

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

Rezultati diplomskega dela so intelektualna lastnina avtorja in Fakultete za računalništvo in informatiko Univerze v Ljubljani. Za objavlanje ali izkoriščanje rezultatov diplomskega dela je potrebno pisno soglasje avtorja, Fakultete za računalništvo in informatiko ter mentorja.¹

Besedilo je oblikovano z urejevalnikom besedil \LaTeX .

¹V dogovorju z mentorjem lahko kandidat diplomsko delo s pripadajočo izvirno 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	HTML5 (2D canvas)[1]	7
3.2	WebGL[2]	9
3.3	libgdx[3]	9
3.4	playn[8]	10
3.5	V8-gl[4] in LycheeJS[5]	10
3.6	Unity[6]	10
3.7	Parse[7]	10
3.8	C++	10
4	Primeri	11
5	Sklepne ugotovitve	13
6	ASM.js	15

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 operacijskega 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 Raspberry 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 naslednjih letih je podjetje Apple Inc. dodalo prvemu telefonu nekaj dodatnih funkcij kot so 3g povezljivost, izboljšana zadnja kamera, zaslon z višjo resolucijo, dvo jedrni 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 zmogljivosti itd.

Poglavje 3

Metode

3.1 HTML5 (2D canvas)[1]

Na področju spletnih tehnologij v obliki HTML5 je bilo v zadnjih letih narejeno veliko. S pojavom spletnega brskalnika Google Chrome se je začela prava tekma za hitrost. Spletne aplikacije, so za svoje delovanje začele uporabljati vedno več Javascripta in je bilo prvenstvenega pomena, da brskalnik lahko to kodo hitro izvede.

Brskalniki so začeli med seboj začeli tekmovati, kdo lahko hitrejši izvede Javascript kodo, rezultate 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 te tehnologije postale dostopne tudi na mobilnih platformah. Zaradi strojnih omejitev in predvsem življenske dobe baterij, so bili hitri in učinkoviti Javascript pogoni na mobilnih platformah mogoče še celo bolj pomembni kot na namiznih računalnikih.

Rezultat vsega tega je, da lahko danes tako na namiznih računalnikih kot tudi na mobilnih napravah izvajamo zahtevne aplikacije napisane v programskem jeziku Javascript.

Programski jezik Javascript je pravzaprav edina skupna točka pametnih telefonov, tablic in namiznih računalnikov. Operacijski sistemi na telefonih in tablicah uporabljajo različne programske jezike za razvoj domorodnih apli-

kacij, vendar pa imajo prav vsi tudi naložen takšen ali drugačen 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. To nam omogoča, da svojo spletno aplikacijo zapakiramo v ovitek (wrapper), ki se potem iz vidika končnega uporabnika obnaša kot vse ostale domorodne aplikacije. Izjema je samo FirefoxOS, ki spletne aplikacije že tako ali tako smatra kot domorodne.

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 tudi 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 vseh 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 *platno* ni primerno za izrisovanje kompleksnih grafik na mobilnih napravah.

Platno je namenjeno izrisovanje dvo dimenzionalnih oblik na zaslon. Programerju nudi preprost API za risanje raznih oblik (*drawRect*, *drawCircle*), risanje besedila (*fillText*), risanje poti (*moveTo*, *lineTo*), risanje slik (*drawImage*) in tudi risanje gradientov (*drawGradient*). Nudi pa 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

in se le malo ozira na delovanje na mobilnih platformah. Pri tem mora seveda biti pozoren na strojne omejitve mobilnih naprav.

Prednost uporabe dvodimenzionalnega platna je, da bo aplikacija delal v večini brskalnikov na večih platformah. Vendar pa je platno tudi dokaj omejeno s svojimi zmožnostmi. Izrisovanje zahtevnejših slik lahko postane počasno, izrisovanje v treh dimenzijah pa zaradi slabe podpore strojnemu pospeševanju skoarjda nemogoče.

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

Grafično intenzivne aplikacije 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...

3.2 WebGL[2]

WebGL je medplatformni API, uporabljen za delo s 3D grafiko znotraj spletnega brskalnika. Je kontekst *canvasa*, 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 Canvasu za splet". Do konca leta 2007 sta tako Mozilla in 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 specifikacije je bila izdana marca 2011.

Mozilla je imel 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 predogledu za razvijalce.

WebGL torej popravlja glavno pomankljivost *2D canvasa*. Določeni spletni brskalniki namreč WebGLa sploh ne podpirajo (Internet Explorer), podpore še nimajo omogočene (Safari na iOS) ali pa je podpora še v beta stanju (Chrome na Androidu).

WebGL uporablja API knjižnico OpenGL ES 2.0, ki je sicer dobro podprta na večini mobilnih naprav. Glavni problem WebGLa je varnost, saj je skorajda nemogoče narediti peskovnik (sandbox) za spletne strani, ki imajo direkten dostop do grafične kartice.

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 V8-gl[4] in LycheeJS[5]

V8-gl omogoča razvoj grafičnih aplikacij za namizne računalnike v jeziku Javascript. 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.

3.6 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.

3.7 Parse[7]

Parse C# stuff

3.8 C++

C++ dostopen povsod

Poglavje 4

Primeri

4.0.1 Primer MIN

Primer Min je bil narejen s pomočjo knjižnice LibGDX. Za svoje delovanje izrablja zmogljivosti 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 definira 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 matrne (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 optimizacij.

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 jezikih, 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 izvirne kode v izvirno 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 Group - <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. Dostupno na: