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

Abstract

1 Uvod **1**

1.1 Potek diplomske naloge 1

2 Pregled platform **3**

2.1 Namizni in prenosni računalniki 3

2.2 Mobilne platforme 6

2.3 Računalniki na eni plošči 10

2.4 Skupne zmogljivosti 11

3 Metode **13**

3.1 Spletne aplikacije 13

3.2 Spletne aplikacije z V8-gl 19

3.3 Xamarin 20

3.4 LibGDX 21

3.5 PlayN 22

3.6 Unity 24

3.7 Programski jezik Haxe 24

3.8 Razvoj medplatformnih aplikacij v programskem jeziku C++ . 25

3.9 Marmalade 26

3.10 Monogame 26

3.11 Adobe Flash 27
3.12 QT 28

4 Asm.js **29**

5 Programiranje na grafičnem čipu **31**
5.1 CUDA 32
5.2 OpenCL 32
5.3 AMP 33

6 Študije primerov **35**
6.1 Primer za učenje jezikov 35
6.2 Primer štiri v vrsto 39
6.3 Primer MIN (WebGL) 41
6.4 Primer OGRE (C++) 43

7 Sklepne ugotovitve **47**

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 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 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 tako odprto kodne kot plačljive metode za premostitev razlik med platformami. Preučili bomo metode, ki temeljijo na spletnih tehnologijah in prevodih med različnimi programskimi jezikih. Zanimale nas bodo metode, ki temeljijo na uporabi jezika C++, ter metode, ki ponujajo celotno razvojno okolje. Ogledali si bomo tudi metodo, ki uporablja namenski programski jezik.

V zadnjem delu diplomske naloge bomo opisali nekaj primerov grafično intenzivnih aplikacij. Razložili bomo uporabljene metode, njihove prednosti in slabosti.

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 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 ukazov in slabše strojne zmogljivosti, paziti tudi na porabo baterije.

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 za centralno procesno enoto. Procesorji v namiznih računalnikih so danes večinoma zgrajeni s CISC arhitekturo (procesorji Intel, AMD), medtem ko mobilne naprave večinoma uporabljajo RISC arhitekturo in procesorje podjetja ARM in nVi-

dia. 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. Vse operacije povezane z izrisovanjem objektov na zaslon, se danes izvajajo na grafičnih procesnih enotah, ki imajo za ta namen visoko stopnjo paralelizma.

Pri prenosnih računalnikih je potrebno paziti na baterijo, podobno kot pri mobilnih napravah. Prenosni računalniki so načeloma po zmogljivostim podobni namiznim računalnikom, uporabljajo tudi enako arhitekturo za centralno procesno enoto. Grafične procesne enote se na prenosnih računalnikih navadno šibkejše kot na namiznih računalnikih. Po večini je bil narejen kompromis med zmogljivostjo grafične kartice in porabo energije.

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 [1] je odprt standard, ki ga razvija skupina Khronos. 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 pojavila prvič pojavila v posodobitvi 3.0. Pred tem se se je uporabljal fiksen funkcijski cevovod. OpenGL za namenske sisteme (OpenGL ES) je v prvi verziji uporabljal fiksen funkcijski cevovod, v reviziji 2.0 pa ga je zamenjal programiran vhod sličen OpenGL 3.0.

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 jeziki za pisanje programov na grafični kartici in njihovi prevajalniki se lahko med seboj razlikujejo.

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 bile kot dodaten sladkor pri prikazovanju. To je ustvarilo resen problem, kjer so razvijalci grafičnih aplikacij morali prilagajati kodo aplikacij glede na posamezne grafične kartice.

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 posebne 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 se je bil že zakoreninjen 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 vmesnih 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 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 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 Linux jedru.

2.2.1 Android

Android [3] 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 namenjen 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 [4] 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 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, dvo jederni 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 (objektni C). 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, Objective C nima avtomatičnega sproščanja pomnilnika, kar pri razvoju zahtevnih aplikacij lahko štejemo kot prednost, saj ima programer na voljo močnejša orodja za delo s 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 [5] 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 te dva jezika v kasnejših verzijah kar precej razlikujeta. Lep primer je implementacija generikov (slo. prevod?), ki je 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 [6] je kratica, ki pomeni Almost Native Graphics Layer Engine, pogon s skoraj domorodno grafično plastjo.

Angle je odprto kodni projekt, ki implementira OpenGL ES 2.0 specifikacijo in jo strojno pospeši z Direct3D vmesnikom. Angle je v uporabi 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 [7] 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 namiznih računalnikih.

Ker podpora temelji na standardih za spletne tehnologije ni podpore za OpenGL ES1.x, ki je na voljo na drugih mobilnih operacijskih sistemih. Podprt pa je 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[2.3.1].

2.2.5 Ubuntu

Operacijski sistem Ubuntu [8] 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.

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.

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 namesto jezika QMLa priporoča C++.

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 v Ubuntuju.

2.2.6 Sailfish OS

Sailfish [9] je operacijski sistem temelječ na Linux jedru, namenjen mobilnim telefonom in drugim napravam. Podobno kot operacijski sistem Ubuntu bo tudi Sailfish uporabljal QML in Qt [3.12] za razvoj domorodnih aplikacij.

Tudi kar se tiče podpore za razvoj grafično intenzivnih aplikacij je slično Ubuntu mobilnemu operacijskemu sistemu. Dostop do OpenGL programskega vmesnika je mogoč s pomočjo QT pogleda. Razlike med Ubuntu in Sailfish so minimalne, tako da bomo lahko teoretično aplikacijo pisali za oba sistema brez večjih sprememb.

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 [10]. 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 ES2.0 in je kljub omejenim pomnilnikom zmožen poganjati zahtevnejše aplikacije. Lep primer je računalniška igra Minecraft.

2.3.2 BeagleBone

Podobno kot Raspberry Pi je tudi BeagleBone [11] 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 tako 3D kot 2D OpenGL aplikacije.

2.4 Skupne zmogljivosti

Kot lahko vidimo iz navedenih primerov platform se ne moremo odločiti za en sam programski jezik in en sam vmesnik API za risanje, s katerima bi pokrili vse možne platforme. Največji del trga lahko pokrijemo, če se odločimo za OpenGL ES 2.0 programski vmesnik s programskim jezikom C++, saj s tem lahko razvijamo grafično intenzivne aplikacije na Wndows, Linux in Mac OSX, kot tudi na mobilnih platformah Android, iOS, Ubuntu, Sailfish, BlackBerry... Kot bomo videli v nadaljevanju obstajajo tudi možnosti kako razširiti podporo.

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 sistemih 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 tak ali drugačen 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

3.1 Spletne aplikacije

Programski jezik Javascript je nastal okoli leta 1995, ko je bil tudi vključen v spletni brskalnik Netscape, takrat se pod drugim imenom. Jezik Javascript je omogočil izvedbo kode na uporabnikovi strani (angl. client-side) in spletne strani so lahko postale interaktivne.

V zgodnjih dneh spleta se programiranje na uporabnikovi strani ni prav veliko uporabljalo, Javscript interpreterji so bili počasni, zaradi neenotnih standardov med brskalniki pa je bilo pisanje aplikacije, ki be 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 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. 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, 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 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) [12].

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 iz udobja namiznega računalnika. 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 lahko traja tudi 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 [13] 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 (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 probleme. 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 je zelo težko, 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 dvodimenzionalnim 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 (angl. shader program).

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, 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 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 za to 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.

3.1.3 Zvok

Aplikacije napisane v dvodimenzionalnem 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 za predvajanje zvoka. 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.

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

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. V industriji 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 [14].

S pomočjo tehnologij, kot je ASM.js 4 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 [15] 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.

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 po več platformah, kot ga danes ponujajo različni spletni brskalniki.

3.2.1 LycheeJS

Najbolj perspektivna uporaba V8-gl knjižnice je trenutno projekt LycheeJS [16]. 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 aplikacije.

3.3 Xamarin

Xamarin [17] nam omogoča, da aplikacijo napišemo v programskem jeziku C#, do programskih vmesnikov domorodnih platform pa dostopamo preko posebne knjižnice. C# prevajalnik našemu programu doda .NET rutino

(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, vendar to danes ne predstavlja večjega problema.

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, sproščanje pomnilnika...

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, saj tako LibGDX [3.4] kot PlayN [3.5] uporabljata Xamarin za grajenje iOS aplikacij.

3.4 LibGDX

Libgdx [18] je v Javi napisano ogrodje za medplatformni razvoj grafičnih aplikacij. Knjižnica abstrahira razlike med namiznimi aplikacijami, Androidom, iOSom in HTML5 ter gradi na odprtih standardih, kot je OpenGL ES/WebGL.

LibGDX omogoča 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. Saj nam ni potrebno vsako spremembo preveriti tudi na mobilnih napravah. Grajenje iOS

ali Android paketa namreč traja nekaj časa, 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 ES 2.0 standarda. 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 Javinega domorodnega vmesnika JNI (angl. Java Native Interface), v programskem jeziku C++. S tem se kritični deli izvajajo še hitreje, kot če bi bili napisani v Javi.

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

Za grajenje iOS paketa je potrebno orodje Xamarin [3.3]. LibGDX s pomočjo LLVM Java kodo spremeni v C# kompatibilno, 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 [19] je ogrodje za razvoj grafično intenzivnih aplikacij na različnih 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 ogrodju LibGDX. Našo aplikacijo lahko razvijamo na namiznem računalniku, kjer lahko uporabljamo tudi vroče izmenjavanje kode.

Za izvoz na različne platforme lahko uporabimo orodje Ant ali Maven. Večjih razlik med pristopoma 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 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 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 tudi odstrani mrtvo kodo, optimizira nize znakov in metode pretvori v enovrstično varianto.

3.5.2 iOS

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 dvodimenzionalni programski vme-

snik je nekoliko lažji za uporabo. Delo z 3D programskim vmesnikom (in dostop do domorodnih OpenGL ES funkcij) pa je precej lažje z uporabo LibGDX knjižnice.

3.6 Unity

Unity3D [20] 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).

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 objetkov v sceno, spreminjanje njihovih nastavitev, določanje materialov in drugih nastavitev. Drugo stanje - stanje igranja - pa simulira potek izvajanja grafičnega programa in razvijalcu omogoča hiter pregled kako bo aplikacija delovala, ko se bo izvozila na eno izmed podprtih platform.

3.7 Programski jezik Haxe

Programski jezik Haxe [21] 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 Java, kar pokrije večji del platform tudi iOS in Android.

Jezik Haxe temelji na jeziku C in se zaradi podobnosti drugim jezikom (Java, Javascript, ActionScript) ni težko učljiv. Haxe je strogo tipiziran jezik, 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 odprte 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 (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), ne pa tudi do elementov za uporabniški vmesnik. Le te 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].

3.8.1 OGRE

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

Prednost OGRE pred drugimi podobnimi projekti je, da ima zelo dobro dokumentacijo. 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.

Ogrodje je stabilno in se še vedno aktivno razvija.

3.9 Marmalade

Marmelade [23] 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.

Za razliko od drugih orodij Marmalade ni na voljo brezplačno, preizkusimo lahko samo 30 dnevno verzijo, potem pa moramo kupiti licenco. 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.

3.10 Monogame

Monogame je orodoje, ki omogoči poganjanje grafično intenzivnih aplikacij narejenih za Windows in Windows Phone, na drugih platformah. Trenutno podprti sistemi so OSX, Linux, iOS, Android, Play Station Mobile in Ouya.

Z verzijo 3.0.0 je bila dodana podpora do 3D programskega vmesnika. MonoGame želi popolnoma podpreti XNA 4 vmesnik API. Za delujočo aplikacijo na iOS in Android se uporablja Xamarin platformo[3.3], kar pomeni,

da moramo kupiti dve licenci. Verzija 3.0.0 se osredotoča na funkcije, ki jih prinaša OpenGL ES2.0 medtem ko so prejšnje verzije temeljile na OpenGL ES 1.X. Na Windows sistemih se namesto OpenGLa uporablja Direct3D.

3.11 Adobe Flash

Adobe Flash [24], 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 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 teh 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.

3.11.1 Stage3D

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

3.11.2 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.

3.12 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 [25] je raziskovalni projekt pri Mozilli, ki definira podmnožico JavaScripta. Cilj te podmnožice, 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 v C++.

Prevajanje v JavaScript ni nič novega. 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 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 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 (Google) 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 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 v naprej (angl. ahead of time) in ne samo ob pravem času (angl. just in time).
- 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 uporablja 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 pod množica JavaScripta lahko že danes teče v vseh brskalnikih, vendar ni nobene garancije, da bo na vseh brskalnikih vidne pohitritve.

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.

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 zahtevna operacija, se pojavlja nova alternativa - pisanje aplikacij direktno na grafičnem čipu. Taka aplikacija lahko s kompatibilno grafično kartico dela na več platformah - premestiti bi bilo 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 pod pokrov. Tak grafični pogon je trenutno samo 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 sta nVidiina CUDA, Microsoftov AMP in OpenCL.

5.1 CUDA

CUDA [26] 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 - fizika, biologija, kemija...

5.1.1 Logan - CUDA na mobilnih platformah

Podjetje nVidia je 24. julija 2013 [27] 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), 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

Open Computing Language [28] je ogrodje za pisanje programov, ki se izvajajo na heterogenih platformah, ki so sestavljena iz večih centralno procesnih enot, grafičnih procesnih enot, digitalnih procesorjev signalov in drugih procesorjev. OpenCL je odprti 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 [29], 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 Microsoftofa 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 od treh tujih sta naključno izbrani, tretja pa je pravilni odgovor.

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, tako da se ima uporabnik možnost naučiti besedo. Ko si uporabnik ogleda pravilni odgovor se igra ponastavi ter začne od začetka.

Na sliki 6.1 je 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. Aplikacije mora biti sposobna izrisovati velik nabor različnih abeced.

Slika 6.1: Primer delovanja Nemške verzije na mobilni napravi

Slika 6.2: Primer delovanja Korejske verzije na namiznem računalniku

Aplikacija je bila zamišljena kot mobilna aplikacijam za platformi iOS in Android, vendar je zelo uporabno imeti možnost razvoja in izvajanja na namiznem računalniku. Zato smo se odločili za metodo PlayN.

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.
- Orodje 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 napisana in razumljiva.

Izbira je potekala med PlayN in podobno knjižnico LibGDX. Na koncu smo se odločili za PlayN zaradi lažje uporabe lastnih pisav v aplikaciji. O uporabi plačljivih rešitev nismo razmišljali.

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*.

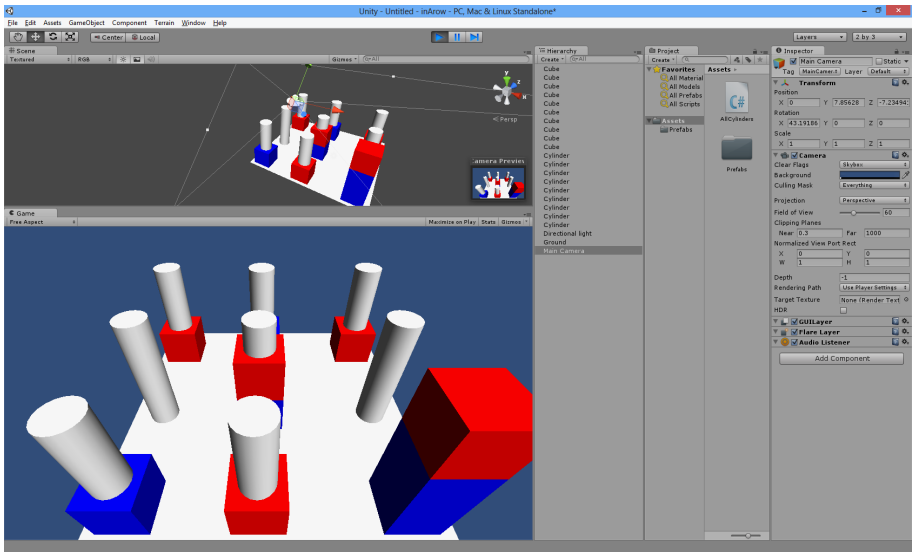
6.1.4 Prednosti izbora metode

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

6.1.5 Slabosti izbora metode

Težave smo imeli pri testiranju 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.

Nekaj težav je povzročil tudi prehod na novo verzijo iz 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.



Slika 6.3: Razvoj grafično intenzivne aplikacije v okolju Unity.

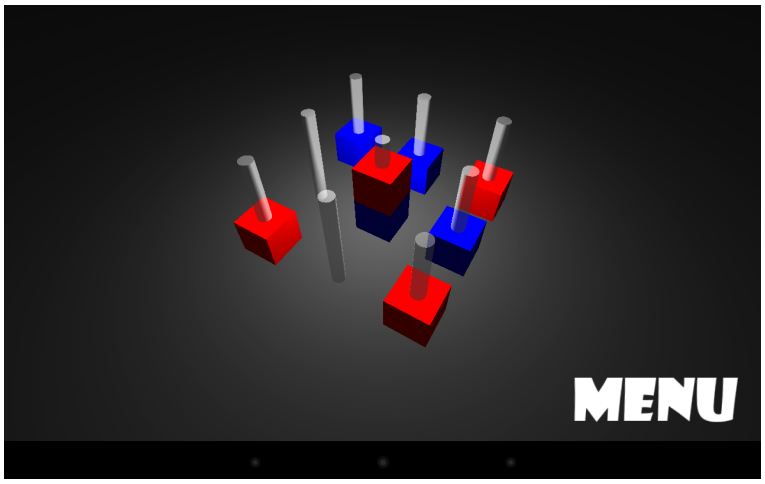
6.2 Primer štiri v vrsto

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.

6.2.1 Uporabljena metoda

Za razvoj smo se odločili za orodje Unity. Unity za razliko od ostalih metod, ki smo jih ogledali, Unity vključuje svoje integrirano razvojno okolje, ki ga prikazuje slika 6.3. Urejevalnik nam omogoča grajenje objektov, postavitev kamere in določitev luči. Za postavitev objektov v 3D prostor se uporablja okno z ortografsko ali perspektivno projekcijo ter različnimi možnimi pogledi. Slika 6.4 prikazuje delovanje aplikacije na mobilni platformi Nexus 7.

S pomočjo uporabniškega vmesnika določimo pozicije objektov, materi-



Slika 6.4: Končana Unity aplikacija izvožena na mobilno platformo.

ale in druge lastnosti. Za logiko, obnašanje in odziv na uporabniški vhod pa vsakemu objektu lahko dodelimo tudi skriptno datoteko. V tej skriptni datoteki v enem od podprtih programskih jezikih (C#, JavaScript, Boo) določimo kako se bo objekt odzival na uporabniški vhod in določimo kako se bo obnašal v času, ko je viden na zaslonu.

Aplikacija je zasnovana preprosto. Ko se igra zažene se ustvarijo prazni valji, na katere se nastavi dogodek *addBlock*, ki se sproži ob dotiku na valj. Unity okolje avtomatično prevede koordinate zaslona v koordinate prostora in pravilno preveli, katerega stolpa smo se dotaknili. Prav tako orodje Unity abstrahira način dotika, le ta je lahko klik z miško ali pa 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 novo potezo pojavila zmagovalna kombinacija.

Če se je igra končala, 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.

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.

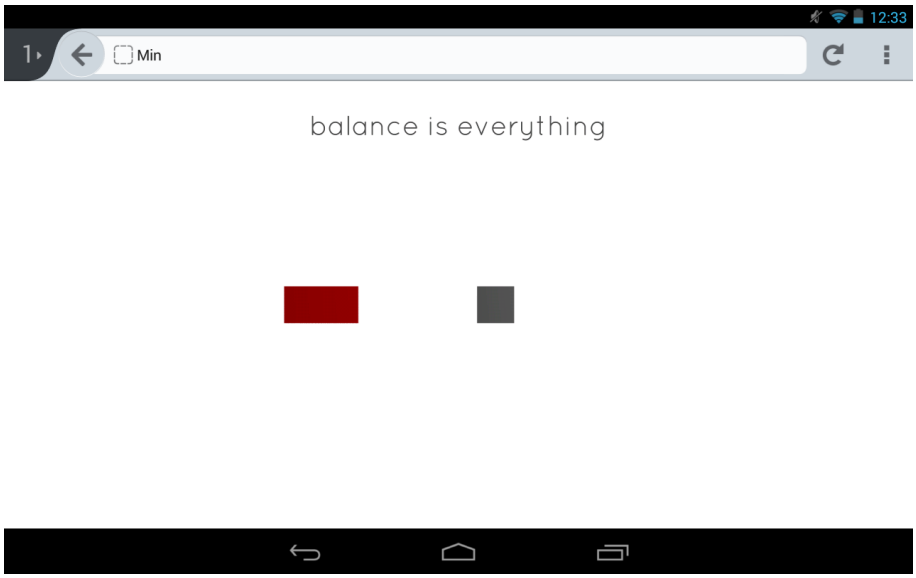
6.2.3 Slabosti metode

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 (WebGL)

Primer MIN je preprosta igra z ortografsko projekcijo. Cilj igre je odstraniti rdeče kocke s sivo kocko, ki jo premika igralec. 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 pa se lahko premikamo navzgor in navzdol, ne vidimo pa več katere kocke so levo in katere desno od nas. Če se med spreminjanjem perspektive siva kocka dotakne črne črte, se igrana stopnja ponastavi. Slika 6.6 prikazuje delovanje končane aplikacije v mobilnem brskalniku.



Slika 6.5: Delovanje igre v mobilnem brskalniku Firefox.

6.3.1 Uporabljena metoda

Aplikacije smo se razvili kot spletno aplikacijo, ki uporablja tehnologijo WebGL. WebGL omogoča strojno pospeševanje, kar pomaga pri animaciji za preklop med perspektivami. Ta uporabniku daje občutek 3D prostora. Razen te animacije, bi lahko vse ostale dele aplikacije brez problema realizirali z uporabo 2D platna.

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, vse potrebno lahko opravi z Javascriptom. Pisanje lastnih GLSL

programov je vseeno možno.

6.3.3 hammer.js

Hammer.js je Javascript knjižnica za zaznavanje gest. Okoli standardnih dogodkov, ki se v brskalnikih sprožijo na dotik zgradi lastne funkcije, ki posamezne dogodke znajo združiti v geste. Tako uporabniku ni potrebno pisati izvorne kode, ki iz posameznik dogodkov na primer zazna gesto za poteg v desno.

Knjižnica omogoča naročanje na dogodke tipa poteg desno. Primer izvorne kode, ki se naroči na ta dogodek vidimo v Primeru **??**.

Hammer(element).on(šwipeleft", function(event) ...);

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

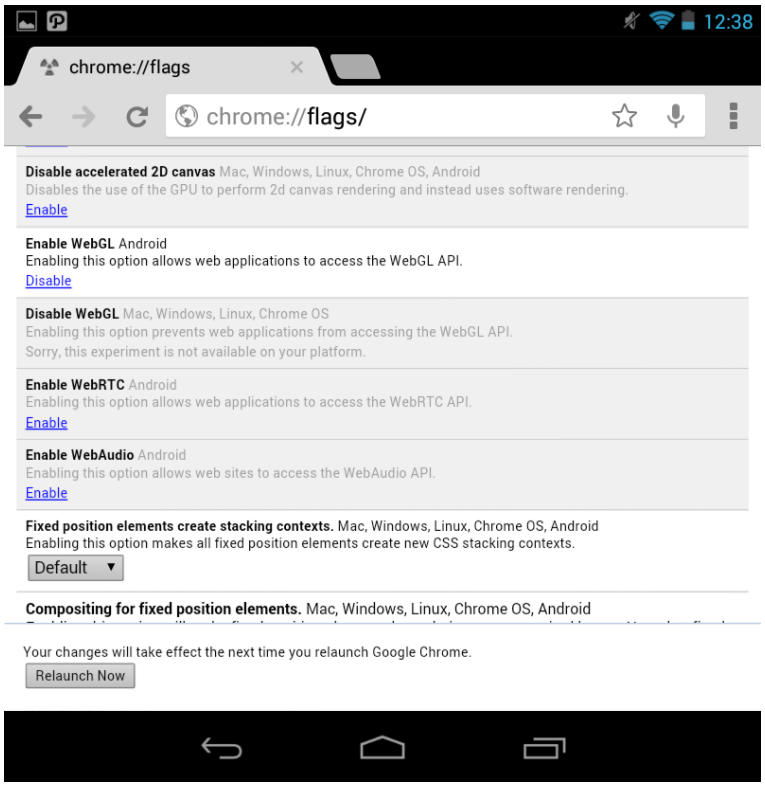
Knjižnica omogoča tudi nastavitve posameznih gest, kot so dolžina pritiska potrebna za dogodek daljši pritisk, ...

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, s kodo, ki zaznava dotik dogodke.

6.3.5 Slabosti izbora metode

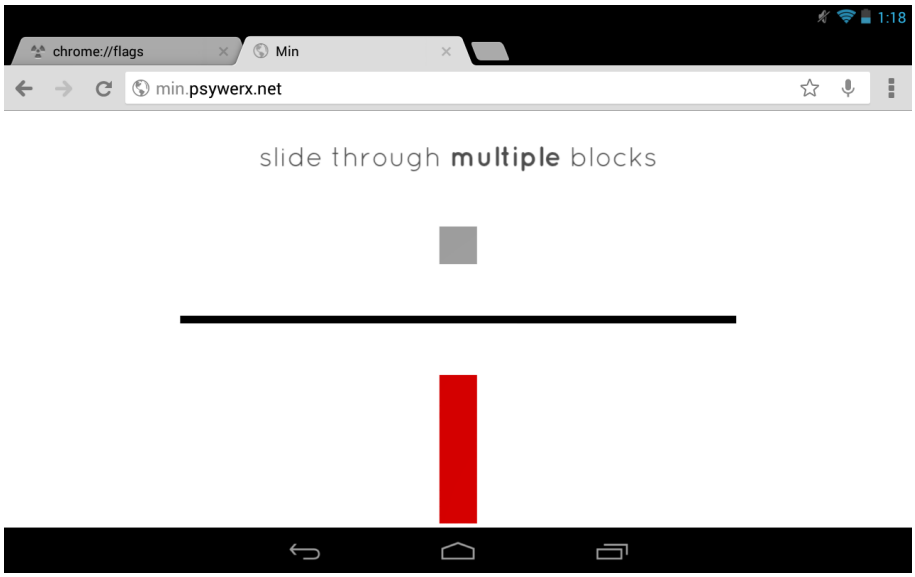
Glavna slabost metode je slaba podpora na mobilnih brskalnikih. Mobilni brskalnik Firefox je sicer prikazal našo aplikacijo pravilno, vendar smo morali za delovanje v brskalniku Chrome omogočiti posebno nastavitev. Slika 6.5 prikazuje zaslon, kjer se nastavi zastavica za omogočanje WebGLa. Po



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

vključimo zastavico aplikacija deluje pravilno, kot kaže tudi 6.7. Dokler podjetje Google privzeto ne omogoči te zastavice, WebGL aplikacije ne bodo primerne za navadne uporabnike.

6.4 Primer OGRE (C++)



Slika 6.7: Delovanje igre v mobilnem brskalniku Chrome.

Poglavje 7

Sklepne ugotovitve

Sklep.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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