Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Anže Pečar

Medplatformni razvoj grafično intenzivnih aplikacij

diplomsko delo

univerzitetni študijski program računalništvo in informatika

Mentor: doc. dr. Matjaž Kukar

Ljubljana 2013

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

[1] V dogovorju z mentorjem lahko kandidat diplomsko delo s pripadajočo izvorno kodo izda tudi pod katero izmed alternativnih licenc, ki ponuja določen del pravic vsem: npr. Creative Commons, GNU GPL. V tem primeru na to mesto vstavite opis licence, na primer tekst [9]

Namesto te strani **vstavite** original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden odda izdelek v vezavo! Glej tudi sam konec Poglavja **??** na strani **??**.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Anže Pečar, z vpisno številko **63060257**, sem avtor diplomskega dela z naslovom:

Medplatformni razvoj grafično intenzivnih aplikacij

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Matjaža Kukarja,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 11. januarja 2013 Podpis avtorja:

Na tem mestu zapišite, komu se zahvaljujete za izdelavo diplomske naloge. Pazite, da ne boste koga pozabili. Utegnil vam bo zameriti. Temu se da izogniti tako, da pozabite na celo zahvalo.

Posvetilo

Kazalo

Povzetek

Abstract

1 Uvod **1**

2 Pregled platform **3**

2.1 Zmogljivosti 5

3 Metode **7**

3.1 Spletne aplikacije 7

3.2 Spletne aplikacije z V8-gl 13

3.3 libgdx[3] 14

3.4 playn[8] 14

3.5 Unity[6] 14

3.6 Parse[7] 14

3.7 C++ 14

3.8 Programski jezik Haxe 15

3.9 Flash 15

4 Primeri **17**

5 Sklepne ugotovitve **19**

6 ASM.js **21**

Povzetek

Abstract

Poglavje 1

Uvod

Mobilne naprave dandanes postajajo vse bolj vsakdanje. Pametni telefoni imajo v sebi več procesorske moči, kot namizni računalniki izpred parih let. Poleg procesorske moči praviloma vsebujejo tudi grafične procesne enote, ki jih razvijalci lahko izkoristijo za razvoj grafično intenzivnih aplikacij. Poleg telefonov pa so se pojavili tudi tablični računalniki, ki imajo praviloma še boljše karakteristike kot pametni telefoni.

Med posameznimi proizvajalci telefonov in tablic obstajajo velike razlike v razvojnem okolju. Vsak izmed mobilnih operacijskih sistemov uporablja drug programski jezik za razvoj nativnih aplikacij pa tudi pri izvedbi različnih knjižnic (npr. OpenGL ES) se porajajo razlike. Razvoj grafične aplikacije, ki bi jo napisali enkrat in bi delovala povsod, je tako skorajda nemogoč.

Problem postane še težji, če želimo poleg vseh mobilnih naprav podpreti še namizne računalnike. Sedaj imamo poleg različnih programskih jezikov in knjižnic, še različne možnosti interakcije z uporabnikom - vnos z dotikom na mobilnih napravah in vnos z miško in tipkovnico na namiznih računalnikih.

Poglavje 2

Pregled platform

Danes lahko grafično intenzivne aplikacije poganjamo na ogromnem naboru različnih naprav. Grafično pospeševanje izrisovanja 3D objektov, ki je bilo še pred kratkim omejeno na namizne računalnike in konzole, je sedaj možno tudi na mobilnih telefonih in grafičnih tablicah. Razvoj aplikacij za mobilne platforme je nekoliko bolj zahteven, saj moramo poleg okrnjenega nabora ukazov in slabše strojne zmogljivosti, paziti tudi na porabo baterije.

2.0.1 Namizni računalniki

2.0.2 Mobilne platforme

Na trgu najdemo pester nabor mobilnih platform. Največji igralci v času pisanja so Google s svojim odprtokodnim sistemom Android, Apple s svojim sistemom iOS, nekaj tržnega deleža pa imata tudi podjetji Microsoft, z mobilno različico opercijskega sistema Windows (Windows 7, 8 RT, Phone), ter podjetje BlackBerry z istoimenskim naborom pametnih telefonov namenjenim predvsem poslovnim uporabnikom.

Poleg že obstoječih pa bodo v bližnji prihodnosti na trg stopili tudi novi igralci. Fundacija Mozilla je razvila Firefox OS, podjetje Canonical pa pripravlja različico Ubuntu operacijskega sistema, ki bo delovala tudi na mobilnih telefonih in tablicah.

V zadnjem letu so postale zanimivi računalniki s čipom na eni sami ploščici (prevedi single-board computer). Tak primer je Rasperry Pi, računalnik, ki je bil razvit s namenom promoviranja učenja osnovnih temeljev računalništva.

2.0.3 iOS

Prvi iPhone je bil predstavljen 9. januarja 2007. Od ostalih mobilnih naprav na tržišču se je razlikoval z dodelanim uporabniškim vmesnikom in zaslonom občutljivim za večprstne dotike. Bil je tudi eden izmed prvih mobilnih telefonov brez tipkovnice - uporabnik je do vseh funkcij telefona dostopal preko na dotik občutljivega zaslona.

V nasljednih letih je podjetje Apple Inc. dodalo prvemu telefonu nekaj dodatnih funkcij kot so 3g povezljivost, izboljšana zadnja kamera, zaslon z višjo resolucijo, dvo jederni procesor itd.

Naprave danes podpirajo OpenGL ES 1.1 in 2.0.

Razvoj aplikacij poteka v jeziku Objective C (objektni C), vendar je možen samo na sistemih z Mac OSX operacijskim sistemom.

2.0.4 Android

Android je operacijski sistem, ki temelji na Linux jedru. Operacijski sistem je razvilo podjetje Android Inc., s finančno pomočjo Googla, ki je leta 2005 primarno podjetje tudi kupil. Prvi mobilni telefon z Android operacijskim sistemom je bil prodan oktobra 2008.

Izvorna koda operacijskega sistema je odprta in dostopna pod Apache licenco.

Android podpira OpenGL ES 1.1 in 2.0 od verzije 2.2 dalje.

2.0.5 Windows

2.0.6 Firefox OS

2.0.7 RaspberryPi

2.0.8 Ubuntu

2.1 Zmogljivosti

Edina skupna točka vseh omenjenih

OpenGL zmonžnosti itd.

Poglavje 3

Metode

3.1 Spletne aplikacije

Na področju spletnih tehnologij je bilo v zadnjih letih narejeno veliko. S pojavom spletnega brskalnika Google Chrome se je začela prava tekma za hitrost. Ker so spletne aplikacije postajale vedno bolj zahtevne, je bilo prvenstvenega pomena, da brskalniki postanejo hitri in odzivni.

Brskalniki so začeli med seboj začeli tekmovati, kdo lahko hitrejše izvaja Javascript kodo, rezlutate te tekme pa je bil, da so v roku parih let vsi novejši brskalniki sposobni hitro izvajati tudi bolj zahtevne Javascript aplikacije.

S pojavom pametnih telefonov in tablic so spletne tehnologije postale dostopne tudi na mobilnih platformah. Zaradi strojnih omejitev in predvsem življenske dobe baterij, so bili hitri in učinkoviti Javascript pogoni na mobilinih platformah celo bolj pomembni kot na namiznih računalnikih.

Kot rezultat vseh prizadevanj za izboljšanje učinkovitosti in hitrosti izvajanja izvorne kode spletnih strani, lahko danes tako na namiznih računalnikih in mobilnih napravah izvajamo zahtevne aplikacije, ki se izvajajo znotraj spletnega brskalnika.

Programski jezik Javascript je pravzaprav ena izmed redkih skupnih točk pametnih telefonov, tablic in namiznih računalnikov. Za razvoj domorodnih aplikacij za specifične naprave je potrebno uporabljati programski jezik, ki je

določen s strani proizvajalca (avtorja mobilnega operacijskega sistema). Prav vse mobilne naprave pa imajo naložen brskalnik, v katerem lahko poganjamo spletne aplikacije napisane v Javascriptu.

Prav vse mobilne platforme imajo tudi takšno ali drugačno implementacijo takoimenovanega elementa za spletni pogled (*webview*). Ta element nam omogoča prikaz določene spletne strani znotraj domorodne aplikacije. Na ta način lahko spletno aplikacijo zapakiramo v ovitek (wrapper), ki se potem iz vidika končnega uporabnika obnaša slično domorodnim aplikacijam. Zanimiva izjema je Mozillin mobilni operacijski sistem FirefoxOS, ki spletne aplikacije smatra kot domorodne in ovitki niso potrebni.

3.1.1 2D platno

Sodobni brskalniki nam za izris svojih oblik na zaslon poleg HTMLja in CSSa ponujajo tudi uporabo *platna* (angl. canvas).

Element *platno* se je pojavilo kot Appleov eksperiment znotraj Mac OSX Webkit komponente leta 2004. Uporabljen je bil v spletnem brskalniku Safari in za vtičnike v Dashboard aplikaciji. Leto kasneje so podporo *canvas* elementu dodali tudi Gecko brskalniki, leta 2006 pa tudi spletni brskalnik Opera. Istega leta je organizacija Web Hypertext Application Technology Working Group (WHATWG) element standardizirala. Internet Explorer je dodal domorodno podporo za *platno* v verziji 8.

platno je danes dobro podprto v vseh modernih spletnih brskalnikih, tudi na mobilnih napravah. Na določenih platformah in brskalnikih lahko uporablja tudi strojno pospeševanje, vendar se ta funkcija zaenkrat smatra še kot eksperimentalna in praviloma ni dostopna na mobilnih napravah. Zaradi tega dejstva obstajajo omejitve kompleksnosti, ki jih s *platno*m lahko dosežemo.

Platno je namenjeno izrisovanju dvo dimenzionalnih oblik na zaslon. Programerju nudi preprost API za risanje raznih oblik ($fillRect$, $fillCircle$), risanje besedila ($fillText$), risanje poti ($moveTo$, $lineTo$), risanje slik ($drawImage$) in tudi risanje gradientov ($drawGradient$). Nudi tudi dostop do operacij nad

posameznimi oblikami, kot so na primer premakni (*translate*), zavrti (*rotate*) itd.

Programer lahko aplikacijo s *platnom* razvija iz udobja namiznega računalnika. Vmesne rezultate dela lahko preverja v svojem spletnem brskalniku in šele nato prenese aplikacijo na mobilno napravo. Pri razvijanju aplikacije na namiznem računalniku mora biti programer še posebaj pozoren na strojne omejitve mobilnih naprav.

Prednost uporabe dvodimenzionalnega platna je, v dobri medplatformni podpori tako na namiznih računalnikih kot tudi na mobilnih napravah. Cena dobro podprtosti pa so omejene zmožnosti. Izrisovanje zahtevnejših slik lahko postane počasno, izrisovanje v treh dimenzijah pa zaradi slabe podpore strojnemu pospeševanju skorajda nemogoče.

Kljub pomankljivostim lahko s pomočjo platna na preprost način napišemo grafično intenzivno aplikacijo, ki bo delala na večih platformah.

Strojno pospeševanje

2D *platno* na določenih konfiguracijah omogoča tudi strojno pospeševanje. Z omogčenim strojnim pospeševanjem centralna procesna enota prenese nekaj svojega dela na grafično procesno enoto. Na ta način se lahko hitrost izrisovanja močno poveča.

Strojno pospešeno *platno* zaenkrat še ni povsod na voljo in zato na prednosti, ki jih prenaša, še ne gre računati. Brskalnik Google Chrome je na primer dodal podporo v verziji 18 (Marec 2012), vendar strojno pospeševanje še vedno ni povsod (Linux, Android).

3.1.2 3D platno WebGL

WebGL je medplatformni API, uporabljen za delo s 3D grafiko znotraj spletnega brskalnika. Je kontekst *canvas*a, ki ima direkten dostop do grafične kartice preko GLSL jezika za pisanje senčilnikov (shaders).

WebGL temelji na OpenGL ES2.0 standardu, ki je na voljo na namiznih računalnikih in mobilnih napravah, kjer WebGL kot tak morebiti še ni

podprt.

WebGL je nastal iz eksperimentov Vladimir Vukićevića, zaposlenega pri Mozilli. Leta 2006 je začel delati na pospešenem "3D platnu za splet". Do konca leta 2007 sta tako Mozilla kot Opera imeli delujočo implementacijo WebGL APIja. Leta 2009 je neprofitna organizacija Khronos ustanovila skupino za delo na WebGLu (WebGL Working Group). Člani skupine so bili tudi Apple, Google, Mozilla, Opera in drugi. Prva verzija speficikacije je bila izdana marca 2011.

Mozilla je dodala podporo WebGLu v Firefoxu 4.0, Google v Chromu od verzije 9 naprej, Apple je dodal podporo v Safari 6.0, v Operi pa se je podpora pojavila v verziji 11, vendar je bila privzeto izklopljena. Internet Explorer je dodal podporo WebGLu šele v verziji 11, ki je v času pisanja na voljo samo kot predogled v okviru Windows 8.1 verzije za razvijalce.

Na mobilnih brskalnikih je stanje še slabše. Večina mobilnih brskalnikov še nima vgrajene podpore (Safari na iOS) ali pa je le ta še v fazi preizkušanja (Chrome na Android). WebGL je najbolje podprt v mobilni različici brskalnika Firefox.

WebGL, za razliko od 2D platna, brez strojne podpore sploh ne deluje.

WebGL kompatibilnost ima svoje probleme tudi na namiznih računalnikih, saj na določenih operacijskih sistemih z določenim brskalniki še vseeno ne deluje. Problem je, da je še vedno veliko gonilnikov za grafične kartice na črni listi, ki ima delovanje privzeto izklopljeno. Na črni listi so gonilniki, ki po mnenju avtorjev brskalnikov še niso dovolj stabilni oziroma imajo pri prikazovanju WebGL vsebin probleme. Omejitve sicer lahko zaobidemo s postavitijo posebne zastavice ob zagonu brskalnika.

Razvijalci brskalnikov kot glavni razlog za slabo podprtost WebGLa navajajo probleme z varnostjo. Narediti peskovnik (sandbox) za spletno stran je zelo težko, če le ta za svoje delovanje potrebuje direkten dostop do grafične kartice.

Razvoj WebGL aplikacije

Razvoj aplikacij z uporabo WebGLa je tudi bolj zahtevno za programerja, kot razvoj aplikacij z 2D platnom. WebGL API namreč ne omogoča preprostih funkcij za risanje na zaslon in tudi za izris najbolj osnovnih oblik je potrebno nastaviti pravilen kontekst in napisati, prevesti ter povezati dva programa za senčilnike.

Senčilniki so programi, ki se izvajajo na grafični procesni enoti. WebGL definira dve vrsti senčilnih programov - vertex in fragment. Prvi skrbi za pozicijo vsakega ogljišča, ki ga izrišemo na na zaslonu, drugi pa za barvo vsakega fragmenta. Pisanje senčilnikov poteka v programskem jeziku GLSL, ki je podzvrst programskega jezika C. Nabor okazov, ki je na voljo, je v primerjavi s programskim jezikom ANSI-C sicer omejen, vendar imamo dodane posebne ukaze za lažje delo z vektorji in matrikami.

Podobno kot za 2D platno velja tudi za WebGL, aplikacijo razvijamo na namiznem računalniku in po potrebi preizkušamo kompatibilnost na mobilnih napravah.

Medplatformnost

V času pisanja WebGL še ni dobra izbira za medplatformni razvoj aplikacij. Vendar vse smernice kažejo v dejstvo, da se bo podpora v prihodnosti precej izboljšala. Dober indikator za to je tudi vključitev podpore v Internet Explorer 11, kljub dejstvu, da sta bila skupina Khronos in Microsoft v nenehni tekmovalnosti, Khronos s svojim OpenGL APIjem, Microsoft pa z lastnim DirectX sistemom.

Za razliko od zaprtih sistemov, kot je na primer Flash, ki mu podprtost pada (na iOSu ni bil podprt nikoli, na Androidu pa v zadnjih verzijah tudi uradno ni več podprt), je WebGL trenutno na dobri poti, da postane primerno orodje za razvoj medplatformnih aplikacij.

3.1.3 Zvok

Aplikacije napisane v 2D platnu in aplikacije pisane v WebGLu dostopajo do zvoka na enak način - to zato, ker je izrisovanje povsem ločeno od ostalih komponent.

HTML5 definira dokaj preprost API za predvajanje zvočnih datotek znotraj brskalnika. Programer ima na voljo ukaze za predvajanje zvoka, premikanje po zvočni datoteki in nastavljanja glasnosti zvočne datoteke, ne pa tudi kakšnih bolj naprednih ukazov kot spreminjanje frekvence...

Za bolj napredne funkcije je potrebno posežti po drugih metodah. Ena izmed najbolj uporabnih je uporaba Flash predvajalnika za predvanje zvoka. S tem pridobimo dodatne funkcije za delo z zvokom, vendar se zaradi vse slabše podpore Flash predvajalnikov na mobilnih podporah tudi lahko precej omejimo.

Uporabimo lahko tudi knjižnico, ki nam za predvanje zvoka ponudi svoj lasten API in potem zvoke predvaja na najboljši način glede na dano platformo, na kateri se potem aplikacija izvaja.

3.1.4 Zaznavanje vhoda

Grafično intenzivne aplikacije potrebujejo potrebujejo tudi procesirati vhod uporabnika. Za vhod je na namiznih računalnikih značilna kombinacija miška in tipkovnica. Na mobilnih napravah pa imamo navadno navoljo zgolj na dotik občutljiv zaslon. V javascriptu je napisanih kar nekaj knjižnic, ki nam pomagajo premostiti razlike med različnimi načini vnosa...

3.1.5 Primernost programskega jezika JavaScript

Za razvoj grafično zahtevnih aplikacij, je hitrost izvajanja programa bistvenega pomena. Za ta namen se praviloma uporablja statično tipizirane jezike in ročno sproščanje pomnilnika (garbage collection). V industriji se za te namene najbolj uporablja programski jezik C++.

Kljub vsem izboljšavam in pohitritvam, ki jih danes najdemo v modernih spletnih brskalnikih, je hitrost izvajanja programa napisanega v programskem jeziku Javascript, še vedno bistveno počasnejša. Razni testi kajžejo, da je ekvivalenten program napisan v programskem jeziku C++ lahko tudi do 5 krat hitrejši.

S pomočjo tehnologij, kot je ASM.js [link na del diplome o ASM.js] je možno Javascript program pohitriti, vendar je kljub temu povprečena hitrost izvajanja za dvakrat počasnejša od ekvivalentnega C++ programa.

Gledano izključno iz vidika hitrosti izvajnja, bo Javascript najbrž vedno manj primeren od klasičnih jezikov.

3.2 Spletne aplikacije z V8-gl

V8-gl je knjižnica, ki omogoča razvoj grafičnih aplikacij za namizne računalnike v jeziku Javascript. Knjižnica programerju nudi Javascript vmesnik do OpenGL APIja. Njen glavni cilj je narediti bogato orodje, ki bo olajšalo delo z 2D in 3D grafiko.

Knjižnica je trenutno še globoko v razvoju in zaenkrat stabilna verzija še ni bila izdana. OpenGL ES2.0 povezave so že delujoče in na voljo za uporabo. To pomeni, da lahko delujočo WebGL prenesemo na V8-gl in se znebimo odvisnosti od brskalnika. Še vedno veljajo enake omejitve s hitrostjo izvajanja, kot veljajo znotraj brskalnika, vendar nam ni potrebno več skrbeti glede delovanja v različnih brskalnikih.

Delo poteka tudi na prevedbi knjižnice tudi na sistema iOS in Android. To bi omogočilo bolj konsistentno medplatformno delovanje aplikacije po večih platformah, kot ga danes ponujajo različni spletni brskalniki.

3.2.1 LycheeJS

Najbolj perspektivna uporaba V8-gl knjižnice je trenutno v projektu LycheeJS. LycheeJS je pogon v Javascriptu, ki nam omogoča pisanje aplikacij za HTML5 platno, WebGL ali celo domoroden OpenGL ES. Aplikacijo lahko

tako objavimo kot spletno aplikacijo ali pa kot Android aplikacijo. Dela se tudi na podpori za iOS.

Z malo abstrakcije je tako možno spletno aplikacijo pretvoriti v namizno aplikacijo. Delo lahko poenostavi knjižnica LycheeJS, cilj njenega avtorja pa je razširiti V8-gl tudi na Android in iOS.

3.3 libgdx[3]

Libgdx je medplatformno ogrodje za razvoj grafičnih aplikacij, s poudarkom na računalniških igrah. Razvijalec svojo aplikacijo napiše v programskem jeziku Java, ogrodje pa potem poskrbi za izvoz napisane aplikacije v različne platforme. Podprte platforme so Windows, Linux in OSX ter Android in iOS, s pomočjo Google Web Toolkit (GWT) knjižnice pa je možen izvoz tudi za spletne brskalnike (WebGL).

3.4 playn[8]

3.5 Unity[6]

Unity je razvojno okolje, ki nam omogoča razvoj grafično intenzivnih aplikacij in izvoz za različne platforme. Za Windows, OSX in Linux je izvoz brezplačen, za iOS in Android pa je avtorjem potrebno plačati.

3.6 Parse[7]

Parse C# stuff

3.7 C++

C++ dostopen povsod

3.8 Programski jezik Haxe

3.9 Flash

Poglavje 4

Primeri

4.0.1 Primer MIN

Primer Min je bil narejen s pomočjo knjižnice LibGDX. Za svoje delovanje izrablja zmoglivosti OpenGL ES 2.0.

4.0.2 Primer DEFG

4.0.3 Primer C++

Poglavje 5

Sklepne ugotovitve

Sklep.

Poglavje 6

ASM.js

asm.js je raziskovalni projekt pri Mozilli, ki definira podmnožico JavaScripta. Cilj podmnožice, kot jo definira asm.js, je preprost način optimizacije izvajanja JavaScript kode znotraj brskalnikov.

C++ prevajalnika Emscripten in Mandreel že znata generirati JavaScript kodo, ki jo defenira asm.js. Patent, ki ga uporabljata Emscripten in Mandreel sta ponazarjanje spomina s samostojno instanco (singleton) tipiziranega spomina in uporaba bitnih operatorjev za spremenljivke, ki se obnašajo kot cela števila (integers) v C++.

Prevajanje v JavaScript ni nič novega. Leta 2006 je podjetje Google izdalo Google Web Toolkit (GWT), ki poleg drugih stvari, lahko tudi prevaja izvorno kodo iz programskega jezika Java v JavaScript. Od leta 2006 se je pojavilo kar nekaj podobnih prevajalnikov za že obsotječe programske jezike (C++, C#), kot tudi za nove jezike kot so na primer CoffeeScript, TypeScript in Dart.

Problem projektov kot je GWT je v tem, da ni standardne dokumentacije, ki bi izdelovalcem JavaScript pogonom omogočilo optimizacije. Zato je tudi na primer znano, da GWT aplikacije v brskalniku Google Chrome tečejo malce hitreje kot v drugih brskalnikih. Razlog je v tem, da sta tako GWT in Chrome razvita pod isto streho (Google) in je veliko več interne komunikacije, ki pa je ostali brskalniki niso deležni. asm.js dokumentira vse

možne pohitritve in navodila za pohitritve da na voljo vsem brskalnikom.

asm.js se izogiba potencialnih upočasnitev v kodi, saj nima spremenljivk z mešanimi tipi. Ime knjižnice izvira v dejstvu, da asm.js izvorna koda izvaja zgolj nizko nivojske izračune, ki so podobni tistim, ki jih izvajajo zbirniki. To pa je točno to, kar preveden C/C++ potrebuje.

Optimizacije v času izvajanja:

1. Tipi spremenljivk se pokažejo med preverjanjem tipov. To omogoča prevajanje v naprej (ahead of time) in ne samo ob pravem času (just in time).

2. JavaScript pogon ima garancijo, da se tipi spremenljivk med izvajanjem ne bodo spreminjali. S tem lahko pogon generira bolj preprosto in bolj učinkovito kodo.

3. Sistem tipov v asm.js olajša globalno strukturiranje programa (klici funkcij, dostop do spomina)

Izvorna koda, ki uporablja asm.js je še vedno dvakrat počasnejša od materne (native) kode napisane v nižje nivojskih jezikih kot je C, vendar se bo s časoma asm.js še dodatno pohitrila.

Ker je asm.js koda pod množica JavaScripta lahko že danes teče v vseh brskalnikih.

Kaj trenutno še ni podprto? C++ izjeme, setjmp/longjmp.

Popolnoma podpra sta trenutno samo C in C++, drugi jeziki so poprti le deloma in niso deležni enakih pohitritev in optimizaij.

Dinamični jeziki, kot so Python, Ruby in Lua, so še v zgodnjih stadijih razvoja in potrebujejo še veliko dela preden bodo uporabni.

Dodaten problem pri jezkih, kot sta Java in C# je v tem, da se veliko optimizacij naredi navidezni stroj na nivoju byte kode. Te optimizacije se izgubijo, če prevajalnik prevaja iz izvorne kode v izvorno kodo.

Edin način kako dobiti boljše pohitritve v teh jezikih je prevajanje celotnih navideznih strojev... To zgleda kot edini način kako izvajati večino jezikov s perfektno semantiko in maksimalno hitrostjo.

Literatura

[1] Canvas Element W3C - http://www.w3.org/wiki/HTML/Elements/canvas

[2] WebGL - Khronos Goup - http://www.khronos.org/webgl/

[3] libGDX - http://libgdx.badlogicgames.com/

[4] V8-gl izvorna koda - https://github.com/philogb/v8-gl

[5] LeechyJS - http://martens.ms/lycheeJS/

[6] Unity - http://unity3d.com/

[7] Parse - https://www.parse.com/

[8] PlayN - http://code.google.com/p/playn/

[9] licence-cc.pdf. Dostopno na: