissa. Tällaisen laskennan sirtäminen näytönohjaimeen saattaisi nopeuttaa piirtämistä huomattavasti.

[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[8] havainnot eri teknologioiden nopeuksista. Oranssit palkit kuvastavat simulointiaikaa, siis se aika joka vaadittiin laskemaan piirrettävien objektien uudet sijainnit (Simulation). Siniset palkit kuvaavat itse piirtämiseen (Rendering) kulunutta aikaa. Matalammat palkit vastaavat nopeampaa laskentaa.

4.2 Kirjasto

4.2.1 Kattavuus

Pelinkehittäjän on aina hyvä tietää mihin valittu teknologia pystyy ja mitä puutteita sillä on. Di Benetto[2] on tunnistanut muutaman tällaisen:

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 ym. Jos tällaiset tiedot 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[14], joten ominaisuutta ei löydy oletuksena WebGL:ää käyttäessä.

Yksi HTML5:n 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 tällaisille objekteille erityisiä elementtejä tai rajapintoja.

Tärkeä asia pitää mielessä WebGL:ssä on että se perustuu OpenGL ES:ään, joka vuorostaan sisältää vain osan täysimittaisen OpenGL-kirjaston toiminnallisuudesta (kuten OpenGL 3.0)[3]. Näin ollen kehittäjä joka on kokenut OpenGL-koodaaja ei välttämättä ole niin tehokas kuin voisi toivoa: WebGL:n rajallisuus OpenGL:ään verrattuna saattaa vaatia sopeutumista.

Huomattavin puute WebGL:ssä muihin grafiikkakirjastoihin verrattuna on lineaarialgebran puuttuminen. 3D-grafiikan piirtämisessä käytetään huomattavia määriä erilaista matriisilaskentaa. Näitä funktioita ei löydy WebGL:stä, joten kehittäjän on joko toteutettava ne itse tai käytettävä kolmannen osapuolen kehittämää kirjastoa.

4.2.2 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, eikä niiden tarkkaa toimintaa ole tarpeellista ymmärtää. Lähdekoodit ovat erinäisten toteutusten piirtosilmukoita, kaikki Hoetzleinin[8] 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[8].

Koodi 1 kuvastaa Canvas 2D Context -rajapintaa hyödyntävää JavaScript-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.

Koodi 2 ja Koodi 3 eivät kuitenkaan eroa toisistaan yhtä paljon: C++-koodin rivimäärä ei eroa WebGL-koodin rivimäärästä yhtä dramaattisesti kuin JavaScript-koodin rivimäärästä. OpenGL- ja WebGL-rajapinnat kuitenkin eroavat toisistaan kattavuuden suhteen, kuten kappaleessa 4.2.1 eriteltiin. Nämä eroavaisuudet kuitenkin koskettavat pääasiassa sellaisia henkilöitä joille jompi kumpi rajapinta on ennestään tuttu. Lähdekoodin tasolla voidaan olettaa eroja olevan vähemmän, joten kummatkin rajapinnat ovat oletettavasti yhtä helppoja käyttää.

Koodi 1: 2D Context-rajapintaa käyttävä, JavaScript-kielellä toteutettu piirtosilmukka[8]

```
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[8]

```
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 );
```

Koodi 3: WebGL-rajapintaa käyttävä, JavaScript-kielellä toteutettu piirtosilmukka[8]

```
1  for( var i=0, j=0, i < num_particles; i++ ) {
2     vert[j++] = particle[i].posX; vert[j++] = particle[i].posY;
3     vert[j++] = particle[i].posX+32; vert[j++] = particle[i].posY;
4     vert[j++] = particle[i].posX+32; vert[j++] = particle[i].posY+32;
5     vert[j++] = particle[i].posX; vert[j++] = particle[i].posY;
6     vert[j++] = particle[i].posX+32; vert[j++] = particle[i].posY+32;
7     vert[j++] = particle[i].posX; vert[j++] = particle[i].posY+32;
8  }
9  gl.bindBuffer(gl.ARRAY_BUFFER, geomVB);
10 gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.DYNAMIC_DRAW);
11 gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
12 gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
13 mat4.identity(pMatrix);
14 mat4.scale ( pMatrix, [2.0/SCREEN_WIDTH, -2.0/SCREEN_HEIGHT, 1] );
15 mat4.translate ( pMatrix, [-(SCREEN_WIDTH)/2.0, -(SCREEN_HEIGHT)/2.0,
      0] );
16 gl.bindBuffer(gl.ARRAY_BUFFER, geomVB);
17 gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
      geomVB.itemSize, gl.FLOAT, false, 0, 0);
18 gl.bindBuffer(gl.ARRAY_BUFFER, geomTB);
19 gl.vertexAttribPointer(shaderProgram.textureCoordAttribute,
      geomTB.itemSize, gl.FLOAT, false, 0, 0);
20 gl.activeTexture(gl.TEXTURE0);
21 gl.bindTexture(gl.TEXTURE_2D, neheTexture);
22 gl.uniform1i(shaderProgram.samplerUniform, 0);
23 gl.drawArrays(gl.TRIANGLES, 0, 6*num_particles );
```

4.3 Alustariippumattomuus

Web-sovellusten suurin etu on alustariippumattomuus: sama sovellus toimii teoriassa 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.

Käytännössä selainten välillä saattaa kuitenkin olla suuriakin eroja. Varsinkin JavaScriptin kääntönopeudet vaihtelevat huomattavasti käytetyimpien selainten välillä[8]. Suurempi ongelma on kuitenkin WebGL:n toteutusten erot: jotkin piirto-ominaisuudet saattavat toimia aivan eri lailla riippuen selaimesta[12]. 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. WebGL:n käyttäessä laitteen näytönohjainta ovat mahdolliset selain-laite yhdistelmät paljon moninaisemmat. Näin ollen selainten ja laitteiston yhteensopivuudet eivät ole taattuja, vaan laitteen toimivuus riippuu käytetystä selaimesta[10][7]. 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.

5 Johtopäätökset

Takaisin sisällysluetteloon

Lähteet

[1] Mark Claypool, Kajal Claypool ja Feissal Damaa. The effects of frame rate and resolution on users playing First Person Shooter games. *Electronic Imaging 2006*, sivut 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.

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

[3] 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.

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

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

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

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

[8] 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.

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

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

[11] 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.

[12] 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.

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

[14] 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.