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 objavljanje 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.

[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

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.

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. Tudi majhni računalniki kot je omenjeni Raspberry Pi imajo namenski grafični procesor, ki omogoča izvajanje grafično bogatih aplikacij.

2.1 Namizni in prenosni računalniki

Največ svobode pri razvoju grafično intenzivnih aplikacij je na voljo na namiznih računalnikih. Ti imajo na voljo najboljšo možno strojno oprema in tudi operacijski sistemi so zreli in dovršeni, saj so se razvijali že več kot deset let. Namizni računalniki nimajo tako strogih omejitev z velikostjo, kot mobilne naprave. Prav nič presenetljivo ni torej, da so grafično intenzivne aplikacije na tej platformi najbolj domače.

Namizni računalniki so bili zelo dolgo edina platforma, ki je bila zmožna poganjati grafično intenzivne aplikacije. S pojavom namenskih grafičnih čipov (3Dfx), je prvič postalo mogoče razvijati aplikacije izrisovanje kompleksnejših grafičnih oblik na zaslonu. S pojavom grafičnih kartic se je zmanjšala obremenitev centralne procesne enote. Vse zahtevne operacije[1] so se sedaj izvajale na ločeni enoti.

Na namiznih računalnikih sta se v začetku 90ih let pojavili dve knjižnici za delo s 3D grafiko. OpenGL in DirectX. Obe ponujata programski vmesnik za komunikacijo z grafično procesno enoto, vendar je precejšna razlika med njima.

2.1.1 OpenGL

OpenGL je odprt standard, ki ga razvija skupina Khronos. Na voljo je na večini operacijskih sistemih (Windows, Mac OSX in Linux).

Tekom let so izšle 4 verzije. Podpora za programiranje grafičnega vhoda (shaderji) so postali podprti v verziji 2.0, ko je bila narejena tudi totalna prenova celotnega sistema. Iz te prenove se je nato razvila tudi mobilna inačica OpenGL ES 2.0.

OpenGL je implementiran v gonilniku za zaslon in vsak proizvajalec grafičnih kartic mora v gonilnike za grafično kartico dodati podporo. Problem s tem principom je, da se vmesnik nekoliko razlikuje med različnimi proizvajalci grafičnih kartic. Tudi programski jezik za pisanje shader programov in njihovi prevajalniki se lahko med seboj razlikujejo.

OpenGL se veliko uporablja za profesionalne aplikacije, kot so programi za grafično modeliranje (Maya) in programi za računalniške simulacije...

2.1.2 Direct3D

Direct3D je bil Microsoftov odgovor na OpenGL. Direct3D je zaprt programski vmesnik, popolnoma v lasti Microsofta. Programski vmesnik je uradno

[1] operacije s plavajočo vejico...

podprt samo na operacijskih sistemih Windows in na Microsoftovi igralni konzoli Xbox. Na drugih platformah je možno Direct3D aplikacije poganjati samo z uporabo posebne virtualizacijske plasti. Na Linuxu to virtualizacijsko plast ponuja orodje Wine, ki pa še vedno ne podpira Direct3D vmesnika v celoti.

Direct3D se ne uporablja toliko v profesionalnih aplikacijah kot OpenGL. Razlog je bil v slabši hitrosti izvajanja na začetku... Je pa toliko bolj popularen pri računalniških igrah.

2.2 RaspberryPi

RaspberryPI je majhen računalnik, velikosti kreditne kartice, ki je bil razvit za promocijo učenja računalniške znanosti. Računalnik vsebuje 700 MHz ARM procesor, 256 (ali 512) MB delovnega pomnilnika in grafično procesno enoto VideoCore IV s 250MHz, ki podpira tudi OpenGL ES2.0.

Raspberry Pi je zanimiva mešanica med mobilnimi in namiznimi računalniki. Po svoji strojni opremi je sicer zelo podoben mobilnim napravam, vendar na njem ne teče mobilni operacijski sistem. Na Raspberry Piju je mogoče poganjati operacijske sisteme, ki so značilni za namizne računalnike. Najbolj pogosto uporabljena je malce prirejena verzija Linux distribucije Debian, uradno možno pa je tudi naložiti distribuciji Arch in Fedora.

Čip za izrisovanje grafike na RaspberryPiju ima popolno podporo za OpenGL ES2.0 in je kljub omejitvam zmožen poganjati zahtevnejše aplikacije. Lep primer je računalniška igra Minecraft.

2.3 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.

2.3.1 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 in ostalih fizičnih gumbov - uporabnik je do vseh funkcij telefona dostopal preko na dotik občutljivega zaslona in enega fizičnega gumba.

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, planirana pa je tudi podpora OpenGL ES3.0.

Razvoj aplikacij poteka v jeziku Objective C (objektni C). Jezik temelji na ANSI Cju, vendar z dodano podporo objektno orientiranim konceptom. Prevajalnik za Objective C lahko prevede vsak program napisan v Cju. Objektno orientirani koncepti so implementirani s konceptom pošiljanja sporočil, slično programskemu jeziku SmallTalk. Za razliko od Jave, ki se uporablja na sistemih Android, Objective C nima avtomatičnega sproščanja pomnilnika, kar pri razvoju zahtevnih aplikacij lahko štejemo kot prednost, saj ima programer na voljo več orodij za delo s pomnilnikom.

Problem pri razvijanju iOS aplikacij je v tem, da je razvoj možen samo na strojni in programski opremi, ki jo proizvaja Apple. Tako za Android in Windows Phone je možno razvijati na konkurenčnih operacijskih sistemih.

2.3.2 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. Verzija 4.3 pa prinaša podporo tudi za OpenGL ES3.0. Podpora za OpenGL ES 2.0 na verziji 2.2 ni popolna in je potrebno napisati lasten C++ wrapper, ki omogoči funkcionalnosti, ki jo API ne podpira.

Programski jezik za razvoj domorodnih aplikacij na sistemu Android je Java, ki teče na virtualnem stroju Dalvik. Aplikacije napisane v Javi se prevedejo v bitno kodo (bytecode) in se nato iz JVM kompatibilnih .class datotek pretvorijo v .dex datoteke, ki so kompatibilne na Dalviku. Format .dex je namenjen sistemom, ki imajo omejeno količino pomnilnika in procesorske moči.

2.3.3 Windows

Mobilni operacijski sistem Windows 8 Phone (in RT), za dostop do grafične kartice uporabljata Microsoftovo knjižnico Direct3D. Za razliko od drugih mobilnih operacijskih sistemov, je Windows eden izmed redkih, ki ne podpira OpenGL ES programskega vmesnika. Izvajanje OpenGL ES aplikacij je tako mogoče zgolj s posebno virtualizacijo in uporabo orodij, kot je ANGLE.

Programski jezik, ki je uporabljen na Windows mobilnih sistemih je C#. C# je bil razvit pri Microsoftu, kot del njihove .NET iniciative. Ecma ga je priznala kot standard 4. julija 2006 (ECMA-334). Podobno kot ObjectiveC je bil tudi C# razvit z namenom dodajanja objektno orientiranih lastnosti v programski jezik C. Glavni razvijalec programskega jezika C# je Anders Hejlberg. C# nekoliko spominja na programski jezik Java, vendar se te dva

jezika v kasnejših verzijah kar precej razlikujeta. Lep primer je implementacija genericov, ki je v C# ustvarjena s pomočjo reifikacijo podatkov, v Javi pa s pomočjo posebne sintakse.

ANGLE

ANGLE: Almost Native Graphics Layer Engine Angle je odprtokodni projekt, ki implementira OpenGL ES2.0 specifikacijo in jo strojno pospeši z Direct3D. Angle je v uporabi kot primarno zaledje za WebGL v brskalnikih Chrome in Firfox. Podpira DirectX9 do DirectX11.

2.3.4 Firefox OS

Firefox OS je mobilni operacijski sistem, ki temelji na odprtih standardih spleta. Domorodne aplikacije pisane za Firefox OS so kar spletne aplikacije narejene po načelih HTML5, ki imajo posebne ovitke za klicanje sistemskih funkcij, kot je klicanje in dostop do senzorjev za lokacijo, pospeške in tako dalje.

Domorodne aplikacije gradimo z uporabo programskega jezika Javascript, za izgled in obliko aplikacij pa skrbita HTML5 in CSS. Aplikacije lahko z uporabo orodij v SDKju, preizkusimo tudi na mobilnih računalnikih.

Ker podpora temelji na standardih za spletne tehnologije ni podpore za OpenGL ES1.x, ki je na voljo na drugih mobilnih operacijskih sistemih. Je pa seveda podprt OpenGL ES 2.0 v obliki WebGLa.

Operacijski sistem Firefox OS sicer temelji na jedru Linux, ki služi kot platforma na katerem se nato zažene Gecko. Gecko je tako imenovani pogon za razporeditev, ki je prisoten v vseh verzijah brskalnika Firefox in pa tudi drugje. Ker Gecko deluje na različnih platformah, je Firefox OS možno naložiti tudi na druge naprave, tudi na RaspberryPi[2.2.5].

2.3.5 Ubuntu

Operacijski sistem Ubuntu je najbolj popularen na Linuxu temelječi operacijski sistem na namiznih računalnikih. Podjetje, ki razvija operacijski sistem Ubuntu ima vizijo spraviti podobno funkcionalnost tudi na mobilne naprave - telefone in tablice. Prva naprava, ki bo izšla z mobilnim operacijskim sistemom Ubuntu, je telefon Edge in je bila napovedana 22. julija 2013. Pred tem so radovedni razvijalci lahko naložili operacijski sistem na mobilni telefon Nexus 4, ki ima privzeto naložen operacijski sistem Android.

Canonical za razvoj aplikacij za mobilni operacijski sistem Ubuntu priporoča uporabo programskega jezika QML (Qt Meta Language). QML je programski jezik namenjen lažjemu razvoju uporabniškega vmesnika.

Dokumentacija za Ubuntu Phone še ni popolna, ampak kot se da trenutno razbrati je uporaba OpenGLa podprta s strani qt 3d iniciative. Za pisanje grafično intenzivnih aplikacij se tako namesto QMLa priporoča C++.

O samih zmogljivostih telefona in tablic se sicer še ne ve veliko, pričakuje pa se, da bo podprt vsaj standard za OpenGL ES2.0 aplikacije.

2.4 Zmogljivosti

Kot smo videli pri posameznih primerih ima večina mobilnih naprav podporo za OpenGL ES2.0, tako da je glavni problem pri razvoju medplatformnih aplikacijah premagati omejitve pri domorodnih jezikih. Večina mobilnih operacijskih sistemih ima svoj programski jezik, ki ga preferira za pisanje aplikacij.

Večina izmed njih ima tudi na voljo tako imenovani NDK native development kit oz. po slovensko domorodni nabor orodij za razvijanje aplikacij. NDK normalno podpira jezik C++, ki je na voljo na večini operacijskih sistemov. Tudi

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, 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 spletne tehnologije postale dostopne tudi na mobilnih platformah. Zaradi strojnih omejitev in predvsem življenjske dobe baterij, so bili hitri in učinkoviti Javascript pogoni na mobilnih 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 tako imenovanega 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 pomanjkljivostim lahko s pomočjo platna na preprost način napišemo grafično intenzivno aplikacijo, ki bo delala na več platformah.

Strojno pospeševanje

2D *platno* na določenih konfiguracijah omogoča tudi strojno pospeševanje. Z omogočenim 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 specifikacije 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 postavitvijo 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 oglišča, 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 ukazov, 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 posesti po drugih metodah. Ena izmed najbolj uporabnih je uporaba Flash predvajalnika za predvajanje 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 predvajanje 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 na voljo 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 povprečna hitrost izvajanja za dvakrat počasnejša od ekvivalentnega C++ programa.

Gledano izključno iz vidika hitrosti izvajanja, 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 več platformah, kot ga danes ponujajo različni spletni brskalniki.

3.2.1 LycheeJS

Najbolj perspektivna uporaba V8-gl knjižnice je trenutno projekt LycheeJS. LycheeJS je pogon v Javascriptu, ki teoretično lahko teče na vseh okoljih kjer je na voljo Javascript. LeechJS podpira vse moderne brskalnike na namizju

(Firefox, Chrome, Opera, Safari in Internet Explorer) in tudi na mobilnih brskalnikih (WebKit, Firefox, Chrome na Androidu in Mobile Safari).

Ker se LycheeJS zanaša na V8-gl, veljajo enake omejitve pri uporabi OpenGL ES2.0 APIja kot tudi pri V8-gl. LycheeJS je zgolj ogrodje zgrajeno nad V8-glom, ki programerju omogoči lažje delo na svoji aplikaciji. Med prednostmi, ki jih LycheeJS prinaša so enoten vmesnik za detekcijo vhoda uporabnikov, kot tudi enoten vmesnik za predvajanje zvočnih datotek. Poleg tega ima LycheeJS vgrajena tudi orodja za pakiranje aplikacij, tako da brez večjega truda svojo spletno aplikacijo zapakiramo za različne platforme. Trenutno podprte platforme so spletne aplikacije, vtičniki za brskalnik Google Chrome ter Android aplikacija.

3.3 xamarin

Xamarin nam omogoča, da našo aplikacijo napišemo v programskem jeziku C#, do APIjev domorodnih platform pa dostopamo. C# prevajalnik našemu programu doda .NET runtime (Mono) in proizvede izvedljiv ARM program, ki je lahko zapakiran kot iOS aplikacija ali pa kot Android aplikacija.

Na ta način lahko Android in iOS aplikacije delijo izvorno kodo.

Xamarin kodo prevede v domorodno izvedljivo binarno datoteko za posamezno platformo. Izvajanje te domorodne je hitro in ni vidnih vplivov na hitrost izvajanja. Koda nam sicer prinese dodatnih 2.5MB podpisa...

Kako točno Xamarin C# prevede v domorodno aplikacijo ni dostopno, saj je to del njihovega zaprtega sistema.

Xamarin temelji na odprtokodni verziji .NET ogrodja - Mono, ki deluje na večini platform, ki so v uporabi danes (Linux, Unix, FreeBSD, MacOSX). Za iOS Xamarinov lasten AOT (ahead of time) prevajalnik prevede C# kodo v ARM zbirni jezik. Na operacijskem sistemu Android pa xamarinov prevajalnik prevede kodo v vmesni jezik (IL), ki se nato prevede ob pravem času (JIT), ko se aplikacija zažene. V obeh primerih Xamarin poskrbi za alociranje spomina, memory allocation, garbage collection, underlying platform

interop, etc.

3.4 libgdx[3]

Libgdx je ogrodje za medplatformni razvoj grafičnih aplikacij napisano v Javi. Knjižnica abstrahira razlike med namiznimi aplikacijami, Androidom, iOSom in HTML5 ter gradi na odprtih standardih, kot je OpenGL ES/WebGL.

LibGDX omogoča hitro prototipiranje, saj je razvoj mobilnih aplikacij možen na namizju. V pravilnost delovanja aplikacije se lahko prepričamo iz namiznega računalnika in šele nato zgradimo paket.

Prednost tega pristopa so krajši nalagalni časi aplikacije. Grajenje iOS ali Android paketa traja nekaj časa (Dex, package), potem pa je potrebno aplikacijo preko USB kabla prenesti na dejansko napravo. Razvoj grafično intenzivnih aplikacij je z uporabo Android emulatorja na primer skorajda nemogoče, saj le ta niti ne podpira OpenGL ES2.0. Vsi ti koraki na sodobnem računalniku sicer ne trajajo več kot eno minuto, ampak v primerjavi z poganjanjem .jar datoteke iz namizja (manj kot 5 sekund) prednost razvoja iz namizja precej zniža čas, ki je potreben za razvoj aplikacije.

Poleg hitrejšega mrzlega zagona aplikacije je razvoj na namizju hitrejši tudi zaradi vročega izmenjevanje kode (code hot swapping). Vroče izmenjevanje kode je proces, ko del kode, ki teče na JVM zamenjamo z novim delom kode med tem ko aplikacija teče. S tem se izognemo ponovnemu zaganjanju aplikacije. Dodatna prednost je, da nam ni potrebno na ponovno nastavljati stanja, v katerem se je aplikacija nahajala preden smo naredili spremembo kode. Vroče izmenjevanje je instantno in programer dobi takojšen odziv na spremembe, ki jih je naredil, kar precej izboljša čas, ki je potreben za razvoj aplikacije.

Določeni deli ogrodja so bili napisani s pomočjo JNI (Java Native Interface, Javin domoroden vmesnik).

LibGDX omogoča razvoj aplikacij za Widows, Linux, Mac OS X, Android 1.5+, iOS, Java Applet, Javascript /WebGL.

TODO: Za razvoj iOS je potrebna Xamarin. LibGDX s pomočjo LLVM Java kodo spremeni v C# kompatibilno. Ta C# koda pa potem xamarin prevajalnik prevede v domorodno ARM kodo.

Projekt LibGDX je odprtokoden in dostopen na internetu.

3.5 playn[8]

PlayN je ogrodje za razvoj grafično intenzivnih aplikacij na večih platformah. Podprte platforme so namizni računalniki (Java), iOS, Android, HTML5 in Flash.

Vmesnik je napisan v programskem jeziku Java, na voljo pa imamo vse prednosti, ki smo jih omenili že pri uporabi LibGDX. Našo aplikacijo lahko razvijamo na namiznem računalniku, kjer lahko uporabljamo tudi vroče izmenjavanje kode.

Za izvoz na različne platforme lahko uporabimo ali orodje Ant ali pa Maven. Večjih razlik med obema orodjema ni, oba pa omogočata preprost način grajenje domorodnih paketov za Android.

3.5.1 GWT

Izvoz aplikacije v HTML5 je mogoč z uporabo orodja GWT (Google Web Toolkit), ki je kot navaja tudi ime orodja, plod dela zaposlenih pri podjetju Google. GWT je razvijalsko orodje, za grajenje kompleksnih aplikacij, ki tečejo v brskalniku. Celoten paket vsebuje Java API knjižnico, prevajalnik in strežnik za razvijanje aplikacije.

GWT program napisan v jeziku Java prevede v optimiziran Javascript. Prevajalnik upošteva prednosti in slabosti posameznih brskalnikov in optimizira za vsak brskalnik posebej. Poleg brskalnikov za namizje je prevedeni Javascript optimiziran tudi za mobilne brskalnike, tako da aplikacija napisana s pomočjo GWTja deluje hitreje tudi na mobilnih napravah.

Poleg specifičnih optimizacij za različne brskalnike prevajalnik tudi odstrani mrtvo kodo, optimizira Stringe, in-lining methods.

3.5.2 iOS

Tudi grajenje iOS paketa poteka na podoben način kot pri LibGDX, z uporabo Xamarin licence.

3.5.3 Primerjava z LibGDX

Razlika med LibGDX in PlayN se pokaže predvsem v uporabi vmesnikov. PlayN podpira nekoliko več platform in 2D API je nekoliko lažji za uporabo. Delo z 3D apijem (in dostop do domorodnih OpenGL ES funkcij) pa je precej lažje z uporabo LibGDX knjižnice.

Projekt je odprtokoden razvijajo pa ga zaposleni pri Googlu.

3.6 Unity[6]

Unity3D je orodje za izdelovanje medplatformnih grafično intenzivnih aplikacij. Poleg mobilnih naprav (iOS, Android, BlackBerry, WinPhone 8) ter vseh glavnih namiznih operacijskih sistemov (Windows, MacOS, Linux), orodje omogoča izdelovanje aplikacij tudi za igralne konzole (Xbox is PS3). Orodje je sestavljeno iz dveh glavnih delov - engine in integrirano okolje za razvijanje.

3.6.1 pogon

Za risanje na zaslon Unity uporablja grafični vmesnik Direct3D (na platformi Windows in Xbox), OpenGL ES (iOS, Android).

3.6.2 integrirano okolje za razvijanje

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

3.9 Programski jezik Haxe

3.10 Ogre

!!!

3.11 Marmelade

3.12 Monogame

3.13 Flash

3.14 QT

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: