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.

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!

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:

Zahvala

Posvetilo

Kazalo

Povzetek

V diplomskem delu smo naredili pregled metod za razvoj grafično intenzivnih aplikacij na različnih platformah. Raziskali smo možne platforme in si ogledali razlike med njimi. Poseben poudarek smo namenili mobilnim napravam. V nadaljevanju diplomskega dela smo si ogledali metode in orodja s katerimi je možno razlike med platformami premostiti. Štiri metode smo opredelili bolj podrobno in za vsako od teh razvili testno aplikacijo. Raziskali smo tudi računanje na grafičnih procesnih enotah, saj lahko postane pomemben faktor pri razvoju medplatformnih aplikacij v prihodnosti. Ugotovili smo, da uporaba obravnavanih metod pohitri in poenostavi razvoj medplatformnih aplikacij.

Abstract

We have investigated methods of developing cross-platform graphics-intense applications. We have researched different platforms and pointed out differences between them. Mobile platforms have received special attention. We have taken a look into methods and tools for overcoming differences between the platforms. We studied four methods more thoroughly and implemented test applications for each of those. Additionaly, we have researched computing on graphics processing units as it may become an important factor for developing cross-platform applications in the future. We concluded that we can develop cross-platform applications faster and easier by exerting the discussed methods.

Poglavje 1

Uvod

Še pred nekaj leti je bilo poganjanje grafično intenzivnih aplikacij domena dragih delovnih postaj. Danes je tega zmožen vsak osebni računalnik in tudi naprave, ki jih prenašamo v žepih.

Zmogljive mobilne naprave 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 domorodnih aplikacij pa tudi pri knjižnicah se pojavljajo razlike (npr. OpenGL ES, Direct3D). 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.

1.1 Potek diplomske naloge

Cilj diplomske naloge je pregledati in preizkusiti možnosti za premostitev razlik med različnimi platformami. Na začetku si bomo ogledali različne platforme, njihove razlike in skupne točke. Osredotočili se bomo na mobilne platforme, vendar se bomo dotaknili tudi namiznih računalnikov in računalnikov na eni sami ploščici.

V nadaljevanju diplomske naloge si bomo ogledali nekatera odprtokodna in plačljiva orodja za premostitev razlik med platformami. Preučili bomo spletne tehnologije in orodja, ki temeljijo na prevodih med različnimi programskimi jeziki. Zanimale nas bodo metode, ki temeljijo na uporabi jezika C++, ter orodja, ki ponujajo celotno razvojno okolje. Ogledali si bomo tudi orodje, ki uporablja namenski programski jezik.

Ena izmed glavnih ovir pri razvoju grafično intenzivnih aplikacij s pomočjo spletnih tehnologij je hitrost izvajanja programa znotraj spletnega brskalnika. Zato bomo raziskali načine za pospešitev Javascript izvorne kode.

Grafično intenzivne aplikacije za tekoče delovanje uporabljajo vedno višje stopnje paralelnosti. S tem razlogom si bomo ogledali procesiranje na grafičnih procesnih enotah.

V zadnjem delu diplomske naloge bomo opisali nekaj primerov grafično intenzivnih aplikacij, ki so bile razvite z namenom pokazati prednosti in slabosti uporabljenih orodij. Razložili bomo uporabljene metode, njihove prednosti in slabosti.

Poglavje 2

Pregled platform za izvajanje grafično intenzivnih aplikacij

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

Omeniti moramo razliko med večplatformnim in medplatformnim razvojem aplikacij. Ker nismo našli točnih definicij teh dveh pojmov, se opiramo na sorodna pojma medjezično in večjezično (npr. iskanje). Medjezično iskanje je iskanje, pri katerem je naravni jezik iskalne zahteve lahko različen od jezika ali jezikov, v katerih je izražena vsebina dokumentov [3]. Podobno je lahko programski jezik, v katerem je napisana medplatformna aplikacija, lahko različen od programskih jezikov, ki jih uporabljajo ciljne platforme. Večplatformna aplikacija je širši pojem, ki zajema tudi medplatformne aplikacije, hkrati pa v to skupino spadajo aplikacije, ki so bile pisane v programskih jezikih posamezne ciljne platforme.

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 opremo in tudi operacijski sistemi so zreli in dovršeni, saj so v razvoju že več deset let. Namizni računalniki nimajo tako strogih omejitev z velikostjo, kot mobilne naprave. Prav nič presenetljivo torej, da so grafično intenzivne aplikacije na tej platformi najbolj domače.

Namizni računalniki uporabljajo drugačno arhitekturo centralne procesne enote, kot večina mobilnih naprav. Procesorji v namiznih računalnikih so danes večinoma zgrajeni s CISC arhitekturo (Intel, AMD), medtem ko mobilne naprave večinoma uporabljajo RISC arhitekturo in procesorje podjetij ARM in nVidia. Seveda obstajajo tudi izjeme, kot so na primer tablice s procesorji Intel Atom in PowerPC namizni računalniki.

Namizni računalniki so bili zelo dolgo edina platforma, na kateri je bilo možno poganjati grafično intenzivne aplikacije. S pojavom grafičnih kartic se je zmanjšala obremenitev centralne procesne enote. Večji del operacij povezanih z izrisovanjem objektov na zaslonu, se danes izvaja na grafičnih procesnih enotah, ki imajo za ta namen visoko stopnjo paralelizma.

Prenosni računalniki so po zmogljivostim podobni namiznim računalnikom. Za centralno procesno enoto uporabljajo tudi enako arhitekturo. Grafične procesne enote se na prenosnih računalnikih navadno sicer šibkejše kot na namiznih računalnikih, vendar so razlike minimalne. Razlog je v kompromisu med zmogljivostjo grafične kartice, porabo energije in prostorski omejenosti.

Na namiznih in kasneje na prenosnih računalnikih sta se v začetku 90. let pojavili dve knjižnici za delo s 3D grafiko: OpenGL in Direct3D. Obe ponujata programski vmesnik za komunikacijo z grafično procesno enoto, vendar so razlike med njima precejšne.

2.1.1 OpenGL

OpenGL je odprt standard, ki ga razvija skupina Khronos [1]. Na voljo je na večini operacijskih sistemov (Windows, Mac OSX in Linux).

Programski vmesnik se je na začetku uporabljal predvsem za profesionalne aplikacije, kot je na primer AutoCAD in različne simulacije. Šele kasneje se je programski vmesnik razvil do te mere, da je bil uporaben tudi za druge namene.

Tekom let so izšle štiri večje posodobitve in kar nekaj manjših revizij. V času pisanja je najnovejša različica OpenGL 4.4. Podpora za programiranje grafičnega vhoda se je prvič pojavila pojavila v posodobitvi 3.0. Pred tem se je za izrisovanje uporabljal fiksen funkcijski cevovod. Prednost programiranega grafičnega vhoda je predvsem v večji fleksibilnosti, vendar za ceno večje kompleksnosti. OpenGL za vgrajene sisteme (OpenGL ES) je v prvi verziji uporabljal fiksen funkcijski cevovod, v reviziji 2.0 pa ga je zamenjal programiran vhod.

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 jeziki za pisanje programov na grafični kartici in njihovi prevajalniki so si med seboj lahko malce različni.

Dolgo časa so obstajale tudi razširitve, ki so bile na voljo samo na določenih grafičnih karticah. Proizvajalci grafičnih kartic so na ta način želeli izkazati superiornost, saj so te razširitve navadno dodale dodatne efekte pri prikazovanju. To je ustvarilo resen problem in razvijalci grafičnih aplikacij so morali prilagajati kodo aplikacij glede na specifikacijo posameznih grafičnih kartic.

2.1.2 Direct3D

Direct3D [2] je bil Microsoftov odgovor na OpenGL. Direct3D je zaprt vmesnik API, popolnoma v lasti Microsofta. Vmesnik je uradno podprt samo na operacijskih sistemih Windows. Nekoliko spremenjena oblika se nahaja

tudi na Microsoftovi igralni konzoli Xbox. Na drugih platformah je možno Direct3D aplikacije poganjati samo z uporabo virtualizacijske plasti. Na Linuxu to virtualizacijsko plast ponuja orodje Wine, vendar ne podpira Direct3D vmesnika v celoti.

Direct3D se ne uporablja v profesionalnih aplikacijah tako pogosto kot OpenGL. Razlogov je več. Na začetku je bil OpenGL vmesnik hitrejši in bolj natančen pri izrisovanju. Ker se je OpenGL pojavil veliko pred Direct3Djem je v tem času postal že standard v grafični industriji. Direct3D je bil za razliko od OpenGL, namenjen v prvi vrsti za osebne računalnike in ne samo za drage delovne postaje. Zaradi fiksnega cevovoda za izrisovanje je onemogočil proizvajalcem grafičnih kartic ustvarjanje lastnih modulov, ki bi otežili razvijanje aplikacij. Programski vmesnik je zaradi teh razlogov postal zelo popularen pri razvijalcih računalniških iger.

2.2 Mobilne platforme

Na trgu najdemo pester nabor mobilnih platform. Največji igralci v času pisanja diplomske naloge so Google s svojim odprto kodnim sistemom Android, Apple s svojim sistemom iOS, nekaj tržnega deleža pa imata tudi podjetji Microsoft, z mobilno različico operacijskega sistema Windows (Windows 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 svojo rešitev - Firefox OS, ki temelji na spletnih tehnologijah. Podjetje Canonical pripravlja različico Ubuntu operacijskega sistema za mobilne naprave, na Finskem pa podjetje Jolla Mobile razvija svoj lastni operacijski sistem Sailfish OS, ki tudi temelji na jedru Linux.

2.2.1 Android

Android [4] je operacijski sistem, ki temelji na Linux jedru. Operacijski sistem je razvilo podjetje Android Inc., s finančno pomočjo Googla. Le ta 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.

Jezik za domorodne aplikacije

Programski jezik za razvoj domorodnih aplikacij na sistemu Android je Java, ki teče na virtualnem stroju Dalvik. Aplikacije napisane v programskem jeziku Java se prevedejo v bitno kodo (angl. bytecode) in se nato iz JVM kompatibilnih *.class* datotek pretvorijo v *.dex* datoteke, ki jih Dalvik lahko poganja. Format *.dex* je prilagojen sistemom, ki imajo omejeno količino pomnilnika in procesorske moči.

Podprost grafičnih knjižnic

Android podpira OpenGL ES 1.1 in 2.0 od verzije 2.2 dalje. Verzija 4.3 prinaša podporo tudi za OpenGL ES 3.0. Podpora za OpenGL ES 2.0 na verziji 2.2 ni popolna, saj je za pravilno delovanje potrebno napisati lasten vmesnik v jeziku C++, ki omogoči funkcionalnost, ki jo vmesnik API ne podpira.

2.2.2 iOS

Prvi iPhone z operacijskim sistemom iOS [5] je bil predstavljen 9. januarja 2007. Od ostalih mobilnih naprav na tržišču se je razlikoval z dodelanim uporabniškim vmesnikom in zaslonom občutljivim na večprstne dotike. Bil je tudi eden izmed prvih mobilnih telefonov brez tipkovnice in fizičnih gumbov - 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, dvojederni procesor itd.

Posebnost pri razvijanju iOS aplikacij je v tem, da je razvoj možen samo na strojni in programski opremi, ki jo proizvaja Apple.

Jezik za domorodne aplikacije

Razvoj aplikacij poteka v jeziku ObjectiveC. Jezik temelji na ANSI Cju, vendar z dodano podporo objektno orientiranim konceptom. Prevajalnik za ObjectiveC 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, ObjectiveC nima avtomatičnega sproščanja pomnilnika, kar pri razvoju zahtevnih aplikacij lahko štejemo kot prednost, saj ima programer na voljo več nadzora nad pomnilnikom.

Podprtost grafičnih knjižnic

Naprave z operacijskim sistemom iOS danes podpirajo OpenGL ES 1.1 in 2.0. V zadnji verziji operacijskega sistema iOS 7 pa bo podprta tudi verzija OpenGL ES 3.0.

2.2.3 Windows

Microsoft ima za mobilne naprave dve različici operacijskega sistema Windows. Windows Phone [6] je namenjen za mobilne telefone, Windows RT pa za tablične računalnike. Prva naprava z operacijskim sistemom Windows Phone se je na trgu pojavila oktobra 2010.

Jezik za domorodne aplikacije

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. C# nekoliko spominja na programski jezik Java, vendar se ta dva jezika v kasnejših verzijah kar precej razlikujeta. Lep primer je implementacija tako imenovanih generikov, ki so v C# ustvarjena s pomočjo reifikacije podatkov, v Javi pa s pomočjo posebne sintakse.

Podprtost grafičnih knjižnic

Mobilni operacijski sistem Windows, za dostop do grafične kartice uporablja Microsoftovo knjižnico Direct3D.

Za razliko od drugih mobilnih operacijskih sistemov, je Windows eden izmed redkih, ki ne podpira OpenGL ES vmesnika. Izvajanje OpenGL ES aplikacij je tako mogoče zgolj s posebno virtualizacijo in uporabo orodij, kot je ANGLE.

ANGLE

Angle [7] je kratica, ki pomeni Almost Native Graphics Layer Engine, pogon s skoraj domorodno grafično plastjo.

Angle je odprtokodni projekt, ki implementira OpenGL ES 2.0 specifikacijo in jo strojno pospeši z Direct3D vmesnikom. Uporablja se kot primarno zaledje za WebGL v brskalnikih Chrome in Firfox na platformi Windows. Podpira verzije od DirectX9 do DirectX 11.

2.2.4 Firefox OS

Firefox OS [8] je mobilni operacijski sistem, ki temelji na odprtih standardih spleta. Domorodne aplikacije pisane za Firefox OS so kar spletne aplikacije narejene po načelih HTML5. Firefox OS standardu HTML5 doda posebne sistemske funkcije, s katerimi je mogoč dostop do funkcionalnosti telefona, kot je klicanje in dostop do senzorjev za lokacijo, pospeške itd.

Operacijski sistem Firefox OS temelji na jedru Linux, ki služi kot platforma na katerem se zažene Gecko. Gecko je pogon za razporeditev, ki je prisoten v vseh verzijah brskalnika Firefox. Ker Gecko deluje na različnih platformah, je Firefox OS možno naložiti tudi na druge naprave, tudi na RaspberryPi (razdelek 2.3.1).

Jezik za domorodne aplikacije

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 namiznih računalnikih.

Podprtost grafičnih knjižnic

Ker podpora temelji na standardih za spletne tehnologije ni podpore za OpenGL ES 1.x, ki je na voljo na drugih mobilnih operacijskih sistemih. Podprt je samo OpenGL ES 2.0 v obliki WebGLa.

2.2.5 Ubuntu

Operacijski sistem Ubuntu [9] je najbolj popularen na GNU/Linuxu temelječi operacijski sistem na namiznih računalnikih. Podjetje Canonical, ki razvija operacijski sistem Ubuntu, ima vizijo spraviti podobno funkcionalnost tudi na mobilne naprave - telefone in tablice.

Jezik za domorodne aplikacije

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

Za aplikacije, kjer je hitrost izvajanja ključnega pomena, je na voljo uporaba programskega jezika C++. Za pisanje grafično intenzivnih aplikacij se namesto jezika QML priporoča C++.

Podprtost grafičnih knjižnic

O samih zmogljivostih telefona in tablic se sicer še ne ve veliko. Edine naprave, ki so v času pisanja kompatibilne z mobilnim operacijskim sistemom Ubuntu, so Nexus 4, 7 in 11. Vse te naprave imajo primarno naložen sistem Android, vendar jih je možno odkleniti in namestiti Ubuntu. Vse imajo podporo za OpenGL ES 1.1 in 2.0, zato se pričakuje, da bodo ti podprti tudi znotraj sistema Ubuntu.

2.2.6 Sailfish OS

Sailfish OS [10] je operacijski sistem temelječ na Linux jedru namenjen mobilnim telefonom in drugim napravam. Razvija ga finsko podjetje Jolla, ki se ukvarja z oblikovanjem, razvijanjem in prodajanjem pametnih telefonov.

Sailfish OS temelji na Mer projektu, ki je nadaljevanje projekta MeeGo. MeeGo je bil operacijski sistem namenjen cenejšim prenosnikom, tabličnim računalnikom, mobilnim telefonom, pametnim televizijam ter drugim vgrajenim sistemom.

Jezik za domorodne aplikacije

Podobno kot operacijski sistem Ubuntu tudi Sailfish OS za razvoj domorodnih aplikacij uporablja QML in Qt (razdelek 3.11). Tudi tu se za razvoj grafično intenzivnih aplikacij priporoča programski jezik C++.

Podprtost grafičnih knjižnic

Dostop do OpenGL programskega vmesnika je mogoč s pomočjo QT pogleda. Uporabljena verzija je OpenGL ES 2.0.

2.3 Računalniki na eni plošči

2.3.1 Raspberry Pi

Raspberry PI je majhen računalnik, velikosti kreditne kartice, ki je bil razvit za promocijo učenja računalniške znanosti [11]. Računalnik vsebuje 700 MHz ARM procesor, 256 (ali 512) MB delovnega pomnilnika in grafično procesno enoto VideoCore IV s 250MHz, ki podpira OpenGL ES 2.0.

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 Pi je mogoče naložiti operacijske sisteme, ki so značilni za namizne računalnike. Najbolj pogosto uporabljena je malce prirejena verzija Linux distribucije Debian, uradno možno pa je naložiti tudi distribuciji Arch Linux in Fedora.

Čip za izrisovanje grafike na Raspberry Piju ima popolno podporo za OpenGL ES 2.0 in je kljub omejenem pomnilnikom zmožen poganjati zahtevnejše aplikacije.

2.3.2 BeagleBone

Podobno kot Raspberry Pi je tudi BeagleBone [12] majhen računalnik, ki je sposoben poganjati Linux sistem. BeagleBone vsebuje ARM procesor z 720MHz in 256 MB RAMa. Obstajajo tudi bolj zmogljive verzije, ki vsebujejo procesor s frekvenco do 1GHz in 512MB RAMa (BeagleBone Black).

Enota za grafično procesiranje PowerVR je zmožna poganjati 3D aplikacije z uporabo vmesnikov OpenGL ES 1.x ali OpenGL ES 2.0.

2.4 Skupne zmogljivosti

Kot lahko vidimo iz navedenih primerov platform, se ne moremo odločiti za en sam programski jezik in vmesnik API, s katerima bi pokrili vse možne platforme. Največji del trga lahko pokrijemo, če se odločimo za programski

vmesnik OpenGL ES 2.0 s programskim jezikom C++. S tem lahko razvijamo grafično intenzivne aplikacije na Wndows, Linux in Mac OSX, kot tudi na mobilnih platformah Android, iOS, Ubuntu, Sailfish in BlackBerry.

Kot smo videli pri posameznih primerih ima večina mobilnih naprav podporo za OpenGL ES 2.0, tako da je glavni problem pri razvoju medplatformnih aplikacijah premagati omejenost na programske jezike za določeno platformo. Večina mobilnih operacijskih sistemov ima svoj programski jezik, ki je v uporabi za pisanje domorodnih aplikacij. Načinov premagovanja teh ovir je več in nekaj si jih bomo ogledali v naslednjem poglavju.

Na večini naprav je tudi na voljo spletni brskalnik. Spletne aplikacije, ki tečejo znotraj brskalnika, za izvajanje uporabljajo programski jezik Javascript, ki je tako eden redkih jezikov, ki je na voljo na vseh napravah. Spletne aplikacije so lahko tudi grafično intenzivne in za svoje delovanje lahko celo uporabljajo dostop do grafičnega procesorja.

Poglavje 3

Metode za razvoj grafično intenzivnih medplatformnih aplikacij

Ogledali si bomo različne metode in orodja, ki so na voljo za razvoj grafično intenzivnih medplatformnih aplikacij. Kot prvi primer pisanja medplatformnih aplikacij bomo obravnavali spletne aplikacije, saj so spletni brskalniki prisotni na večini platform. Ogledali si bomo tudi kako spletne aplikacije izvajati kot namizno aplikacijo brez brskalnika. Preučili bomo, kako s programskim jezikom Java in orodjem Xamarin razvijamo medplatformne aplikacije. Zanimalo nas bo tudi orodje Unity, ki je v času pisanja zelo popularno. Dotaknili se bomo tudi programskega jezika Haxe, ki je bil ustvarjen z razlogom, da bi olajšal razvoj medplatformnih aplikacij. Na koncu poglavja si bomo ogledali še razvoj grafično intenzivnih aplikacij v programskih jezikih C++, C# in ActionScript.

3.1 Spletne aplikacije

Programski jezik Javascript je nastal okoli leta 1995, ko je bil tudi vključen v spletni brskalnik Netscape. Jezik Javascript je omogočil izvedbo kode na

uporabnikovi strani (angl. client-side) in spletne strani so postale interaktivne.

V zgodnjih dneh spleta se programiranje na uporabnikovi strani ni veliko uporabljalo. Javscript interpreterji so bili počasni, zaradi neenotnih standardov med brskalniki pa je bilo pisanje aplikacije, ki bi delala enako na vseh, zelo težavno. Šele kasneje, ko so brskalniki malo napredovali, je postalo pisanje aplikacij na strani uporabnika dejanska opcija.

S pojavom spletnega brskalnika Google Chrome se je začela nova doba brskalnikov in bogatih aplikacij na strani uporabnika. Google Chrome je začel pravo tekmo 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 tekmovati, kdo lahko hitrejše izvaja Javascript izvorno kodo. Rezultat 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. 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 implemen-

tacijo tako imenovanega elementa za spletni pogled (angl. webview). Ta element nam omogoča prikaz določene spletne strani znotraj domorodne aplikacije. Na ta način lahko spletno aplikacijo zapakiramo v ovitek, 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 zato ne potrebuje posebnega ovitka.

3.1.1 2D platno

Sodobni brskalniki nam za izris svojih oblik na zaslon poleg HTMLja in CSSa ponujajo tudi uporabo platna (angl. canvas) [13].

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 platnu 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 povsod. Zaradi slabe strojne podpore obstajajo omejitve kompleksnosti, ki jih s platnom lahko dosežemo.

Platno je namenjeno izrisovanju dvodimenzionalnih oblik na zaslon. Programerju nudi preprost vmesnik 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 ($rotate$).

Programer lahko aplikacijo s platnom razvija in testira na namiznem računalniku. Vmesne rezultate dela lahko preverja v svojem spletnem brskalniku in šele nato, ko se prepriča v pravilno delovanje, prenese aplikacijo

na mobilno napravo. Ta način dela zelo pohitri razvoj, saj prenos aplikacije na mobilno napravo traja več časa. Pri razvijanju aplikacije na namiznem računalniku mora biti programer še posebej 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 dobre 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

Dvodimenzionalno 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 na voljo povsod 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 omogočeno povsod (Linux, Android).

3.1.2 3D platno WebGL

WebGL [14] je medplatformni programski vmesnik, uporabljen za delo s tridimenzionalno grafiko znotraj spletnega brskalnika. Je kontekst platna, ki ima direkten dostop do grafične kartice preko GLSL jezika za pisanje programov, ki se izvajajo direktno na grafični kartici. Ti programi se imenujejo senčilniki (angl. shaders).

WebGL temelji na OpenGL ES 2.0 standardu in je na voljo na namiznih računalnikih in na nekaterih mobilnih napravah. Podpora OpenGL ES 2.0

na mobilni napravi še ne pomeni, da naprava podpira WebGL. Tak primer je iOS, ki WebGL standarda zaenkrat še ne podpira.

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 vmesnika API. 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 dvodimenzionalnega platna, brez strojne podpore sploh ne deluje.

WebGL kompatibilnost ima svoje probleme tudi na namiznih računalnikih, saj na določenih kombinacijah operacijskih sistemov, grafičnih kartic in brskalnikov 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 težave. Omejitve sicer lahko zaobidemo s postavitvijo posebne zastavice ob zagonu brskalnika, vendar s tem lahko tvegamo anomalije pri prikazovanju ali celo nestabilnost brskalnika.

Razvijalci brskalnikov kot glavni razlog za slabo podprtost WebGLa navajajo probleme z varnostjo. Narediti peskovnik (angl. sandbox) za spletno

stran ni trivialno, še posebej če le ta za svoje delovanje potrebuje direkten dostop do grafične kartice.

Razvoj WebGL aplikacije

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

Senčilniki so programi, ki se izvajajo na grafični procesni enoti. WebGL definira dve vrsti senčilnih programov - ogliščni (angl. vertex) in fragmentni (angl. 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 je v primerjavi s programskim jezikom ANSI-C sicer omejen, vendar so dodani posebni ukazi za lažje delo z vektorji in matrikami.

Podobno kot za dvodimenzionalno 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, da se bo podpora v prihodnosti precej izboljšala. Dober indikator je tudi vključitev podpore v Internet Explorer 11, kljub dejstvu, da sta bila skupina Khronos in Microsoft v nenehni tekmovalnosti.

Za razliko od zaprtih sistemov, kot je na primer Flash, ki mu podprtost pada[1], je WebGL trenutno na dobri poti, da postane primerno orodje za razvoj grafično intenzivnih medplatformnih aplikacij.

[1] Adobe Flash na iOSu ni bil podprt nikoli, na Androidu pa v zadnjih verzijah tudi uradno ni več podprt.

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 vmesnik 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 ali tonalitete.

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

Uporabimo lahko tudi knjižnico, ki nam za predvajanje zvoka ponudi svoj lasten vmesnik 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 se morajo odzivati tudi na vhod uporabnika. Za vhod je na namiznih računalnikih značilna kombinacija miška in tipkovnica, na mobilnih napravah pa imamo navadno na voljo samo 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. V industriji je najbolj pogosto uporabljen 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 kažejo, da je ekvivalenten program napisan v programskem jeziku C++ lahko tudi do 5 krat hitrejši [15].

S pomočjo tehnologij, kot je ASM.js 4 je možno Javascript program pohitriti, vendar je kljub temu povprečna hitrost izvajanja še vedno 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 vmesnika API. Njen glavni cilj je narediti bogato orodje, ki bo olajšalo delo z 2D in 3D grafiko [16].

Knjižnica je trenutno še globoko v razvoju in zaenkrat stabilna verzija še ni bila izdana. OpenGL ES 2.0 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 več potrebno skrbeti glede delovanja v različnih brskalnikih.

Delo poteka tudi na prevedbi knjižnice za sistema iOS in Android. To bi omogočilo bolj konsistentno medplatformno delovanje aplikacije na več platformah.

3.2.1 LycheeJS

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 [17]. 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 za 3D zmogljivosti zanaša na V8-gl, veljajo enake omejitve pri uporabi OpenGL ES2.0 APIja kot pri V8-gl. LycheeJS je zgolj ogrodje zgrajeno nad V8-glom, ki programerju omogoči abstrakcijo in lažje pakiranje aplikacije. Med prednostmi, ki jih LycheeJS prinaša so ogrodje za delo z 2D grafiko, enoten vmesnik za detekcijo vhoda uporabnikov in enoten vmesnik za predvajanje zvočnih datotek. Orodja za pakiranje aplikacij, ki so vključene v projekt LycheeJS, omogočajo pakiranje aplikacije za različne platforma z enim samim ukazom. Trenutno podprte platforme so spletne aplikacije, vtičniki za brskalnik Google Chrome, namizna aplikacija ter Android aplikacija. Seznam aplikacij pa se bo v prihodnosti še povečal.

3.3 Xamarin

Xamarin nam omogoča, da medplatformno aplikacijo napišemo v programskem jeziku C#. Znotraj programskega jezika C# pri Xamarinu pripravili posebni programski vmesniki, s katerimi lahko dostopamo do knjižnic na domorodnih platformah. Na voljo imamo tudi funkcije iz .NET ogrodja, saj C# prevajalnik našemu programu doda .NET rutino (Mono). Prevajalnik proizvede izvedljiv program za ARM procesorje, ki je lahko zapakiran kot iOS ali Android aplikacija. Na ta način lahko delimo del izvorne kode med Android in iOS aplikacijami. Določen del kode pa mora vseeno biti ločen, saj Xamarin ne ponuja skupne abstrakcije nad iOS in Android sistemom.

Xamarin kodo prevede v domorodno izvedljivo binarno datoteko za posamezno platformo. Izvajanje binarne datoteke je hitro in ni vidnih vplivov na hitrost izvajanja [18]. Izvorna koda .NET rutine nam sicer prinese dodatnih 2.5MB podpisa, vendar to danes ne predstavlja večjega problema.

Xamarin temelji na odprto kodni verziji .NET ogrodja - Mono, ki deluje na platformah Linux, Unix, FreeBSD in MacOSX. Za iOS so razvili lasten prevajalnik, ki izvorno kodo v ARM zbirni jezik prevede pred časom (angl.

ahead of time). Na operacijskem sistemu Android Xamarinov prevajalnik prevede kodo v vmesni jezik (IL), ki se nato prevede ob pravem času (angl. just in time), ko se aplikacija zažene. V obeh primerih Xamarin poskrbi za dodeljevanje spomina, sproščanje pomnilnika in osnovne komponente platforme.

Pisanje grafičnih aplikacij z Xamarinom je sicer mogoče, vendar nam Xamarin pri razvoju le malo pomaga. Kot bomo videli v nadaljevanju je Xamarin bolj uporaben kot vmesna plast. Tako ogrodje LibGDX [3.4] kot PlayN [3.5] uporabljata Xamarin za grajenje aplikacije za iOS platformo.

Uporaba Xamarina prevajalnika je sicer možna brez plačila, vendar je velikost aplikacije omejena. Za neomejeno velikost aplikacije je potrebno kupiti eno od plačljivih licenc. Najcenejša licenca, ki omogoča prevajanje večjih aplikacij stane $299 na posamezno platformo.

3.4 LibGDX

LibGDX [19] je v Javi napisano ogrodje za medplatformni razvoj grafičnih aplikacij. Knjižnica abstrahira razlike med namiznimi računalniki, Androidom, iOSom in spletnimi aplikacijami ter gradi na odprtih standardih, kot sta OpenGL ES in WebGL.

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

Prednost tega pristopa je krajši čas razvijanja aplikacije. Vsake spremembe nam ni potrebno preveriti tudi na mobilni napravi. Grajenje iOS ali Android paketa namreč traja nekaj časa, zgrajen paket pa je potem potrebno prenesti še na mobilno napravo. Testiranje na Android emulatorju ni mogoče, saj le ta ne podpira OpenGL ES 2.0 standarda. Precej hitreje vidimo rezlutat, če na namiznem računalniku poženemo domorodno aplikacijo.

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 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.

Kritični deli ogrodja LibGDX so bili napisani v programskem jeziku C++, iz Jave pa se jih kliče s pomočjo domorodnega vmesnika (angl. Java Native Interface). S tem ogrodje LibGDX ni omejeno na hitrost izvajanja Jave.

LibGDX omogoča razvoj aplikacij za Widows, Linux, Mac OS X, Android 1.5+, iOS in WebGL.

Za grajenje iOS paketa je potrebno orodje Xamarin [3.3]. LibGDX s pomočjo LLVM Java kodo spremeni v kompatibilno s C#, ki jo potem Xamarin prevajalnik prevede v domorodno ARM kodo.

Projekt LibGDX je odprto koden in se še vedno aktivno razvija.

3.5 PlayN

PlayN [20] je ogrodje za razvoj grafično intenzivnih aplikacij na različnih platformah napisano v Javi. Podprte platforme so namizni računalniki (Java), iOS, Android, spletne aplikacije in Flash.

Vmesnik je napisan v programskem jeziku Java. Našo aplikacijo lahko razvijamo na namiznem računalniku, uporabimo pa lahko tudi vroče izmenjavanje kode.

Za izvoz na različne platforme uporabimo orodje Ant ali Maven. Večjih razlik med pristopoma ni iz perspektive uporabnika ogrodja ni, saj oba omogočata preprost način grajenja domorodnih paketov za Android, iOS, HTML5 in Flash.

3.5.1 GWT

Izvoz aplikacije v HTML5 je mogoč z uporabo Googlovih spletnih orodij (angl. Google Web Toolkit, GWT). GWT omogočajo razvijanje kompleksnih aplikacij v Javi. Java aplikacija se prevede v Javascript, ki je optimiziran za posamezne spletne brskalnike. Celoten paket vsebuje knjižnico z Java programskim vmesnikom, prevajalnik in strežnik za razvijanje aplikacije.

GWT program napisan v jeziku Java se 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 odstrani tudi mrtvo kodo, optimizira nize znakov in metode pretvori v enovrstično varianto.

3.5.2 iOS

Za grajenje iOS aplikacije veljajo enake zahteve kot pri LibGDX knjižnici. Java izvorna koda se prevede v C#, ki se potem s pomočjo Xamarin prevajalnika prevede v ARM izvorno kodo.

3.6 Unity

Unity3D [22] je razvojno okolje za izdelovanje medplatformnih grafično intenzivnih aplikacij. Poleg mobilnih naprav (iOS, Android, BlackBerry, Windows Phone 8) ter vseh glavnih namiznih operacijskih sistemov (Windows, MacOS, Linux), orodje omogoča izdelovanje aplikacij tudi za igralne konzole (Xbox in PS3). Orodje je sestavljeno iz dveh glavnih delov - Unity pogon in integrirano okolje za razvijanje.

3.6.1 Pogon

Za risanje na zaslon Unity uporablja grafični vmesnik Direct3D (na platformi Windows in Xbox) in OpenGL ES (iOS, Android). Pogon omogoča napredne napredne funkcijem, kot so dinamične sence, celozaslonski efekti po procesiranju in renderiranje na teksturo.

V pogon lahko uvozimo modele v različnih formatih. Podprti so 3ds Max, Blender, Maya in drugi.

3.6.2 Integrirano razvojno okolje

Integrirano razvojno okolje razvijalcu omogoči hiter razvoj grafične aplikacije in tudi celovit pregled nad sceno, ki jo trenutno razvija. Razvojno okolje ima dve glavni stanji. Prvo stanje - stanje razvijanja - je namenjeno dodajanju objektov v sceno, spreminjanje njihovih nastavitev, določanje materialov in drugih nastavitev. Drugo stanje - stanje igranja - pa simulira potek izvajanja grafičnega programa in razvijalcu omogoča pregled nad delovanjem aplikacije.

Izvoz na posamezne platforme poteka preko dialoga za izvoz. V dialogu določimo želeno platformo in nastavitve. Aplikacija se potem prevede in optimizira za izbrano platformo.

3.7 Programski jezik Haxe

Programski jezik Haxe [23] je bil ustvarjen z razlogom, da bi olajšal razvoj medplatformnih aplikacij. Prevajalnik zna izvorno kodo prevesti na veliko različnih platform. Izvorna koda napisana v jeziku Haxe je lahko prevedena v JavaScript, Adobe Flash, NekoVM, PHP, C++, C# in Javo.

Jezik Haxe temelji na jeziku C in se zaradi podobnosti drugim jezikom (Java, Javascript, ActionScript) ni težko učljiv. Haxe je strogo tipiziran, kar pomeni, da prevajalnik že med prevajanjem programa lahko odkrije določene vrste napak. Na voljo je tudi inferenca tipov, generiki in zaprtje funkcij.

Jezik je odprto koden in prost za uporabo tako za odprto kodne kot

komercialne projekte.

3.8 Razvoj medplatformnih aplikacij v programskem jeziku C++

C++ dostopen povsod. Linux, iOS, OSX in Windows imajo domorodno podporo za C++. Android C++ podpira z orodjem za domoroden razvoj aplikacij (angl. Native Development Kit, NDK), na voljo pa je tudi na platformi iOS. Uporaba jezika C++ za razvoj aplikacij na mobilnih paltformah nam omogoči dostop do sistema za grafiko (OpenGL, oz. Direct3D na Windows), redko pa tudi do elementov za uporabniški vmesnik. Le te na mobilnih napravah praviloma lahko uporabljamo samo iz domorodnih programskih jezikov za določeno platformo ali pa s pomočjo orodij kot je ObjectiveC++, .NET most in Java JNI.

Značilnost grafično intenzivnih aplikacij je v tem, da za svoje delovanje ne potrebujejo veliko elementov za uporabniški vmesnik, saj večinoma tečejo v celozaslonskem načinu. Zato problem iz prejšnjega odstavka ni tako pereč, še posebej, če se zavedamo prednosti, ki jih pridobimo z uporabo C++.

Prednosti vključujejo eno programsko kodo aplikacije za vse platforme. Le te ni potrebno prevajati v druge jezike, kot smo videli pri [3.4, 3.7, 3.5] hkrati pa tudi nimamo problemov z učinkovitostjo in slabo podprtostjo [3.1.1, 3.1.2].

Vseeno se izkaže, da implementacija programskega jezika C++ na posameznih platformah ni povsem enaka. Androidov NDK nima vgrajenih velik nabor funkcij, ki so drugje standardne. Tak primer so izjeme, ki v NDKju niso podprte, vendar se jih moramo dodati sami. Za premostitev razlik med implementacijami se uporabljajo orodja, kot je cmake, ki pred korakom prevajanja poskrbi, da se bodo prevedle tudi vse dodatne knjižnice, ki jih aplikacija potrebuje na določeni platformi.

3.8.1 Gameplay

Projekt Gameplay je odprtokodni 3D grafični pogon, ki deluje na različnih platformah [24]. Pogon razvija podjetje Blackbarry, ki s projektom želi vzpodbuditi razvoj za njihove naprave. Poleg BlackBerry naprav so podprte še iOS od verzije 5 naprej, Android od verzije 2.3 dalje, Windows 7, OSX in Linux. Mobilne naprave z operacijskim sistemom Windows niso podprte.

Poleg metod za dostop do OpenGL ES 2.0 knjižnice na mobilnih napravah, oziroma OpenGL 3.2 na namiznih računalnikih, ima projekt vgrajen tudi generator terenov, graf scene, sistem za računanje fizike, deklarativen sistem za uporabniški vmesnik, podprt pa je tudi skriptni jezik Lua.

Grajenje medplatformnih aplikacij omogoča orodje cmake, ki poskrbi za vse pritikline na posameznih platformah.

3.8.2 OGRE

OGRE (Object-Oriented Graphics Rendering Engine) [25] je pogon za upodabljanje napisan v programskem jeziku C++. Prednost pogona je, da nam ponudi enoten vmesnik tako za OpenGL kot Direct3D. OGRE podpira večino različnih sistemov Linux, Windows (tudi Phone in RT), OSX, iOS in Android.

Prednost OGRE pred drugimi podobnimi projekti je je tudi, da ima zelo dobro dokumentacijo in da se striktno drži standardov programiranja. Tudi na splošno je pogon dobro zamišljen in konsistenten.

OGRE ima preprost objektno orientiran vmesnik, ki poenostavi delo, ki je potrebno za ustvarjanje 3D scen.

3.8.3 Marmalade

Marmelade [26] je zanimivo orodje, ki nam omogoča pisanje medplatformnih aplikacij v jeziku C++, Javascript ali pa s programskim jezikom Lua. Izbira jezika je odvisna od naših preferenc in zahtevnosti projekta.

Orodje Marmalade ni na voljo brezplačno, brez plačila je na voljo samo 30 dnevna verzija. Cena najbolj osnovne licence, ki nam omogoča grajenje

aplikacij za platforme iOS in Android je $15 na mesec. Če pa bi želeli podpreti še BlackBerry in Windows Phone pa cena naraste na $499 na leto.

Marmalade nudi direkten dostop do OpenGL ES vmesnika API, pester nabor možnosti za razvoj uporabniških vmesnikov in kompatibilnost z popularnimi knjižnicami za C++.

Orodje Marmalade je sestavljeno iz dveh delov. Prvi je Marmalade sistem, ki je plast abstrakcije za delo z operacijskim sistemom. Drugi pa Marmalade studio, ki je komplet orodij za delo z 2D in 3D grafiko. Moduli skrbijo za 2D in 3D izris, renderiranje modelov in bitmap pisave.

Za grajenje medplatformnih aplikacij se uporabljajo MKB datoteke, v katere se za vsako platformo zapišejo nastavitve, kot so na primer katere datoteke naj se vključijo, definicije za predprocesor in nastavitve za izvoz na naprave. Iz izvorne kode aplikacije se ustvari platformno neodvisna binarna datoteka. To binarno datoteko potem s priloženim orodjem za izvoz lahko izvozimo na želene platforme.

3.9 Monogame

Monogame je orodje, ki lahko obstoječe aplikacije pisane za platforme Windows in Windows Phone, pripravi še za druge platforme. Trenutno podprti sistemi so OSX, Linux, iOS, Android, Play Station Mobile in Ouya.

Sprva je Monogame podpiral samo prevedbo 2D aplikacij, vendar je dobil podporo za 3D programski vmesnik z verzijo 3.0.0. Cilj projekta je v celoti podpreti XNA 4 vmesnik API. Za prevod obstoječe aplikacije na iOS in Android se uporablja Xamarin platforma [3.3], kar pomeni, da moramo pridobiti dve licenci. Verzija 3.0.0 je osredotočena na funkcije, ki jih prinaša OpenGL ES 2.0 medtem ko so prejšnje verzije temeljile na OpenGL ES 1.X. Na Windows sistemih se namesto OpenGLa uporablja Direct3D.

3.10 Adobe Flash

Adobe Flash [27], znan tudi pod imeni Shockwave Flash in Macromedia Flash, je uporabnikom najbolj poznan v obliki vtičnika za spletne brskalnike. Začetki segajo v leto 1995, ko je bil razvit kot vtičnik za animacije na spletnih straneh. Matično podjetje je nato kupila Macromedia, ki pa se je kasneje prodala Adobeju.

Od Flash verzije 11 naprej ima predvajalnik podporo tudi za strojno pospeševanje 3D vsebin.

Podpora na mobilnih platformah je slaba, saj iOS Flasha nikoli ni podpiral, na Androidu pa je od verzije 4.0 uradno ni več podprt, neuradno je možno Flash predvajalnik namestiti tudi na Android sistemih 4.1 in 4.2. Zaradi slabih smernic Flash ni najbolj primeren za razvoj grafično intenzivnih aplikacij.

Probleme s slabo podporo lahko nekoliko premostimo z uporabo vmesnikov, ki aplikacijo znajo zapakirati, kot domorodne aplikacije za posamezne platforme.

3.10.1 Stage3D

Stage3D programski vmesnik nam mogoča strojno pospešeno arhitekturo, ki deluje na več platformah.

3.10.2 Starling

Starling je odprto koden projekt, ki temelji na Stage3Dju. Nad stage3D doda dodatno funkcionalnost kot so sistem za dogodke, podpora partiklom, podpora različnim teksturam, teksturni atlasi, različni načini blendanja in podpora za več dotikov.

3.11 QT

Projekt QT je medplatformno ogrodje za ustvarjanje aplikacij. Za razvijanje aplikacij sta na voljo C++ in QML, ki je jezik podoben Javascriptu in CSSu.

Za razvijanje aplikacij je na voljo tudi vgrajeno razvijalno okolje QT Creator.

Qt je možno uporabljati na velikem številu različnih naprav naprav in platform z uporabo različnih CPU arhitektur. QT skupnost med drugimi podpira tudi naprave Beagleboard [2.3.2] in RaspberryPi [2.3.1].

iOS in Android sta trenutno eksperimentalno podprta kot možni platformi.

Poglavje 4

Asm.js

Asm.js [28] je podmnožica Javascripta, ki je nastala kot raziskovalni projekt pri Mozilli. Izvorno kodo Javascripta je zaradi dinamičnosti jezika zelo težko optimizirati v modernih brskalnikih. Asm.js specifikacijo jezika omeji tako, da je brskalniki kodo lažje optimizirajo. Cilj projekta je tudi dokumentirati možne pospešitve Javascript izvorne kode.

Asm.js ni novost. 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 (angl. singleton) tipiziranega spomina in uporaba bitnih operatorjev za spremenljivke, ki se obnašajo kot cela števila v C++.

ASM.js tudi ni prvi projekt, ki iz različnih jezikov generira Javascript izvorno kodo. Leta 2006 je podjetje Google izdalo Google Web Toolkit (GWT), ki poleg drugih stvari, prevaja izvorno kodo iz programskega jezika Java v JavaScript. Od leta 2006 se je pojavilo kar nekaj podobnih prevajalnikov za že obstoječe 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 razvijalcem JavaScript pogonov 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 in je veliko več interne komunikacije. Le te pa ostali brskalniki niso deležni. Asm.js dokumentira vse možne pohitritve in navodila za pohitritve da na voljo vsem razvijalcem brskalnikov.

Asm.js se izogiba potencialnih upočasnitev v kodi tako, da 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.

Asm.js poskrbi za precej optimizacij v času izvajanja:

- Tipi spremenljivk se pokažejo med preverjanjem tipov. To omogoča prevajanje pred časom in ne samo ob pravem času.
- JavaScript pogon ima garancijo, da se tipi spremenljivk med izvajanjem ne bodo spreminjali. S tem lahko pogon proizvede bolj preprosto in bolj učinkovito kodo.
- Sistem tipov v Asm.js olajša globalno strukturiranje programa (klici funkcij, dostop do spomina).

Izvorna koda, ki jo definira Asm.js, je še vedno dvakrat počasnejša od domorodne kode napisane v nižje nivojskih jezikih kot sta C in C++, vendar se bo s časoma Asm.js še izboljšal.

Ker je Asm.js koda podmnožica JavaScripta lahko že danes teče v vseh brskalnikih, vendar ni nobene garancije, da bodo brskalniki pohitritve znali tudi izkoristiti.

Asm.js trenutno podpira prevajanje iz več jezikov, vendar sta popolnoma podprta samo jezika C in C++. Drugi jeziki so podprti le deloma in niso deležni enakih pohitritev in optimizacij.

Dinamični jeziki, kot so Python, Ruby in Lua, so še v zgodnjih fazah 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 bitne 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.

Kljub omejitvam pri prevajanju dinamičnih jezikov in jezikov na virtualnih strojev, je Asm.js zelo perspektiven projekt, ki bo zaradi svoje odprtosti vsekakor pripomogel k hitrejšemu izvajanju Javascript kode znotraj brskalnikov.

Poglavje 5

Programiranje na grafičnem čipu

Ker je grafični cevovod (tako pri Direct3D, kot pri OpenGLu) z leti postal zelo zapleten in je prenašanje podatkov med centralno procesno enoto in grafično procesno enoto zahtevna operacija, se pojavlja nova alternativa - pisanje aplikacij direktno na grafičnem čipu. Taka aplikacija lahko s kompatibilno grafično kartico dela na več platformah - premestiti je potrebno samo dele aplikacije, ki imajo opravka z nastavljanjem ogrodja.

Nekompatibilnosti in razlike med posameznimi grafičnimi grafičnimi karticami bi bilo mogoče odpraviti z uporabo grafičnega pogona. Tak grafični pogon bi uporabniku ponudil enoten vmesnik API, s katerim bi uporabnik napisal grafično intenzivno aplikacijo. Grafični pogon bi abstrahiral delovanje posameznih kartic in skril razlike med implementacijami. Tak grafični pogon je še zelo daleč od realnosti, vseeno pa se splača pogledati namenske programske jezike, ki nam omogočajo izvajanje programske kode na grafičnem procesorju.

Primeri takšnih splošno namenskih jezikov so nVidiina CUDA, Microsoftov AMP in OpenCL.

5.1 CUDA

CUDA [29] je platforma za paralelno procesiranje in programski model, ki z uporabo grafične kartice omogoča dramatično pohitritev izračunov. Podjetje nVidia je orodje CUDA predstavilo leta 2006, danes pa se veliko uporablja za znanstvene izračune na različnih področjih fizike, kemije, biologije in drugih.

Z uporabo platforme CUDA lahko izvorno kodo napisano v jezikih C, C++ in Fortran pošljemo direktno na grafično procesno enoto. Lažje razvijanje aplikacij je na voljo tudi orodjarna CUDA, ki vsebuje prevajalnik, matematične knjižnice in orodja za razhroščevanje in optimizacijo aplikacij.

5.1.1 Logan - CUDA na mobilnih platformah

Podjetje nVidia je 24. julija 2013 [30] na spletni strani objavila predogled novega jedra za mobilne naprave. Logan, kot se novo jedro imenuje, bo prvo jedro, ki bo podpiralo CUDA tehnologijo na mobilnih napravah.

Grafična procesna enota projekta Logan, temelji na nVidijini Kepler arhitekturi. Kepler arhitektura je uporabljena tudi na namiznih računalnikih, prenosnikih, delovnih postajah in super računalnikih.

Kepler bo tudi na mobilnih napravah imel podporo standardom OpenGL4.4, OpenGL ES 3.0 in DirectX11.

Poleg orodja CUDA bo Logan na mobilne naprave prinesel tudi vmesnik API, ki bo razvijalcem grafično intenzivnih aplikacij omogočil uporabo teselacije (to je tehnologija, ki aktivno spreminja količino izrisanih trikotnikov glede na potrebe aplikacije), preneseno izrisovanje, ki omogoča izračun vplivov različnih luči v sceni v enem samem prehodu procesiranja, napredno glajenje robov in vmesnik API za računanje fizike in podobnih simulacij.

Z vsemi temi funkcijami se bo izrisovanje na mobilnih napravah precej približalo zmožnostim namiznih računalnikov.

5.2 OpenCL

OpenCL (angl. Open Computing Language) [31] je ogrodje za pisanje programov, ki se izvajajo na heterogenih platformah, ki so sestavljena iz več centralno procesnih enot, grafičnih procesnih enot, digitalnih procesorjev signalov in drugih procesorjev. OpenCL je odprt standard, ki je podprt na različnih napravah, od namiznih računalnikov in delovnih postaj, do mobilnih naprav. Za razvoj standarda skrbi skupina Khronos.

5.2.1 Accelereyes

Obstajajo knjižnice, ki lahko prednosti OpenCLja uporabljajo tudi na mobilnih napravah. Ena izmed teh knjižnic je AccelerEyes [32], ki obljublja procesiranje videa v realnem času, hitrejše procesiranje podatkov in boljše izračune nad fotografijami. AccelerEyes je C/C++ knjižnica s preprostim matričnim programskim vmesnikom. Enaka izvorna koda se lahko uporabi tako na namiznih kot na mobilnih platformah. Od mobilnih platform sta trenutno podprta Android in iOS.

5.3 AMP

AMP (Accelerated Massive Parallelism) je Microsoftova knjižnica, ki pospeši delovanje C++ programske kode tako, da uporabi prednosti paralelnega izvajanja, ki ga ponujajo grafične procesne enote.

Implementacija AMP temelji na standardu DirectX11 in je zato dostopna samo na napravah, kjer je le ta podprt (Windows 7 in Windows Server 2008 R2 ali novejše). Na mobilnih napravah trenutno še ni na voljo. Ker pa je standard odprt se pričakuje, da bo vedno več podprtih naprav v prihodnosti. Na platformah, kjer AMP ni podprt potrebno delo namesto grafične procesne enote opravi centralna procesna enota. Tudi v tem primeru lahko uporaba AMPja pohitri delovanje, saj je izvorno kodo pisano za AMP lažje izvajati paralelno na več jedrih.

AMP trenutno podpira večdimenzionalne sezname, indeksiranje, prenašanje spomina, in tiling.

Poglavje 6

Študije primerov

6.1 Primer za učenje jezikov

6.1.1 Opis problema

Prvi primer je preprosta grafična aplikacija, ki uporabnikom olajša učenje tujih jezikov. Deluje na preprostem principu pomnjenja besed. Igralcu se na zaslonu pokaže beseda v domačem jeziku in tri besede v jeziku, ki se ga uporabnik skuša naučiti. Dve besedi sta naključno izbrani, tretja pa je pravilni odgovor. Vrstni red besed je naključen.

V primeru pravilnega odgovora se uporabniku pokaže naslednja beseda in trije novi odgovori. Tako uporabnik nadaljuje z učenjem. Če uporabnik izbere napačni odgovor, se na zaslonu pojavi pravilni prevod besede. Na ta način se ima uporabnik možnost naučiti novo besedo. Ko si uporabnik ogleda pravilni odgovor, se igra začne na začetku.

Primer delovanja aplikacije je na sliki 6.1, kjer je viden glavni zaslon aplikacije s tremi možnimi odgovori.

Ena izmed glavnih zahtev igre je izpis različnih pisav in posebnih znakov. Poleg prikazane verzije nemščina-angleščina, je bila aplikacija razvita tudi z idejo učenja jezikov, ki ne uporabljajo latinice. Slika 6.2 prikazuje primer učenja Korejščine.

Slika 6.1: Primer delovanja Nemške verzije na mobilni napravi

Slika 6.2: Primer delovanja Korejske verzije na namiznem računalniku

6.1.2 Uporabljena metoda

Izmed vseh obravnavanih metod se nam je zdela najbolj primerna metoda PlayN. Razlogi za izbiro metode so naslednji:

- PlayN je odprto koden projekt. V primeru težav lahko pogledamo v izvorno kodo projekta in po potrebi težave odpravimo sami.
- Licenca, ki jo PlayN uporablja, ni omejujoča in v primerjavi z nekaterimi plačljivimi metodami, ne zahteva nobenega plačila pred uporabo. To velja tudi, če bi aplikacijo uporabili za komercialne namene.
- PlayN je še vedno v aktivnem razvoju, razvijalci pa so odzivni na listi za elektronsko pošto.
- PlayN omogoča preprosto podporo dodajanju lastnih pisav, kar je ključnega pomena za podporo jezikom, kot je Korejščina.
- Število platform, ki jih PlayN podpira, zadošča potrebam aplikacije.
- PlayN omogoča uporabo vročega izmenjevanja kode, kar zelo pohitri razvoj aplikacije.
- Poleg samega ogrodja PlayN je na voljo tudi precej vtičnikov, ki so jih napravili uporabniki. Tak primer je vtičnik Tripleplay, ki je v aplikaciji uporabljen za prikaz menijev.
- Dokumentacija je dobro urejena.

6.1.3 Opis metode

Projekt z uporabo pogona PlayN sestavlja več imenikov. Glavni imenik se imenuje `core` in vsebuje logiko celotne aplikacije. Izvorna koda, ki se nahaja v tem imeniku, definira kaj se bo na zaslonu prikazalo ter kako se aplikacija odziva na vnose uporabnikov.

Imenik `assets` služi kot imenik vseh sredstev, ki jih aplikacija potrebuje za delovanje. V imeniku se nahaja vsa grafika, ki je v uporabi v igri, in vse zvočne datoteke.

Ostali imeniki so namenjeni posameznim platformam. Imenik `java` vsebuje preprost program, ki odpre okno in nastavi grafično okolje. Ko je okno pripravljeno pokliče glavno metodo imenika `core` in aplikacija se začne izvajati. Slično delujeta tudi imenika `android` in `ios`, vsak za svojo platformo. Oba definirata vse potrebno za zagon na sistemih Android in iOS in nato pokličeta metodo iz imenika `core`.

Aplikacija mora biti sposobna izrisovati velik nabor različnih abeced. To dejstvo je močno vplivalo na izbor metode. PlayN podpira tako 3D, kot 2D vmesnik za izrisovanje, vendar smo se zaradi kompleksnosti izrisovanja različnih pisav s 3D vmesnikom, odločili za uporabo 2D vmesnika. Naložitev lastnih pisav v projekt je zelo preprosta. Datoteko dodamo v imenik `assets`, iz programske kode pa pisavo registriramo s pomočjo metode `public void registerFont(String name, String path)` in nato uporabimo z uporabo metode `Font createFont(String name, Font.Style style, float size)`. Pisava, ki jo vrne zadnja metoda je tako na voljo za uporabo.

Pomembna zahteva aplikacije je bila tudi delovanje na mobilnih platformah iOS in Android. Tudi tu nam je ogrodje PlayN olajšalo delo, saj postavitev delovnega okolja ni bilo težavno. Proces izvoza aplikacije na različne platforme poteka z uporabo orodja Ant ali Maven. Maven je orodje za upravljanje s projekti, ki skrbi za grajenje projektov in njihovo dokumentacijo [21]. PlayN ima že predpripravljene nastavitve za Maven. Za grajenje Android aplikacije moramo izvesti samo preprost ukaz: `mvn install -Pandroid`.

6.1.4 Prednosti izbora metode

Izbrana metoda nam je pomagala izpolniti vse zastavljene cilje. Brez truda smo aplikacijo razvijali na namiznem računalniku in po potrebi preizkusili delovanje na Android napravi. Vmesnik API je razumljiv. Krivulja učenja ni bila strma.

6.1.5 Slabosti izbora metode

Težave smo imeli pri preizkušanju verzije za iOS naprave. Kljub izčrpni dokumentaciji smo naleteli na probleme s konflikti med pritiklinami ogrodja PlayN. Z nekaj raziskovanja nam je vse probleme uspelo odpraviti.

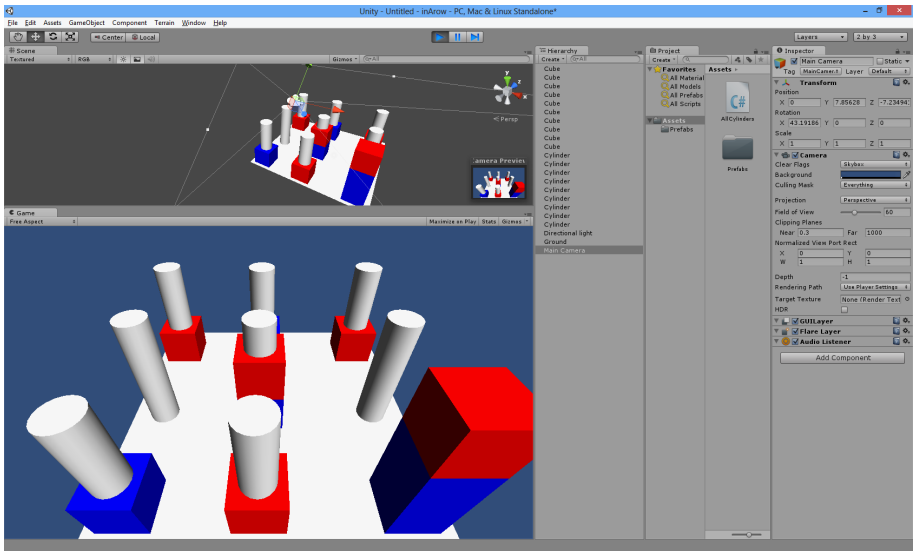
Težave je povzročil tudi prehod iz verzije 1.6 na 1.7, ki se je zgodil med razvojem aplikacije. Nova verzija je malce spremenila način dela s sredstvi in potrebnih je bilo nekaj ročnih popravkov, da sta Android in iOS verziji ponovno delovali.

6.2 Primer štiri v vrsto

Druga testna aplikacija je preprosta igra štiri v vrsto postavljena v treh dimenzijah. Ideja aplikacije je skozi preprosto igro izboljšati uporabnikovo orientacijo v 3D prostoru. Za razliko od standardne igre štiri v vrsto, kjer so zmagovalne kombinacije omejene v vodoravni, navpični in horizontalni smeri, imamo v 3D verziji veliko več možnosti. Poleg osnovnih smeri lahko zmagovalno kombinacijo zgradimo tudi v globino, kar odpre obilico novih kombinacij.

Igra se začne, ko prvi igralec z dotikom na valj postavi prvo kocko. Drugi igralec nato na podoben način postavi svojo prvo kocko, ki pa se barvno loči od kocke prvega igralca. Igralca se na tak način izmenjujeta, dokler eden izmed njiju uspe postaviti štiri kocke v vrsto. Takrat se zmagovalne kocke odebelijo in igra se konča.

Za lažjo orientacijo v prostoru igre je igralno površino možno obračati. Obračanje poteka tako, da s klikom na miško ali z dotikom na dotik občutljivi zaslon potegnemo po okolici igralne površine. Kamera se pomakne okoli središča glede na dolžino potega. Razdalja med kamero in središčem se ne spreminja. Prav tako se ne spremeni smer kamere, ki ves čas gleda proti središču igralne površine.



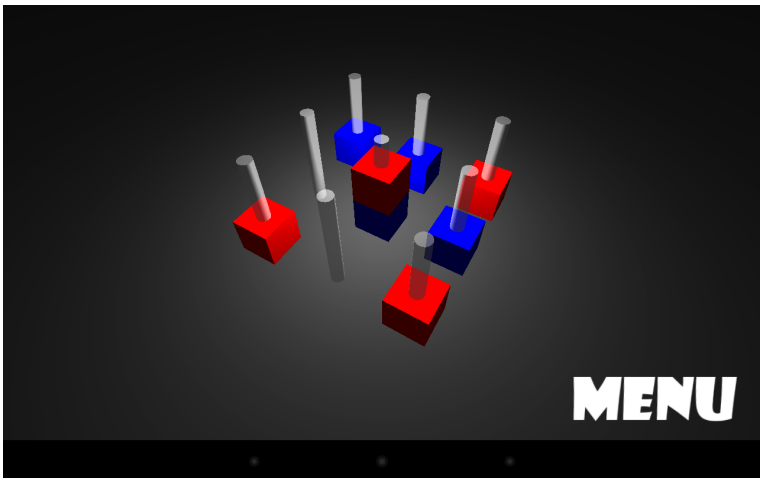
Slika 6.3: Razvoj grafično intenzivne aplikacije v okolju Unity.

6.2.1 Uporabljena metoda

Za razvoj smo se odločili za orodje Unity. Unity za razliko od ostalih metod, ki smo jih ogledali, vključuje lastno integrirano razvojno okolje. Integrirano razvojno okolje prikazuje slika 6.3. Omogoča nam postavljanje objektov v sceno, postavitev kamere, določitev luči in še množico drugih stvari, vse preko grafičnega vmesnika. Za postavitev objektov v 3D prostor imamo na voljo okno z ortografsko ali perspektivno projekcijo. Na preprost način sceno ogledamo iz različnih zornih kotov.

S pomočjo integriranega razvojnega okolja lahko določimo tudi različne lastnosti posameznih objektov. Te lastnosti so določitev materiala, različni sprožilci za dogodke in tip senc, ki jih objekt ustvarja. Za logiko, obnašanje in odziv na uporabniški vhod lahko vsakemu objektu dodelimo tudi skriptno datoteko. V skriptni datoteki v enem od podprtih programskih jezikih (C#, JavaScript, Boo) določimo, kako se bo objekt odzival na različne dejavnike in kako se bo obnašal v času, ko je viden na zaslonu.

Aplikacija ima preprosto zasnovo. Ko se igra zažene se ustvarijo prazni valji, na katere se nastavi dogodek `addBlock`, ki se sproži ob dotiku na valj.



Slika 6.4: Končana Unity aplikacija izvožena na mobilno platformo.

Unity okolje avtomatično prevede koordinate zaslona v koordinate prostora in pravilno določi valj, ki smo se ga dotaknili. Prav tako orodje Unity abstrahira način različne vrste vhoda, ki so lahko ali klik z miško ali dotik na dotik občutljivem zaslonu.

Dogodek `addBlock` na izbrani valj položi novo barvasto kocko - rdečo ali modro, odvisno kateri igralec je na voljo. Nova kocka se tudi doda na seznam, ki interno predstavlja stanje igralne plošče. Algoritem nato preveri novo stanje seznama. Če se je z zadnjo potezo pojavila zmagovalna kombinacija se igra konča.

Ko se igra konča, se pojavi besedilo, ki sporoči zmagovalca in ponudi ponovno igranje. V bolj pogostem primeru, ko poteza ne zaključi igre, se samo zamenja aktivni igralec in igra se nadaljuje.

Izvoz na različne platforme poteka preko posebnega dialoga za grajenje aplikacije (File - Build Settings...). V dialogu določimo ciljno platformo in nastavitve. S klikom na gumb Build se aplikacija prevede in zgradi za ciljno platformo. V primeru PC, Mac in Linux verzij kot rezultat grajenja dobimo binarno datoteko, ki je izvedljiva na teh platformah. V primeru iOS in Android aplikacije nam lahko Unity pripravi projekt, ki ga nato odpremo v

orodjih Eclipse ali Xcode. Tako lahko aplikacijo podpišemo in pripravimo za objavo v trgovinah z mobilnimi aplikacijami.

6.2.2 Prednosti metode

Unity urejevalnik nam je omočil zelo hiter razvoj aplikacije, saj je bilo ustvarjanje osnovnih oblik in postavitev le teh v prostor, zelo preprosto. Pisanje skriptnih datotek za obnašanje in logiko aplikacije tudi ni bilo težavno.

6.2.3 Slabosti metode

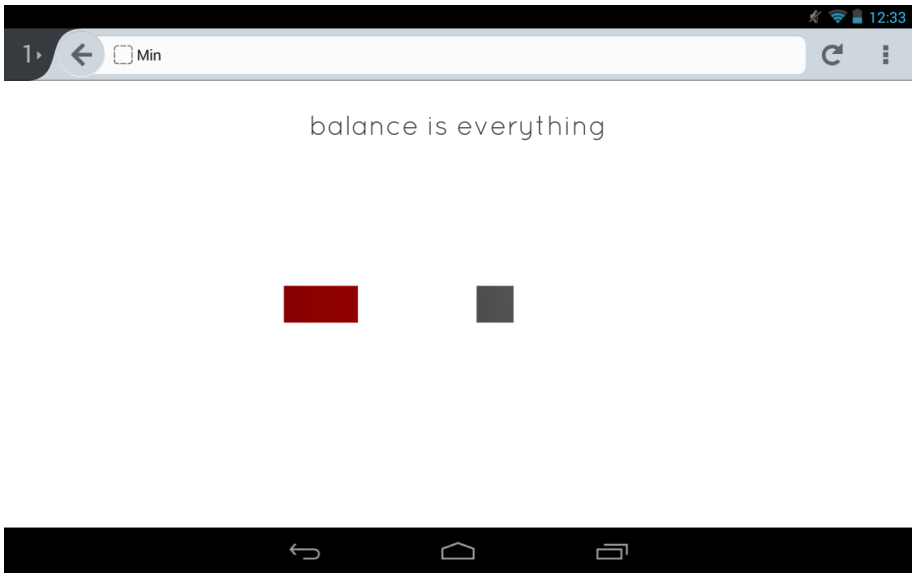
Problem uporabe orodja Unity je učna krivulja. Za razliko od večine ostalih metod, se moramo poleg vmesnika API naučiti tudi dela s priloženim urejevalnikom. Sicer je urejevalnik preprost za uporabo, vendar učenje vseh potrebnih operacij vzame precej časa.

Druga slabost metode je v zaprtost sistema. Če tekom razvoja aplikacije naletimo na omejitev orodja nimamo dostopa do izvorne kode, kjer bi to omejitev lahko odpravili. Orodje je sicer na voljo brezplačno, vendar moramo za uporabo naprednih funkcij plačati za licenco. Aplikacije razvite z brezplačno verzijo programa imajo na vseh platformah pred zagonom zaslon z napisom Unity.

6.3 Primer MIN

Tretji primer je preprosta igra, ki uporablja ortografsko projekcijo. Cilj igre je s sivo kocko odstraniti rdeče kocke. Sivo kocko premika igralec, rdeče kocke pa so postavljene v obliki ugank. Premikanje sive kocke levo, desno, navzgor in navzdol je možno samo v naprej določenih perspektivah.

V ptičji perspektivi, kjer vidimo pozicije vseh rdečih kock, premikanje sploh ni možno. V drugi perspektivi se lahko premikamo samo levo in desno, zaradi ortografske projekcije pa ne ločimo katere kocke so višje ali nižje nad nami. V tretji perspektivi se lahko premikamo navzgor in navzdol, ne



Slika 6.5: Delovanje igre v mobilnem brskalniku Firefox.

vidimo pa več katere kocke so levo in katere desno od nas. Zaradi omejenega gibanja si mora igralec v glavi ustvariti celotno sliko igralnega polja. Če se med spreminjanjem perspektive siva kocka dotakne črne črte, se igrana stopnja ponastavi. Slika 6.5 prikazuje delovanje končane aplikacije v mobilnem brskalniku.

Igra na prvi pogled deluje kot preprosta 2D aplikacija, vendar animirani prehodi med posameznimi perspektivami močno otežijo uporabo 2D platna. Iz tega razloga smo se odločili za uporabo 3D platna WebGL.

6.3.1 Uporabljena metoda

Za lažje delo z WebGLom smo uporabili knjižnico THREE.js, ki poenostavi delo s 3D objekti. Uporabili smo tudi knjižnico hammer.js, ki nam pomaga zaznavati geste na dotik občutljivem zaslonu.

6.3.2 THREE.js

THREE.js je knjižnica za delo z WebGLom. Ponudi nam pester nabor uporabnih metod, ki močno poenostavijo izrisovanje modelov, določanje materialov, postavitev kamere in raznoraznih luči v sceno. Knjižnica pripravi tudi potrebne senčilnike tako da uporabniku ni potrebno pisati GLSL izvorne kode. Pisanje lastnih senčilnikov vseeno obstaja kot opcija.

Nastavitev ortografske kamere je v knjižnici THREE.js zelo preprosto. Potreben je samo klic posebne metode `var camera = new THREE.OrthographicCamera( left, right, top, bottom, near, far )` s pravilnimi parametri. Kamero nato dodamo v sceno s klicom funkcije `scene.add(camera);`. Kamero premikamo po prostoru na podoben način kot vse ostale objekte - z nastavljanjem parametra `position`, ki ga ima vsaka instanca objekta kamere.

V sceno smo dodali tudi dve točkovni luči. Prvo smo postavili na rob igralne plošče tako, da poudari stranice kock v različnih perspektivah. Druga luč pa se nahaja nad igralno površino, da so kocke od zgoraj navzdol dobro osvetljene. Dodajanje točkovnih luči je podobno, kot dodajanje kamere. Najprej ustvarimo objekt `var cornerLight = new THREE.PointLight(0xFFFFFF);` nato pa ga dodamo v sceno `scene.add(cornerLight);`.

Na osebnih računalnikih premikamo kocko z uporabo smernih tipk, prehod med posameznimi perspektivami pa delamo s preslednico. Na mobilnih napravah nimamo na voljo fizične tipkovnice, navidezne pa zasedejo veliko prostora na zaslonu in tudi tipkanje je težavnejše. Zaradi tega na mobilnih napravah kocko uporabljamo s pomočjo gest. Za lažje delo z gestami smo uporabili knjižnico hammer.js.

6.3.3 hammer.js

Hammer.js je Javascript knjižnica za zaznavanje gest. Na podlagi standardnih dogodkov, ki jih brskalniki sprožijo ob dotiku zaslona, knjižnica ustvari nove višje nivojske dogodke, ki se sprožijo ob posameznih gestah. Tako se lahko naročimo na te višje nivojske dogodke in smo obveščeni, ko uporabnik

aplikacije izvede določeno gesto.

Višje nivojski dogodki so tipa poteg desno, poteg levo, dvoprstni poteg in tako dalje.

Na podoben način se lahko naročimo tudi na druge vrste dogodkov, med katere spadajo potegi v vse možne smeri, daljši pritisk, kratek pritisk, uščip za povečavo in tako naprej.

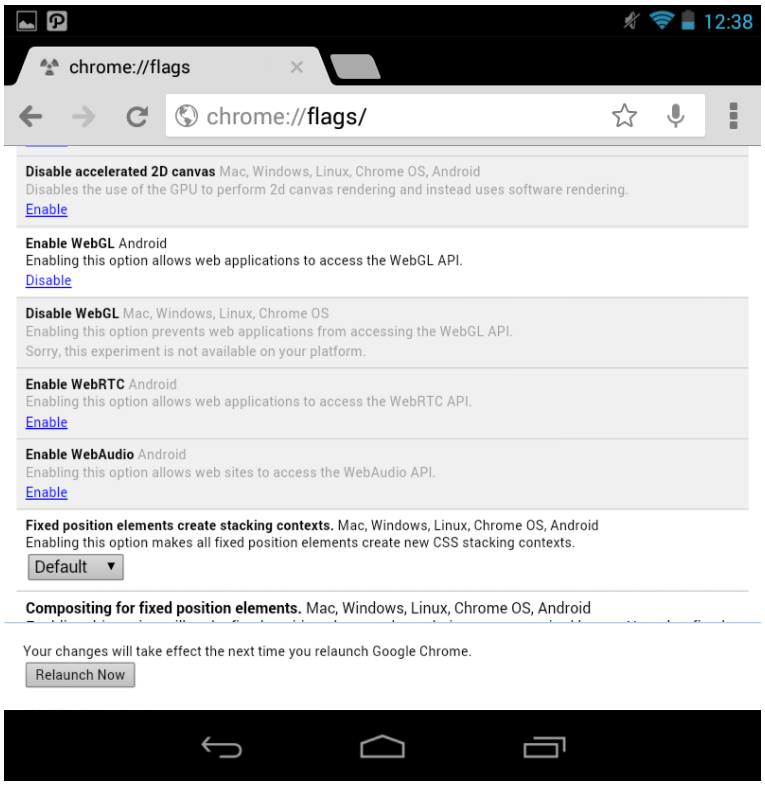
Knjižnica omogoča tudi nastavitve posameznih gest, kot so dolžina pritiska potrebna za dogodek daljši pritisk tako da lahko delovanje knjižnice prilagodimo našim specifičnim zahtevam. Za delovanje naše aplikacije to ni bilo potrebno.

6.3.4 Prednosti izbora metode

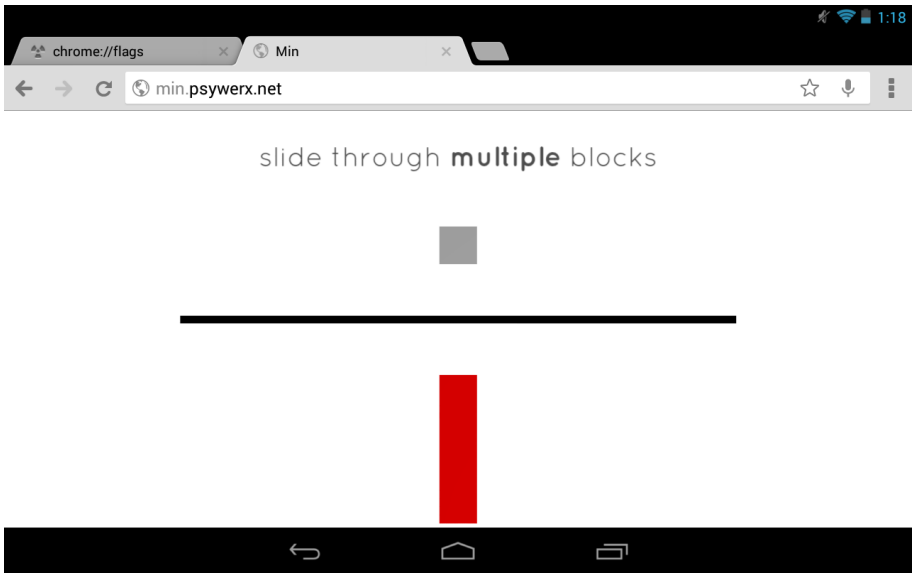
Izbrana metoda nam je omogočila hiter razvoj aplikacije. Aplikacijo smo razvijali na namiznem računalniku, kjer moderni brskalniki (Chrome, Firefox) podpirajo celo simuliranje dogodkov, ki se sprožajo na zaslonih občutljivih na dotik. Potrebe po testiranju na mobilni napravi je bilo zelo malo. V tem primeru smo delovanje na tabličnem računalniku Nexus 7 preizkusili šele, ko je bila celotna aplikacija končana.

6.3.5 Slabosti izbora metode

Glavna slabost metode je slaba podpora na mobilnih brskalnikih. Mobilni brskalnik Firefox je sicer prikazal našo aplikacijo pravilno, vendar smo morali za delovanje v mobilnem brskalniku Chrome omogočiti posebno nastavitev. Slika 6.6 prikazuje zaslon, kjer se nastavi zastavica za omogočanje WebGLa. Po vklopu zastavice je aplikacija delovala pravilno. Pravilno delovanje je vidno tudi na Sliki 6.8. Dokler podjetje Google privzeto ne omogoči te zastavice, WebGL aplikacije ne bodo primerne za navadne uporabnike.



Slika 6.6: Za delovanje v brskalniku Chrome na Androidu je potrebno nastaviti posebno zastavico.



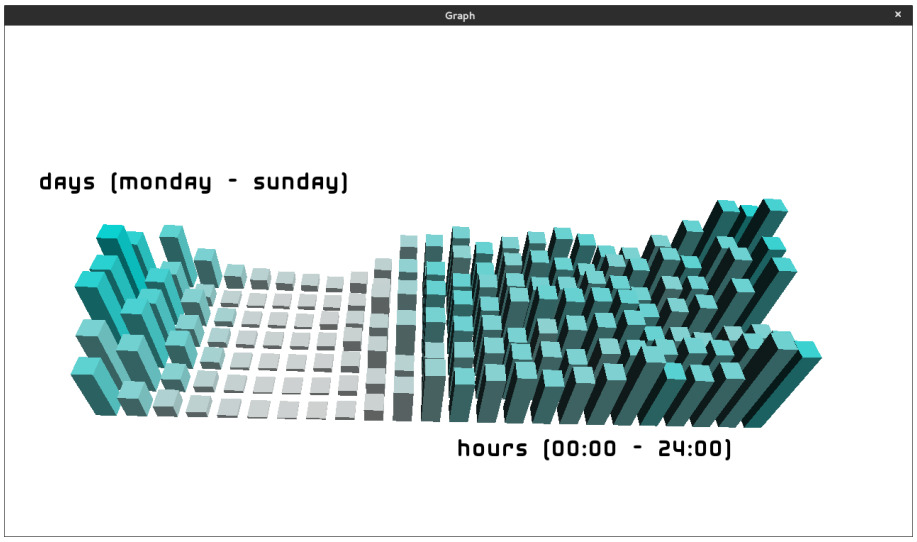
Slika 6.7: Delovanje igre v mobilnem brskalniku Chrome.

6.4 Primer 3D graf

Naš zadnji primer je 3D grafični prikaz aktivnost uporabnikov IRC kanala glede na dan v tednu in uro v dnevu. Graf prikazuje podatke zajete tekom enega leta. Vsak dan ima 24 stolpcev za vsako uro v dnevu. Celoten graf vsebuje 7 vrst po 24 stolpcev, kar je vse skupaj 168 stolpcev. Višina posameznega stolpca ponazarja količino aktivnosti.

Aplikacija je sestavljena iz dveh delov. Prvi del je preprosta skripta napisana v programskem jeziku Python in je zadolžena za pripravo ustreznih podatkov. Linearno se sprehodili čez celoten zapisnik pogovorov in vsako aktivnost zabeležili v 2D polje. 2D polje je sestavljeno iz 168 številskih vrednosti, kjer vsaka vrednost predstavlja količino aktivnosti na določen dan v tednu ob določeni uri. To polje se izvozi v tekstovno datoteko.

Drugi del aplikacije uporabi pripravljene podatke v tekstovni datoteki za izris 3D grafa. Za izrisovanje smo uporabili programski jezik C++ in pogonom gameplay.



Slika 6.8: 3D graf aktivnosti narejen z uporabo gameplay C++ knjižnice

6.4.1 Uporabljena metoda

Glavni imenik Gameplay projekta vsebuje imenik `res`, ki vsebuje vsa sredstva, ki jih aplikacija potrebuje. Sem spadajo materiali uporabljeni v aplikaciji, izvorna koda uporabljenih GLSL programov, teksture in fonti. Pod sredstva lahko dodamo tudi izvoženo sceno iz grafičnih programov Maya ali Blender, vendar jih moramo pred uporabo pretvoriti v poseben binarni format. Orodje za pretvorbo je priloženo.

Imenik `src` vsebuje vso izvorno kodo programa, imenik `android` pa vse potrebno za grajenje projekta za platformo Android. V glavnem imeniku je tudi imenik `.xcodeproj`, ki vsebuje potrebno za grajenje za iOS.

Grajenje aplikacije poteka v dveh fazah. Na najprej poženemo ukaz cmake, ki pripravi vse potrebne knjižnice, ki so potrebne za prevajanje aplikacije na izbrani platformi. Ko orodje cmake opravi svoje delo imamo vse pripravljeno za prevajanje aplikacije z ukazom make, ki pripravi binarno datoteko, ki jo lahko zaženemo. Za platformo android se namesto make ukaza uporabi ukaz ndk-build, ki je del Android paketa za razvoj domorodnih aplikacij.

6.4.2 Prednosti izbora metode

Gameplay pogon pripravi vse potrebno za razvoj medplatformnih aplikacij. Navodila za grajenje aplikacij na različnih platformah so preprosta in dobro dokumentirana. Uvoz datotek iz grafičnih programov deluje.

6.4.3 Slabosti izbora metode

Dokumentacija za uporabo pogona je na določenih mestih pomanjkljiva, tako da je med razvojem potrebno pogledati v izvorno kodo pogona ali priloženih primerov.

Poglavje 7

Sklepne ugotovitve

V diplomskem delu smo pregledali načine za medplatformni razvoj grafično intenzivnih aplikacij. Ogledali smo si različne platforme, na katerih je možno izvajati grafično intenzivne aplikacije, in ugotovili, da skoraj vsaka za domorodne aplikacije uporablja svoj programski jezik.

Prva ovira pri pisanju medplatformnih aplikacij je premostiti razlike med različnimi programskimi jeziki. Naslednja ovira je podprtost knjižnic za dostop do grafične kartice. Na mobilnih napravah so Microsoftovi telefoni in tablice edini, ki ne podpirajo odprtega standarda OpenGL, hkrati pa so to tudi edine naprave na katerih je podprta Direct3D knjižnica.

Premagati ovire z različnimi načini uporabniške vnosa ni bilo zahtevno, saj se dogodki za dotik na večini naprav obravnavajo na podoben način kot klik z miško. Na voljo so pa tudi knjižnice, ki te razlike še dodatno poenostavijo.

Načinov za premostitev razlik med posameznimi platformami je zelo veliko, zato smo se omejili na najbolj popularne metode. Pomemben faktor pri izbiri obravnavanih metod je bila tudi podpora različnim mobilnim napravam. Metod, ki za grajenje aplikacij uporabljajo računalništvo v oblaku, nismo obravnavali. Problem pri takih metodah je, da uporabljeni pristop za doseganje medplatformnosti navadno ni dobro dokumentiran, hkrati pa uporabniki tvegajo, da enkrat v prihodnosti storitev za grajenje aplikacije ne

bo več na voljo.

Izmed vseh obravnavanih metod smo si izbrali štiri, ki smo si jih ogledali bolj podrobno. Na izbor teh metod so vplivale zahteve posamezne aplikacije. Najbolj dovršena izmed vseh metod je bila Unity. Grajenje aplikacij za različne platforme je potekalo brez zapletov, pri vseh drugih metodah pa je bilo potrebno odpravljati napake in skrbno slediti navodilom v priloženi dokumentaciji. Z metodo Unity je tudi sam razvoj potekal zelo gladko zaradi zelo dobrega urejevalnika scene. Druge metode so v primerjavi mnogo bolj primitivne.

Kljub primitivnosti so tudi druge metode vredne ogleda. WebGL skupaj s knjižnico THREE.js omogoča zelo hitro prototipiranje aplikacij. Problem je le v slabi podprtosti WebGL standarda tako na namiznih računalnikih kot tudi na mobilnih napravah.

Grajenje medplatformnih aplikacij v programskem jeziku Java s knjižnico PlayN je bilo dokaj enostavno. Orodja za grajenje na različnih platformah so enostavna za uporabo in navadno zahtevajo samo en dodaten ukaz. Prednost pri uporabi te metode je tudi vroče izmenjevanje izvorne kode, ki močno pohitri razvoj.

Primer aplikacije napisane v jeziku C++ je bil mogoče še najbolj zahteven. Razlog za to je uporaba jezika C++, ki je nekoliko bolj primitiven, kot vsi ustali uporabljeni programski jeziki. Tudi proces grajenja je z uporabo orodij cmake in make nekoliko bolj zahteven, kot pri ostalih metodah. Prednost te metode je hitrost izvajanja. Na noben drug način na različnih platformah ne dobimo take hitrosti kot z domorodno C++ aplikacijo.

Poleg tradicionalnih metod smo si v diplomskem delu ogledali tudi povsem nov pristop: pisanje medplatformne aplikacije za grafično procesno enoto namesto za centralno procesno enoto. Prednost te metode bi bila potencialna visoka paralelnost izvajanja aplikacije in neposreden dostop do grafične kartice. Za delujoč prototip je potrebno še veliko dela, saj trenutne tehnologije CUDA in OpenCL ne omogočajo vseh potrebnih funkcij. Da bi bil projekt uporaben za končne uporabnike bi bilo pa potrebno premostiti tudi razlike

med grafičnimi karticami posameznih razvijalcev.

V diplomskem delu smo si ogledali tudi projekt ASM.js, ki omogoča prevod C/C++ izvorne kode v podmnožico Javascripta. Spletni brskalniki lahko to podmnožico Javascripta še dodatno optimizirajo. Tako se lahko hitrost izvajanja Javascript aplikacije približa hitrosti izvajanje domorodne aplikacije. Poleg tega je projekt pomemben za nadaljni razvoj brskalnikov, saj so vse možne pohitritve pri prevajanju Javascripta dobro dokumentirane in standardizirane.

Literatura

[1] OpenGL Dostopno na:
http://www.opengl.org/

[2] Direct3D Dostopno na:
http://msdn.microsoft.com/en-us/library/windows/desktop/hh309466(v=vs.85).aspx

[3] Jure Dimec (2002), Medjezično iskanje dokumentov, URL:
http://clir.craynaud.com/clir/MEDJEZICNOISKANJEDOKUMENTOV.pdf

[4] Android Dostopno na:
http://www.android.com/

[5] iOS Dostopno na:
http://www.apple.com/ios/

[6] Windows Phone Dostopno na:
http://developer.windowsphone.com/en-us

[7] ANGLE Dostopno na:
https://code.google.com/p/angleproject/

[8] Firefox OS Dostopno na:
http://www.mozilla.org/en-US/firefox/os/

[9] Ubuntu Dostopno na:
http://www.ubuntu.com/phone

[10] Sailfish OS Dostopno na:
https://sailfishos.org/

[11] Rasperry Pi Dostopno na:
http://www.raspberrypi.org/

[12] Beagle Bone Dostopno na:
http://beagleboard.org/bone

[13] Canvas Element W3C Dostopno na:
http://www.w3.org/wiki/HTML/Elements/canvas

[14] WebGL - Khronos Goup Dostopno na:
http://www.khronos.org/webgl/

[15] Primerjava izvajanja Javascript in domorodnih aplikacij Dostopno na:
http://www.html5rocks.com/en/mobile/nativedebate/

[16] V8-gl izvorna koda Dostopno na:
https://github.com/philogb/v8-gl

[17] LeechyJS Dostopno na:
http://martens.ms/lycheeJS/

[18] Xamarain Dostopno na:
http://xamarin.com/

[19] libGDX Dostopno na:
http://libgdx.badlogicgames.com/

[20] PlayN Dostopno na:
http://code.google.com/p/playn/

[21] Maven Dostopno na:
http://maven.apache.org/

[22] Unity Dostopno na:
http://unity3d.com/

[23] Programski jezik Haxe Dostopno na:
http://haxe.org/

[24] Gameplay Dostopno na:
http://www.gameplay3d.org/

[25] OGRE Dostopno na:
http://www.ogre3d.org

[26] Marmalade Dostopno na:
http://www.madewithmarmalade.com/

[27] Adobe Flash Dostopno na:
http://www.adobe.com/products/flash.html

[28] ASM Dostopno na:
http://asmjs.org/

[29] CUDA Dostopno na:
https://developer.nvidia.com/what-cuda

[30] CUDA na mobilnih napravah Dostopno na:
http://blogs.nvidia.com/blog/2013/07/24/kepler-to-mobile/

[31] OpenCL Dostopno na:
http://www.khronos.org/opencl/

[32] AccelerEyes Dostopno na:
http://www.accelereyes.com/products/mobile