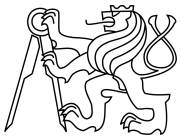
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Port FurryBall na OS X
Student:	Jakub Mirovský
Vedoucí:	Ing. Jiří Chludil
Studijní program:	Informatika
Studijní obor:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2016/17

Pokyny pro vypracování

Aplikace FurryBall je velmi výkonný renderer od společnosti AAA-studio (zadavatel) v současné době fungující pod OS WIN. Cílem práce je port aplikace pod OS X.

1. Analyzujte rozdíly mezi OS Windows a OS X, zaměřte se na problematiku spouštění aplikace psané v jazyce C a C++.
2. Analyzujte současnou implementaci aplikace FurryBall a detekujte části kódu, které závisí na hostujícím OS.
3. Pomocí metod softwarového inženýrství navrhněte proces tzv. portace z jednoho OS do druhého. Zaměřte se zejména na:
- dekomponování aplikace na samostatné bloky,
- proces reimplementace,
- proces testování.
4. Navrženou portaci ověřte převedením několika částí aplikace FurryBall pod OS X.

Seznam odborné literatury

Dodá vedoucí práce.

L.S.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 9. února 2016

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalářská práce

Port FurryBall pro OS X

Jakub Mirovský

Vedoucí práce: Ing. Jiří Chludil

29. června 2016

Poděkování

V první řadě děkuji mému vedoucímu práce Ing. Jiřímu Chludilovi za trpělivost, hodnotné rady a připomínky. Dále děkuji Ing. Janu Smejkalovi za precizní vysvětlení kódu a trpělivost při mých neustálých dotazech. Na závěr děkuji svým blízkým a rodině za podporu při celém mém studiu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 29. června 2016 .....

Odkaz na tuto práci

Mirovský, Jakub. *Port FurryBall pro OS X*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Tato bakalářská práce si dává za cíl provést analýzu současné implementace realtimové renderovací aplikace FurryBall, najít místa závislá na hostujícím operačním systému Windows a navrhnout proces postupného přepisu na operační systém OS X. Následně navržený proces portace ověřit přepisem části aplikace FurryBall. Výsledkem práce je spustitelný program a ukázka renderu na operačním systému OS X.

Klíčová slova Aplikace, Windows, OS X, analýza, port aplikace, počítačová grafika, vykreslování, stabilita, rychlost, C++, CUDA, OptiX.

Abstract

The goal of this bachelor thesis is to analyze current implementation of real-time rendering application FurryBall, find pieces of code dependent of hosting operation system Windows and propose process of rewriting to operation system OS X. After that verify the proposed process by rewriting part of the FurryBall application. Result of this thesis is executable program and picture of render on operating system OS X.

Keywords Application, Windows, OS X, analysis, port aplikace, computer graphics, rendering, stability, speed, C++, CUDA, OptiX.

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Tato bakalářská práce se zabývá analýzou, návrhem a implementací aplikace FurryBall na operační systém OS X. Aplikace FurryBall je v současné době napsaná v programovacím jazyce C++ na platformě Windows, která není kompatibilní s OS X. FurryBall je kvalitní grafický renderer s pluginy pro programy Autodesk Maya a Cinema4D.

V první části se zaměříme na analýzu rozdílů mezi oběma operačními systémy, porovnáme jejich rozdílnosti a společné prvky. V další kapitole navrhneme obecný proces pro port aplikace z platformy Windows na jinou platformu, v našem případě OS X. V implementační části tento proces ověříme přepsáním části aplikace FurryBall a na závěr se zaměříme na testování. Převážně na používané jednotkové testování a navrhneme integrační, regresní, QA testování.

Existuje velké množství grafiků pracujících na operačním systému OS X, kteří v současné době nemají možnost využívat FurryBall ke své práci. Cílem této práce je ověřit, zde je možné zpřístupnit FurryBall i uživatelům OS X a rozšířit tím klientelu, která má možnost tuto aplikaci používat.

Kapitola 1

Analýza

1.1 Výběr IDE

Základní množina programů pro psaní počítačových aplikací obsahuje textový editor, kompilátor a emulátor terminálu. Při dobré znalosti programovacího jazyka není problém takto napsat jednodušší aplikaci, ale i tak je snadné udělat typografickou chybu, chybu v podmínce nebo mít problémy s pamětí. Tyto chyby se obtížně hledají bez pomocných programů, takzvaných debuggerů. Z tohoto důvodu byly vytvořeny komplikovanější aplikace, integrovaná vývojová prostředí (IDE), která kombinují veškerou potřebnou funkcionalitu. Prvním krokem k úspěšnému napsání jakékoliv aplikace je výběr vhodného IDE.

V této kapitole jsou porovnána preferovaná IDE na každé platformě a dvě vybraná z multiplatformních. Na platformně Windows je preferované IDE Microsoft Visual Studio[29], základní IDE na platformě OS X je Xcode[7] a jako multiplatformní IDE je vybrán Clion[20] od společnosti Jetbrains a NetBeans[35] od společnosti Oracle.

1.1.1 Metodika porovnání

Každé IDE je v krátkosti představeno a ohodnoceno v níže uvedených kategoriích. V každé kategorii může IDE získat od 0 do 10 bodů. Všechny kategorie jsou hodnoceny vzhledem k programovacímu jazyku C/C++ ve standardu C++11 a vývoji pro platformu OS X. Tyto kategorie jsou:

- **Multiplatformnost.** Platformy, na kterých se dá IDE použít.
- **Dokumentace.** Úroveň poskytované dokumentace ke každému IDE.
- **Návaznost.** Návaznost na současnou implementaci projektu.
- **Kompilátor.** Kvalita použitého kompilátoru, jeho výhody a nevýhody a dostupnost na platformě OS X.

- **Debugger.** Kvalita použitého debuggeru, jeho přednosti, nevýhody a kvalita výstupu.
- **Funkce.** Popis funkcí textového editoru v IDE.
- **Jazyky.** Podporované programovací jazyky.
- **User experience.** Přehlednost uživatelského rozhraní, pocitová rychlost programu a jednoduchost IDE.

1.1.2 Microsoft Visual Studio

Microsoft Visual Studio je kvalitní sada nástrojů pro vývoj desktopových, mobilních a webových aplikací, převážně pro platformu Windows.

Visual Studio je vývojové prostředí pouze pro platformu Windows. Z tohoto důvodu se nehodí pro port aplikace FurryBall, ale je zmíněn kvůli zpětné kompatibilitě a testování, zda je nově přepsaná aplikace přeložitelná a spustitelná na platformě Windows.

Microsoft poskytuje velice kvalitní dokumentaci pro všechny dostupné jazyky ve Visual Studiu. Dokumentace je přehledně napsaná, s kvalitním uživatelským rozhraním a dobrým vyhledáváním a filtrováním.

Kompilátor použitý ve Visual Studiu je kvalitní kompilátor pro psaní aplikací pro platformu Windows. Ač je to kvalitní kompilátor, podporuje pouze některé části novějších C++ standardů a v některých případech se jeho chování oproti ostatním kompilátorům velice liší. Kvůli úzké specifikaci na platformu Windows není kompilátor vhodný pro psaní multiplatformních aplikací.

Visual Studio obsahuje jeden z nejkvalitnějších existujících debuggerů pro C a C++. Debugger úzce spolupracuje s kompilátorem a poskytuje tak velice kvalitní výstup. Dokáže zobrazovat hodnoty, stavy proměnných za běhu, doby běhu funkcí pro profilování programu, kontrolovat využívanou paměť a přehledně zobrazovat všechna běžící vlákna.

Textový editor Visual Studio obsahuje velké množství funkcí zjednodušujících práci s kódem, tj. přehledný editor s možností zvýrazňování syntaxe, doplňování kódu, generování definic a deklarací tříd a funkcí. Visual Studio má implementované pokročilé vyhledávání jak v textu, tak i v projektu.

Editor, kompilátor a debugger ve Visual Studiu podporuje velké množství programovacích jazyků od C a C++ přes C#, F#, Visual Basic až po webové technologie jako HTML, CSS a Javascript.

Visual Studio je na první pohled zbytečně komplikované, ale po počátečním zorientování se s ním pracuje intuitivně a jednoduše. Všechny důležité funkce jsou dostupné pod klávesovou zkratkou zrychlující práci a v kontextových nabídkách se dostaneme ke všem ostatním. Možnost přesunout a rozdělit okna přidává na přehlednosti a rychlosti práce.

1.1.3 Xcode

Xcode je vývojářská sada od společnosti Apple pro vývoj desktopových aplikací pro OS X a mobilních pro iOS.

Xcode se zaměřuje na ekosystém od společnosti Apple, proto je toto prostředí dostupné pouze na operačním systému OS X. Mezi Apple vývojáři patří k oblíbeným IDE s možností psaní kódu v několika jazycích. V případě této práce není vhodné z důvodu jeho zaměření pouze na OS X a převážně na jazyky Objective-C a Swift.

Apple poskytuje pro Xcode solidní dokumentaci s popsanými postupy a návody. Část dokumentace je společná pro všechny jazyky, ale většina z ní je zaměřena na preferovanější jazyky Objective-C a Swift. Z tohoto důvodu není dokumentace příliš přívětivá k psaní v C a C++.

Xcode používá pro kompilování C, C++ a Objective-C kódu Clang[24], který je moderní kompilátor postavený na projektu LLVM s velice rychlou kompilací. LLVM je kolekce moderních a znovupoužitelných technologií. Jednou z hlavních součástí je jádro LLVM (LLVM Core), které poskytují optimalizaci výsledného kódu, čehož Clang využívá.

Pro debuggování kódu používá Xcode další z součástí LLVM projektu, a to LLDB[23], které je moderní a velice rychlý debugger s podporou jazyků C, C++ a Objective-C. LLDB podporuje širokou škálu ladících funkcí jako zobrazování hodnot a stavů proměnných za běhu, krokování programu, zásobník volání funkcí.

Editor kódu v Xcode obsahuje velké množství pomocných funkcí pro psaní kódu, tj. zvýrazňování syntaxe, inteligentní doplňování, generování definic a deklarací tříd a funkcí. Vyhledávání textu v souborech v celém projektu je samozřejmostí.

Mezi programovací jazyky podporované Xcodem patří C, C++, Objective-C, Java a Swift. Dále podporuje skriptovací jazyky jako AppleScript, Python a Ruby.

Xcode je od začátku velice přehledné IDE. První projekt je možné napsat velice rychle, ale složitější funkce a úpravy jsou pro nezkušeného uživatele hůře a pomaleji řešitelné. Po naučení základních zkratek a rozložení se práce zrychlí, ale je potřeba se IDE, pro pochopení všech funkcí, chvíli věnovat.

1.1.4 CLion

CLion je multiplatformní vývojářské prostředí od české společnosti JetBrains založené v roce 2000. Jedná se o velmi mladé IDE, které má velký potenciál v multiplatformním vývoji.

CLion je C/C++ IDE pro platformu Windows, OS X a Linux. Pro sestavení aplikací používá CMake[22]. CMake je sada nástrojů pro sestavení a testování softwaru. Používá se pro kontrolování procesu kompilace díky na platformě nezávislých konfiguračních souborů, ze kterých je možné vygenero-

vat nativní makefily a projekty. Díky této sadě je možné v CLionu vyvíjet pro několik platforem zároveň.

Jelikož je CLion nové IDE, nemá tak obsáhlou dokumentaci jako ostatní starší IDE, ale vše, co je třeba najít, se v ní dá vyhledat. Obsahuje taktéž několik dobrých návodů do začátků, referenci zkratek pro všechny platformy a početnou komunitu na fórech.

CLion podporuje pro kompilaci kódu GCC[16] a Clang, na platformě Windows pak MinGW gcc[32] nebo Cygwin gcc[37]. Neznamená to ale, že stejná část programu napsaná na Linuxu půjde spustit i na Windows. Je třeba brát v potaz rozdíly v jednotlivých systémech již při psaní aplikace.

Pro debuggování kódu používá CLion GDB[17] nebo LLDB na platformě OS X, na platformě Windows pak MinGW gdb nebo Cygwin gdb[38]. Podporuje všechny standardní funkce jako breakpointy, zobrazování hodnot a stavů proměnných za běhu nebo krokování programu.

Editor kódu programu umožňuje zvýrazňovat syntax kódu, doplňovat, generovat třídy, funkce a dokumentaci a vyhledávat v textu a v souborech.

CLion je IDE zaměřené na programovací jazyky C a C++. Jako bonus podporuje značkovací jazyk HTML a XML, CSS a programovací jazyk Javascript.

CLion je přehledné IDE, které stačí zapnout a začít psát. Všechna důležitá funkcionalita je snadno k nalezení a při hledání vždy pomůže dokumentace. Jedinou nevýhodou je občasná pomalejší odezva při velkých souborech. Všechny důležité změny v projektu se automaticky ukládají do přiloženého CMakeLists.txt, který se automaticky načte a neobtěžuje tak programátora. Při znalosti ostatních IDE od společnosti JetBrains nemá uživatel žádný problém, protože se všechny chovají velice podobně.

1.1.5 NetBeans

NetBeans je cross platformní IDE od společnosti Oracle. NetBeans je populární IDE mezi mnoha programátory, ale ohlasy se dělí na dvě skupiny. Jedni na něj nedají dopustit, druzí se od něj drží na míle daleko. V našem případě se jedná o IDE, které běží na všech platformách, což je hlavní rozhodovací faktor.

NetBeans je IDE pro platformu Windows, Linux i OS X napsané v jazyce Java. Pro sestavování programu používá program make, na platformě Windows od projektu Cygwin nebo MinGW, na OS X a Linuxu pak GNU make.

NetBeans má obsáhlou dokumentaci, kvalitní návody a díky oblíbenosti u vývojářů i rozsáhlou podporu na fórech. V dokumentaci se dají rychle najít všechny potřebné informace, kvalitně zpracované pro všechny platformy.

NetBeans používají pro kompilaci kódu několik různých kompilátorů. Na UNIXových systémech jde o GNU gcc a na platformě Windows pak Cygwin gcc nebo MinGW gcc. GNU gcc je kvalitní kompilátor pro GNU operační systémy.

Sice se řadí mezi starší kompilátory, ale je stále v aktivním vývoji. Stejná skupina lidí, kteří tvoří standardy C++ zároveň vyvíjí gcc kompilátor a proto má velice dobrou podporu novějších standardů.

NetBeans používají GNU gdb, MinGW gbd nebo Cygwin gbd pro debuggování kódu. GNU gdb je spolehlivý debugger s velkou škálou funkcí jako nastavení breakpointů, zobrazování hodnot a řadou dalších. Gdb je velice vyspělý debugger, který umí vše, co od něj programátor vyžaduje.

Díky dlouhé době vývoje mají NetBeans velice vyspělý editor kódu s mnoha funkcemi, které usnadňují práci při psaní, tj. zvýrazňování syntaxe, fulltextové vyhledávání v projektu, generování definic, deklarace tříd a funkcí, vytváření dokumentace jsou samozřejmostí.

NetBeans je IDE podporující velké množství jazyků, od nativní Javy, ve které je napsané C, C++, PHP, přes značkovací jazyky jako HTML a XML, po CSS, Javascript a Python.

NetBeans je přehledné IDE, ve kterém je možné se rychle vyznat a zorientovat. Funguje stabilně na všech platformách, ale na OS X s monitorem s vyšším rozlišením se "seká", a tudíž není použitelné.

1.1.6 Výsledné porovnání

Tabulka 1.1: Výsledné hodnocení

	Microsoft VS	**Xcode**	**CLion**	**NetBeans**
Multiplatformnost	0	0	10	10
Dokumentace	9	8	7	9
Kompilátor	8	9	9	8
Debugger	10	9	9	8
Funkce editoru	10	7	8	8
Jazyky	10	8	7	9
User experience	8	8	10	7
Výsledné body	**55**	**48**	**60**	**59**

Z výše uvedeného porovnání je zantelný pouze nepatrný rozdíl mezi jednotlivými vývojářskými prostředími. Pokud bychom nebrali v potaz vývoj na více platforem, vyhrává Microsoft Visual Studio. V našem případě je ale důležitá podpora více platforem a z toho důvodu má navrch CLion a NetBeans. CLion vyhrává z důvodu kvalitnějšího kompilátoru, debuggeru a editoru na platformě OS X. Používá Clang a LLDB, které jsou rychlejší než gcc a gdb a poskytují stejně kvalitní a optimalizovaný výstup.

Na základě tohoto porovnání byl vybrán CLion jako IDE, ve kterém je psaná tato bakalářská práce. Bude použit CMake jako build systém, a tím bude umožněn jednodušší možný port na další platformy, například Linux.

1.2 Analýza základních rysů OS Windows a OS X

Oba operační systémy se vyvíjely nezávisle na sobě a každý vyvíjela jiná společnost s odlišnými potřebami. Díky tomu bylo učiněno mnoho rozdílných rozhodnutí, která odlišují systémy dodnes. Přesto si můžeme všimnout některých podobností, které tyto systémy sdílejí. Oba systémy dokáží pracovat s velkým množstvím periferií, mají pokročilé uživatelské rozhraní, dokáží otevřít velké množství různých typů souborů a mnoho dalšího.

Abychom mohli zanalyzovat a porovnat oba systémy, v krátkosti si připomeneme jejich historii. Jelikož Mac OS X je UNIX operační systém, základní porovnání provedeme mezi platformou Windows a UNIX. Následně blíže rozepíšeme filesystem a aplikační rozhraní, které nás bude více zajímat při přepisu aplikace FurryBall.

1.2.1 Historie

Zdroje informací pro tuto sekci byly [27, 6, 30, 31, 3]

1.2.1.1 Windows

V roce 1980 začal vývoj nové verze systému Windows nazvaný Windows NT. Microsoft začal s návrhem kvůli nově vyvíjeným procesorům, které přidávaly novější funkcionalitu, a se snažil využít těchto funkcí ve Windows NT. Všechny pozdější systémy od Windows Server 2003 jsou založené na verzi Windows NT.

Pro lepší orientaci vidíme na obrázku 1.1 historii operačního systému Windows od verze 3.1.

1.2.1.2 UNIX

Vývoj systému UNIX započal již kolem roku 1970 a jeho počáteční úspěch rozdělil systém do několika různých verzí. Nové verze vznikaly jako volně dostupné projekty, komerční produkty, univerzitní práce nebo vládní systémy.

Jelikož existuje více než 50 různých verzí UNIXu, uvádíme na obrázku 1.2 všechny důležité v přehledné tabulce.

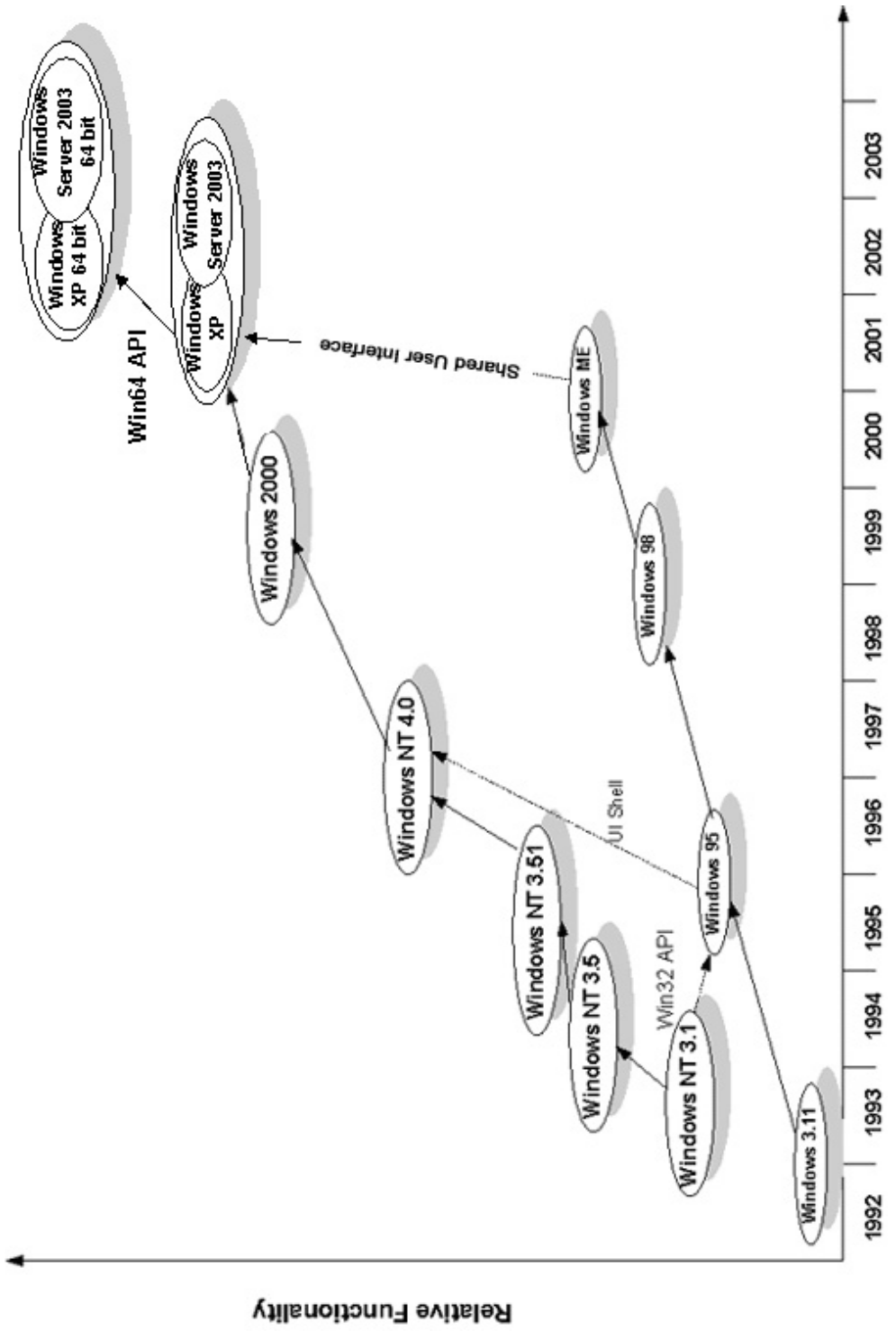
1.2.2 Filesystem

Oba operační systémy využívají jiný souborový systém.

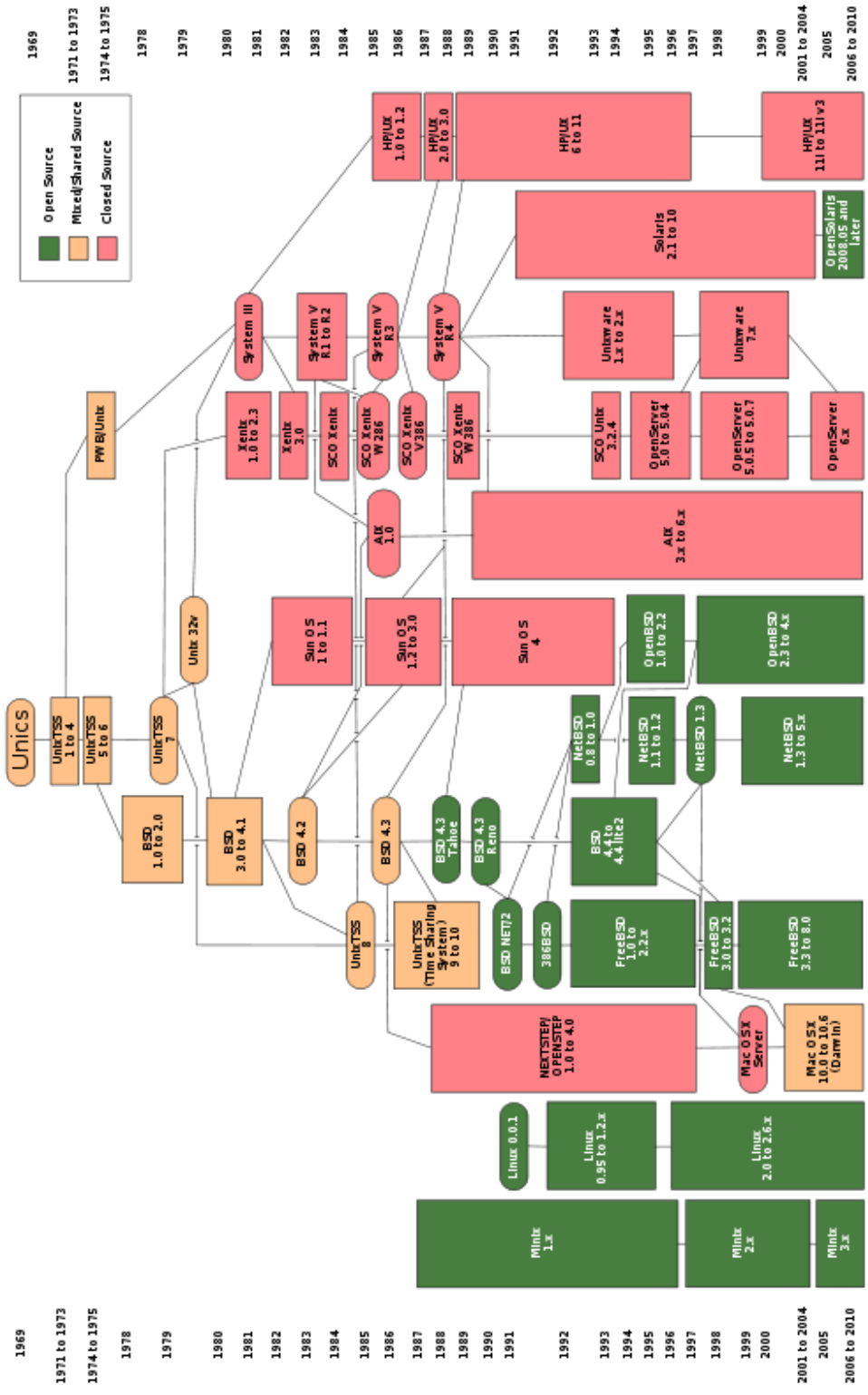
1.2.2.1 NTFS

NTFS je souborový systém vyvinutý společností Microsoft pro operační systémy od Windows NT. NTFS byl vytvořen na konci 80. let 20. století. Byl navržen tak, aby byl rozšiřitelný a schopný se přizpůsobit požadavkům novějších systémů.

Obrázek 1.1: Historie operačního systému Windows[28]



Obrázek 1.2: Historie UNIX operačních systémů[43]



Mezi některé vlastnosti NTFS patří žurnálování, komprese, šifrování a možnost vytvoření pevných i symbolických linků.

NTFS není bezchybný a obsahuje některé limity. Jedním z velkých problémů je délka relativní cesty k souboru na pouze 255 unicode znaků. Dále je velkým problémem fragmentace, díky které je čtení některých souborů zbytečně pomalé.

1.2.2.2 HFS+

HFS+ je hierarchický souborový systém vyvinutý společností Apple, který v roce 1998 nahradil původní HFS. Název HFS+ je převážně znám mezi vývojáři softwaru, pro běžné uživatele používá Apple označení Mac OS Extended.

Oproti předchozí verzi podporuje HFS+ mnohem větší soubory díky 32 bitové adrese namísto 16 bitové v HFS. Dále používá Unicode pro kódování řetězců názvů složek a adresářů a také využívá 32 bitovou alokační tabulku.

Ani HFS+ není bezchybný a obsahuje několik neduhů, které novější souborové systémy nemají. Mezi vlastnostmi, které chybí, jsou checksumy dat, fyzické pevné linky a problém, že HFS+ nebyl navržen pro moderní OS X. Z tohoto důvodu je třeba každá metadata před čtením převést z big-endianu na little-endian, který je používán v dnešní době.

1.2.3 Aplikační rozhraní

V této části si ukážeme rozdíly mezi aplikačním rozhraním UNIX systémů a Windows. Oba systémy, jak Windows, tak OS X, používají jádro hybridního typu, které využívá kladů z monolitického typu i mikrokernelu. Jelikož se obě platformy vyvíjely nezávisle na sobě, Windows před verzí Windows NT používal jádro typu mikrokernel. OS X používal až do verze 8.6 jádro monolitického typu stejně jako Linux, FreeBSD a Solaris.

Kvůli různým výhodám, které každý z těchto typů představuje, nakonec Windows i OS X přestoupily na hybridní jádro.

Monolitické jádro Monolitické jádro je typ jádra, který celý běží v jednom paměťovém prostoru, většinou označovaném jako kernel space. Monolitická jádra jsou strukturovaná tak, že jsou jednotlivé funkce oddělené, ale přesto velice provázané. Ve starších verzích nebyla možnost zavádění dynamickým modulů do jádra. Například pro připojení USB disku bylo třeba restartovat systém. V dnešní době existuje i dynamické načítání modulů. Jádro je ale stále monolitického typu, protože všechny moduly existují v jednom paměťovém prostoru. Rozhraní pro propojení operačního systému a jednotlivých procesů pak zajišťují systémová volání, díky kterým mohou používat služby operačního systému.

Mikrokernel Mikrokernel je typ jádra operačního systému, který má velice malou velikost. Obsahuje pouze základní funkce pro správu paměti, plá-

nování procesů a meziprocesorovou komunikaci. O všechny ostatní části se starají procesy, které běží v uživatelském prostoru. Tyto procesy se nazývají servery (někdy také známé jako démoni). Mezi uvedené servery patří správa souborového systému, ovladačů zařízení a periferií či podpora počítačových sítí. V jádru typu mikrokernel je velice důležitá meziprocesová komunikace, která nahrazuje systémová volání v jádru monolitického typu.

Hybridní jádro Hybridní jádro kombinuje obě výše popsaná dohromady z důvodu získání výhod z obou řešení. Hybridní jádra se používají ve většině komerčních operačních systémů jako Windows NT, Windows CE, Max OS X a DragonFly BSD.

1.2.3.1 Windows API (WinAPI)

Jelikož operační systém Windows se vyvinul z mikrokernelu, jednotlivé funkce jádra nejsou programátorům veřejně dostupná, jako je tomu u jádra monolitického typu. Místo toho se používá meziprocesová komunikace.

Všechny programy dostupné pro operační systém Windows musí, nezávisle na platformě, používat Windows API. Windows API obsahuje funkce základní, ale i funkce pro vytváření uživatelského rozhraní. Pro nízkoúrovňový přístup k Windows se používá Windows Driver Foundation nebo Native API.

Windows API můžeme rozdělit do několika kategorií, mezi které patří správa vstupu, paměti, souborů, Windows Environment (Shell) a uživatelské rozhraní.

1.2.3.2 Systémová volání

Systémové volání je mechanismus používaný k volání funkcí operačního systému. Používá se u systému s jádrem monolitického typu nebo z něj vzešlých. Systémová volání najdeme u všech unixových systémů.

Jelikož moderní procesory umožňují spouštět procesy v jejich vlastním adresovém prostoru, mohou požadovat data nebo služby z operačního systému. K tomuto se v Unixových systémech používají systémová volání, která odstiňují programátora od volání přímo systémové funkce a zabraňují tím rozbití běžícího systému.

Systémová volání jsou většinou zprostředkovaná knihovnou, která programátorovi skrývá některé detaily a zvyšuje tím přenositelnost kódu.

1.3 Analýza aplikace FurryBall

Aplikace FurryBall je velice rychlý a výkonný renderer napsaný v jazyce C++. FurryBall používá velké množství různých technologií a knihoven k docílení věrného výstupu, začínaje technologií CUDA[34] pro výpočty na grafické

kartě, přes framework OptiX pro jednodušší tvorbu Ray tracingových aplikací, až po knihovnu QT[40] obstarávající GUI v aplikaci. Všechny používané knihovny jsou posány v následující kapitole.

Renderování, neboli vykreslování, je technika generování (vytváření) obrazu z 2D nebo 3D scény. V běžném slova smyslu je scéna kolekce 3D modelů s nastavenými texturami, osvětlením, částicovými efekty a kamerou, snažící se přiblížit fiktivní či reálnou scenérii lidskému oku. Z programátorského hlediska je scéna datovou strukturou obsahující jednotlivé vertexy, odkazy na jejich textury, uložené osvětlení a kamery. Při vykreslování se tato datová struktura rozloží na menší části, které se nahrají na grafickou kartu. Na grafické kartě se vypočítá jejich pozice, osvětlení a barva a výsledky těchto výpočtů se zobrazí na obrazovce. Zároveň velkou výhodou používání grafických karet pro výpočty je výrazně snadnější rozšiřitelnost. Většinou nejsou kladena omezení na počet grafických karet.

1.3.1 Technologie CUDA a OptiX framework

FurryBall používá pro výpočty na grafické kartě dvě důležité technologie. Obě technologie vyvíjí společnost NVIDIA pro zjednodušení výpočtů na grafických kartách. Technologie CUDA dovoluje spouštět programy napsané v jazycích C, C++ a Fortran, popřípadě programy napsané v technologiích OpenCL[21] a DirectCompute[33] v paralelním běhu na grafických jádrech. Typický program napsaný v technologii CUDA se skládá ze dvou částí. První je hostující program běžící na procesoru a rozdělující práci grafické kartě. Druhou je samotný program běžící na grafické kartě. Výhodou této technologie je využití jader grafické karty, která poskytuje mnohonásobně větší výkon než procesor.

Framework OptiX[25] usnadňuje programování aplikací, které potřebují používat vysílání paprsků do scény, ať se již jedná o grafické aplikace, simulování radiace, odrážení akustických vln či analýzu kolizí. OptiX odděluje programátora od většiny náročného programování spojeného s raytracingem a dovoluje mu soustředit se na řešení problému.

Abychom si dokázali představit, kolik výpočtů je v dnešní době třeba provést pro běžný animovaný film, musíme si vysvětlit techniku, kterou se tyto filmy renderují. Této technice se říká Path tracing, což je pouze specializovaná odnož Ray tracingu.

1.3.1.1 Ray tracing a Path tracing

Zdroje informací pro tuto sekci byly [10, 11]

Obě technologie se snaží co nejvíce napodobit přenos světla, který existuje v reálném světě. V reálném světě zdroj světla vyšle paprsky do všech směrů, které se následně odrážejí od solidních objektů a částečně procházejí poloprůhlednými objekty. Pokud bychom chtěli toto praktikovat i v počítači, plýtvali bychom zbytečně mnoha zdroji, protože většina paprsků by vůbec netrefila

kameru a odrazila by se do nekonečna na druhou stranu. Zároveň i v reálném světě vidíme pouze paprsky, které trefí naše oko.

Tím pádem Ray tracing i Path tracing fungují přesně obráceně, než by se očekávalo. Paprsky se vysílají pouze přes pixely, které budou tvořit výsledný obrázek, a tím šetří vzácný výpočetní výkon. Kvalita výsledného obrázku je poté závislá na SPP, neboli vzorcích na pixel. Čím více vzorků vypočítáme, tím kvalitnější a přesnější bude výsledný obraz.

Hlavním rozdílem mezi Ray tracingem a Path tracingem je ve výpočtu, který se provádí. Ray tracing je schopný vypočítat pouze přímé osvětlení a pro ostatní druhy světel, jako nepřímé osvětlení a globální osvětlení, používá jiné technologie. U Path tracingu si můžeme paprsek představit jako malý míč, který se vystřelí do scény a odráží se od jednotlivých objektů. Pro zrychlení výpočtů se posílá několik paprsků najednou, kdy každý se odrazí pod jiným úhlem a každý počítá jinou část, světlo, zastínění, nepřímé osvětlení nebo globání světlo.

Obě metody používají vysílání paprsků do scény, ale každá počítá výsledek jiným způsobem. Path tracing je brute-force metoda s větším počtem výpočtů, ale mnohem kvalitnějším výstupem, kdežto Ray tracing je rychlejší za cenu menší obrazové věrnosti.

1.3.1.2 Příklad

Pro lepší představu se můžeme podívat na níže uvedené obrázky 1.3 a 1.4. Obrázky byly vyrenderované v rozlišení 960 pixelů na šířku a 480 pixelů na výšku Path tracingovým rendererem. Výsledný obrázek tudíž obsahuje 460 800 pixelů. Při Path tracingu se neposílá jediný paprsek pro každý pixel, ale vyšle se několik primárních paprsů, které se pomocí náhodné funkce později rozdělí, kam budou pokračovat. V tomto případě bylo použito 400 primárních paprsků. Ve scéně se počítalo primární osvětlení, globální světlo i nepřímé osvětlení. Zároveň si můžeme všimnout reflekcí a částečné průhlednosti.

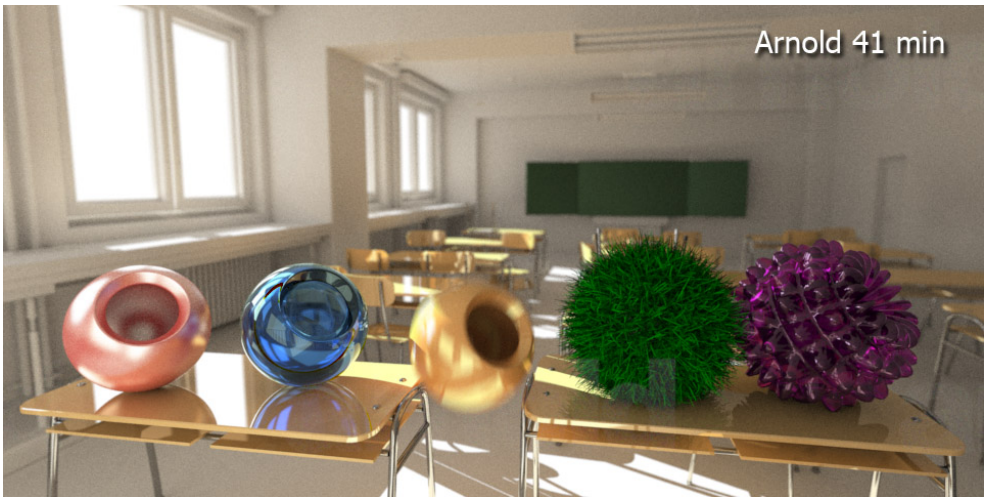
Na obou ilustracích je dobře vidět rozdíl v době renderování mezi rendererem Arnold a FurryBall RT rendererem. Oba obrázky byly renderované na stejné sestavě.

1.3.2 Části aplikace FurryBall

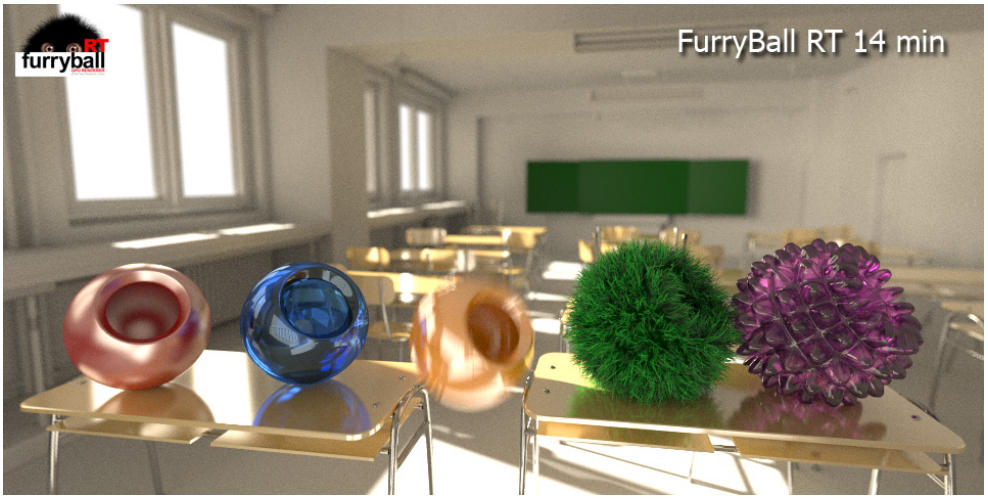
Aplikace FurryBall se skládá ze dvou dílčích aplikací FurryBall App a FurryBall Standalone a ze čtyř knihoven FurryBall Common, FurryBall Render, FurryBall Licensing a FurryBall Utils. V této části jsou blíže představeny a je zde vysvětleno, jakou úlohu hrají v celku.

FurryBall App je GUI aplikace, která spouští aplikaci FurryBall Standalone. Zobrazuje stav licence, průběh renderování a umožňuje pustit testovací render bez potřeby dalších aplikací.

Obrázek 1.3: Scéna renderovaná CPU rendererem Arnold.[8]



Obrázek 1.4: Scéna renderovaná GPU rendererem FurryBall RT.[9]



FurryBall Standalone je konzolová aplikace, která se stará o vlastní průběh renderování a posílá stavové informace do FurryBall App.

FurryBall Common je dynamická knihovna, která obsahuje většinu společných funkcí pro ostatní knihovny.Obsahuje definici a implementaci metod pro práci s 3D objekty, světly a texturami.

FurryBall Render je dynamická knihovna, která se stará o renderování. Obsahuje třídy, které řeší obsluhu renderování objektů, materiálů a světel.

FurryBall Licensing je soubor hlavičkových souborů, které se vkládají do ostatních částí aplikace a jeden .cpp soubor, který se kompiluje zvlášť s každou částí aplikace FurryBall.

FurryBall Utils je statická knihovna, kterou využívají všechny ostatní části aplikace. Tato knihovna obsahuje všechny deklarace a definice datových typů, třídy pro pracování s řetězci znaků, třídu obstarávající zápisy do registrů, serializaci a řadu dalších.

1.4 Analýza požadavků prototypu

1.4.1 Funkční požadavky

Konzolové rozhraní - Prototyp bude obsahovat funkční konzolové rozhraní a bude reagovat na všechny příkazy, které jsou dostupné v plné verzi aplikace.

Renderování snímku - Vyrenderování snímku je základní funkcí FurryBallu a z tohoto má tuto funkci i prototyp.

Nastavitelnost - Vykreslování lze nastavit pomocí souboru, kde jsou obsažena nastavení, která potlačí základní.

Logování - O průběhu renderování bude aplikace informovat uživatele pomocí konzolových logů.

Ukládání - Prototyp bude schopen uložit všechny požadované obrázky v různých grafických formátech.

Benchmark - Prototyp bude schopný pustit benchmark a odeslat jeho výsledky zpět na server se statistikama o výkonu grafické karty.

1.4.2 Nefunkční požadavky

Platforma - Prototyp bude spustitelný na operačním systému OS X ve verzi 10.10 nebo vyšší.

Stabilita - Aplikace bude stabilní a nebude neočekávaně padat při renderování snímku.

Paměť - Prototyp nebude potřebovat více grafické paměti než je potřeba.

Rozšiřitelnost - Prototyp bude schopný používat všechny dostupné prostředky, využívat všechny jádra procesoru a všechny dostupné grafické karty pro výpočty.

Kapitola 2

Návrh

Při přepisu velké aplikace, jako je FurryBall, máme dvě možnosti, ze kterých můžeme vybírat. První, a v některých případech jednodušší, je použít Windows emulátor, který obstará většinu práce za Vás. Pro přepis aplikace FurryBall byl jako jedna z možností vybrán emulátor WINE[44]. Druhou možností, pracnější, je vlastnoruční přepis celé aplikace, odstranění nativních závislostí. Výhodou této možnosti je větší kontrola nad výslednou aplikací, protože programátor přesně ví, která část dělá jaké úkony. Nevýhodou je delší čas implementace a následná údržba, kdy se při každé nové funkcionalitě musí kontrolovat zpětná kompatibilita s druhým systémem.

Po analýze celé aplikace byla vyřazena první možnost z důvodu nedostatečné rychlosti. Jelikož WINE je emulátor, můžeme si představit, že pustí pod pokličkou instanci Windows, ve které pak pustí požadovanou aplikaci. FurryBall si velice zakládá na rychlosti výpočtů a jelikož používá nativní drivery pro práci s grafickou kartou, použití emulátoru by výrazně zpomalilo výsledné výpočty.

Pro vlastnoruční přepis aplikace z Windows na OS X neexistuje žádný podrobný návod, ale existují jisté poučky, kterými se můžeme řídit.

Aplikace psané v čistém C/C++ by neměly mít problém s kompilací a mělo by stačit dávat pozor na soubory, systém souborů a GUI. Bohužel to tak zcela neplatí, hrají zde roli další překážky jako různá implementace standardních knihoven, nestandardní funkce v hlavičkových souborech, závislost na nativních knihovnách a samozřejmě rozdíly v kompilátorech používaných na Windows a OS X.

2.1 Proces přepisu

Po analýze aplikace FurryBall byly navrženy dva procesy pro port, podle kterých se budeme v této práci při tvorbě prototypu řídit. Proces na diagramu 2.1 znázorňuje kompilaci projektu a proces na diagramu 2.2 jeho linkování s knihovnami.

Přepis každého projektu je přímočará práce zahrnující opakovanou kompilaci a opravování chyb, které při ní nastanou. V následujícím diagramu jsou ve zkratce znázorněny ty situace, které mohou nastat nejčastěji. Jsou zde podrobněji popsány, včetně návrhu, jak je možné se s nimi vypořádat.

Kompilace kódu je v tomto případě velice jednoduchá. CMake, který používáme jako build systém, řeší veškeré problémy za nás. Vytvoří nativní makefile nebo projekt a spustí kompilaci toho projektu.

Pokud chybí knihovny, musíme zjistit, zda-li máme dostupnou požadovanou knihovnu. Pokud neexistuje v systému, musíme ji nainstalovat. Dále musíme zjistit, zda existuje soubor Find#NazevKnihovny.cmake, který zajistí nalezení požadované knihovny a jejích hlavičkových souborů a zároveň poskytne proměnné, díky kterým je můžeme přidat do CMakeLists.txt souboru.

Pokud nalezneme hlavičkový soubor windows.h nebo jiné windows hlavičkové soubory jako stdafx.h, musíme je obalit do preprocesorového makra #ifdef _WIN32, abychom zajistili zpětnou kompatibilitu pro Windows a zároveň dokázali kód zkompilovat na dalších platformách. S tím je spojeno i stejné obalení funkcí, které byly používané z příslušných hlavičkových souborů nebo napsání funkcí s ekvivalentní signaturou, ale spustitelné na OS X.

Takto projdeme a upravíme soubory v celém projektu, dokud se nám ho nepodaří zkompilovat.

Kompilací naše práce nekončí. Po kompilaci následuje linkování programu se všemi knihovnami a i v této části mohou nastat komplikace. Na diagramu 2.2 vidíme zjednodušený postup pro opravování chyb v linkování.

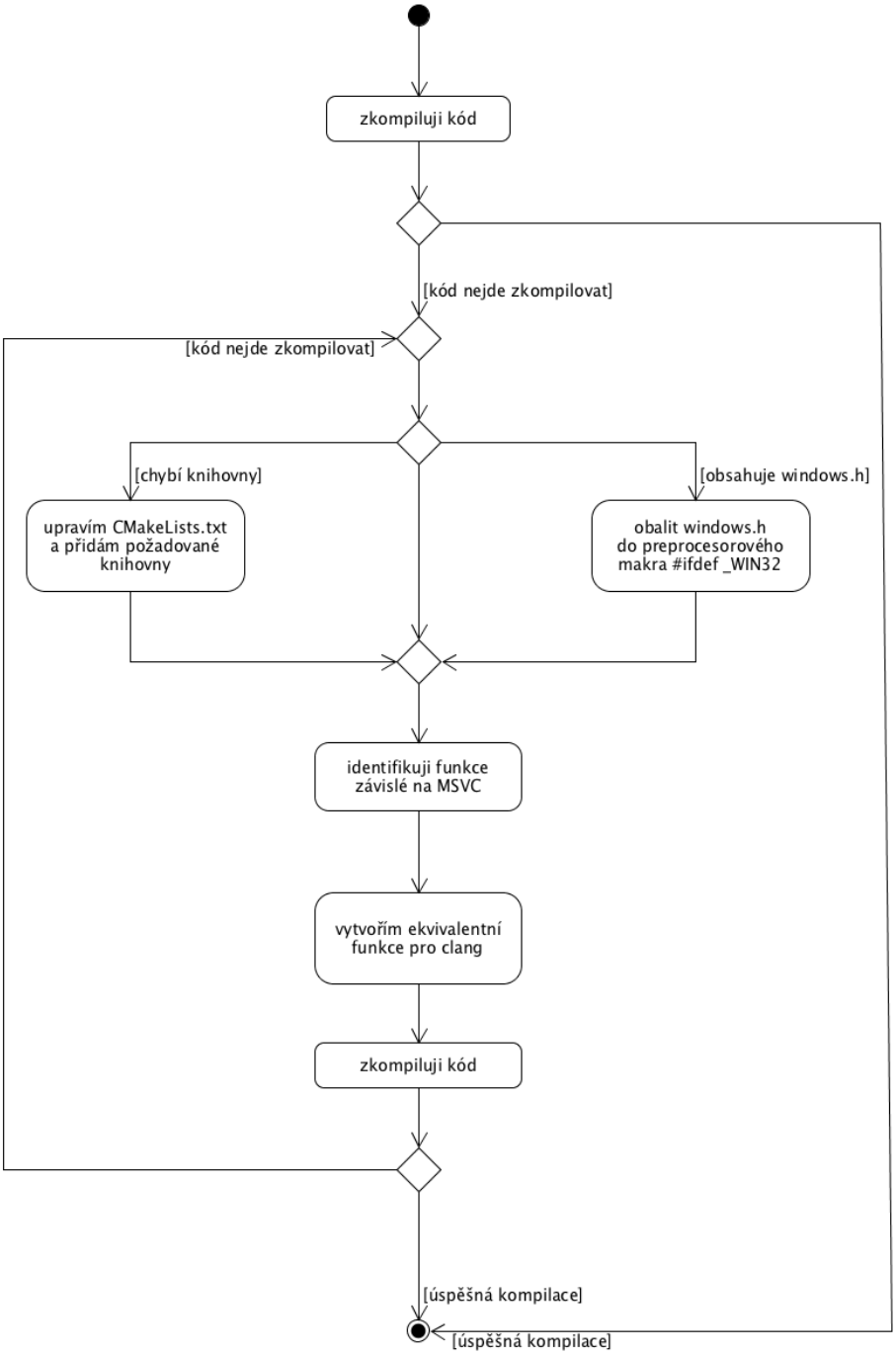
Při linkování programu mohou nastat dva problémy, buď knihovna zcela chybí, nebo máme špatnou verzi knihovny se špatnou standardní knihovnou.

Pokud nám linker ukazuje, chybějící knihovnu, kterou jsme již přidávali do CMakeLists.txt, pak je nainstalovaná na špatném místě. Poslední možností je špatně napsaný soubor Find#NazevKnihovny.cmake. Pokud pročteme soubor Find#NazevKnihovny.cmake, zjistíme, ve kterých složkách CMake hledá knihovnu, a můžeme změnit její umístění nebo přepsat tento soubor, aby hledal knihovnu na správném místě.

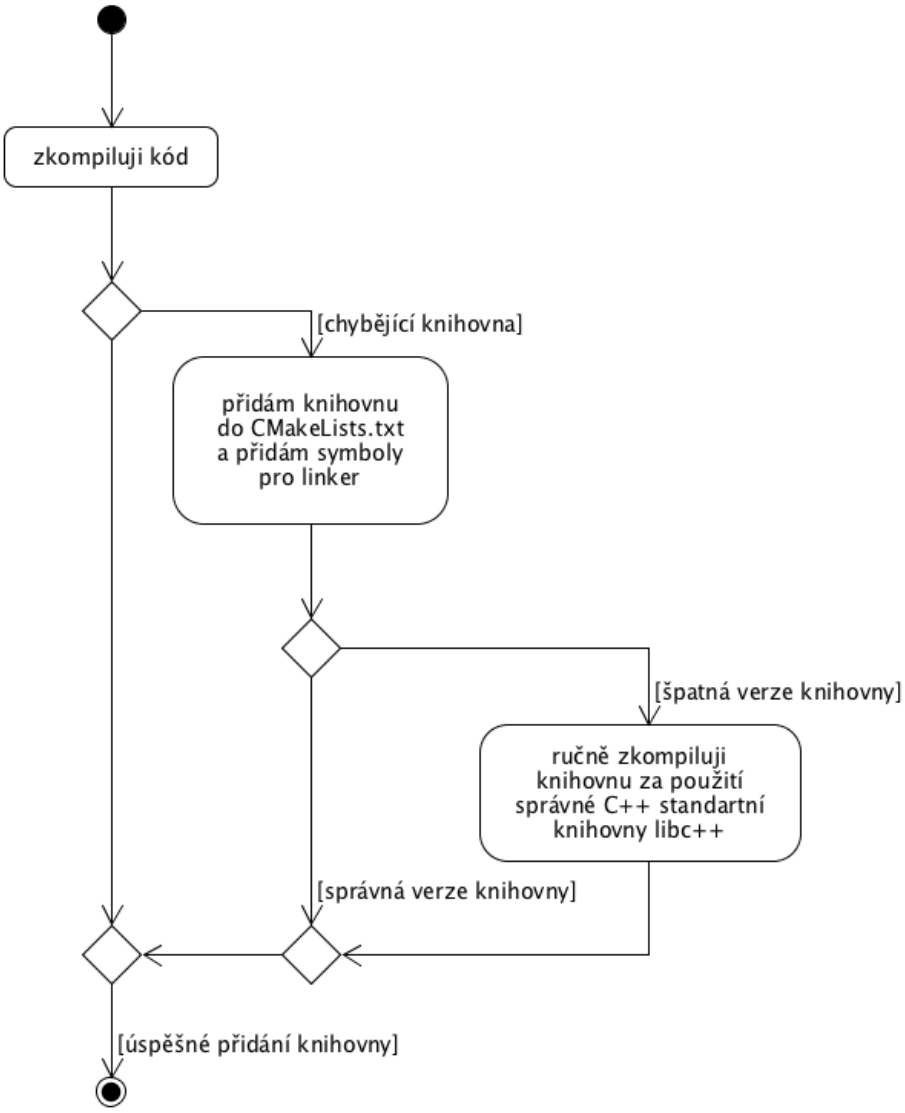
V některých případech se stane, že se knihovna zkompiluje při instalaci se špatnou standardní knihovnou. Na operačním systému OS X jsou dostupné dvě knihovny. První z nich, používaná například kompilátorem gcc, je knihovna, která se linkuje přepínačem -libstdc++. Tuto knihovnu používá většina hlavních UNIX distribucí. LLVM má implementovanou vlastní C++ standardní knihovnu dostupnou pod přepínačem -libc++. Problém s používáním těchto implementací zároveň je, že ačkoliv obě mají totožné API, což znamená, že std::string v implementaci libc++ a std::string v implementaci libstdc++ bude mít stejnou signaturu, jejich ABI bude rozdílné. Linker tento rozdíl pozná a zobrazí chybu, kde je použití namespace std::__1 nápovědou pro odstranění chyby.

Přepis jednotlivých částí budeme brát v pořadí důležitosti a používanosti v aplikaci. V první řadě je třeba začít s knihovnou FurryBall Utils, která je

Obrázek 2.1: Diagram přepisu aplikace



Obrázek 2.2: Diagram linkování aplikace



využívaná ve všech částech výsledné aplikace. Další částí, kterou bude třeba přepsat, je dynamická knihovna FurryBall Common a následné vytvoření mockupu projektu FurryBall Licensing, který mimikoval funkcionalitu licensování pro potřeby prototypu. Předposledním projektem pak je samotná renderovací knihovna FurryBall Render a posledním konzolová aplikace FurryBall Standalone.

2.2 Jednotné vývojářské místo

Práce na aplikaci FurryBall má několik specifických požadavků, od specifického hardwaru, po rozličný software a knihovny. Práce na prototypu bude vyžadovat počítač s nainstalovaným operačním systémem OS X ve verzi 10.10 a vyšší. Jelikož je FurryBall renderer využívající výhod programovatelných grafických karet technologií CUDA od společnosti NVIDIA, zároveň je zapotřebí mít grafickou kartu od společnosti NVIDIA. Mezi programy, které jsou potřeba, je vybrané IDE, v našem případě Clion, SVN klient pro správu verzí, CMake, ovladače CUDA pro grafickou kartu a Homebrew[26] pro jednodušší instalaci závislostí. FurryBall používá velké množství knihoven, mezi které patří LibJpeg[41], LibPng[18], OpenEXR[19], OpenSubdiv[36], CUDA, OptiX, Boost[1], GUI knihovna QT a testovací knihovna UnitTest++[4]. V této kapitole je popsán návod pro instalaci jednotlivých programů a knihoven na OS X.

Instalace všech potřebných programů je přímočará. Všechny programy, kromě IDE, jsou zdarma ke stažení, v plné nebo alespoň omezené verzi. Pro naše vybrané IDE Clion existuje komerční licence nebo studentská, poskytnutá FIT ČVUT.

Homebrew Homebrew je náhrada za chybějící balíčkový manager pro OS X. Obsahuje velké množství používaných knihoven a zjednodušuje práci s jejich instalací. Stažená knihovna se zkompiluje a zkopíruje do složky /usr/local/Cellar/, soubory knihovny se nalinkují do /usr/local/lib/ a hlavičkové soubory do /usr/local/include/.

SVN klient Jako SVN klient byl vybrán Smart SVN[39] z důvodu použití stejného programu i na ostatních vývojářských stanicích. Smart SVN poskytuje omezenou licenci, ale s veškerou potřebnou funkcionalitou.

CMake CMake je build systém, který dokáže vytvořit nativní projekty pro velké množství editorů od Microsoft Visual Studio, přes XCode, až po nativní GNU makefile. Dovolí nám jednoduše pokračovat v dalším možném použití, například na operačních systémech Linux.

Nainstalování knihoven je náročnější záležitost. Některé z nich již mají připravené balíčky v manageru Homebrew, ale některé z nich musíme zkompilovat ručně. U všech knihoven se předpokládá, že budou nainstalované ve

složce /usr/local/lib/ a hlavičkové soubory budou dostupné přes /usr/local/include/.

QT Knihovna QT byla jednou z náročnějších na instalaci, převážně z důvodu nepřehledné struktury knihoven a hlavičkových souborů. Knihovna je třeba ve verzi 5 a je třeba pouze knihovna bez editoru. Knihovnu lze najít na stránkách `https://www.qt.io`.

CUDA Knihovna CUDA pro práci s grafickou kartou je od společnosti NVIDIA a má kvalitně zhotovený instalátor. Existující FindCUDA.cmake práci ještě více ulehčí. Knihovnu CUDA je potřeba mít nainstalovanou ve verzi 7.5. Knihovnu CUDA lze stáhnout na stránkách `https://developer.nvidia.com/cuda-downloads` bez nutnosti registrace.

OptiX Knihovna OptiX, taktéž od společnosti NVIDIA, má kvalitní instalátor, bohužel bez existence FindOptiX.cmake souboru, ale ten je jednoduše k nalezení. Instalátor můžeme stáhnout na stránkách `https://developer.nvidia.com/optix`, bohužel s nutností registrace jako NVIDIA vývojář.

OpenSubdiv OpenSubdiv je volně dostupná knihovna od společnosti Pixar. Tato knihovna poskytuje všechny zdrojové soubory, které je třeba zkompilovat na každé platformě zvlášť. Naštěstí Pixar připravil velice podrobný návod pro kompilaci. Knihovna je k dispozici na stránkách `http://graphics.pixar.com/opensubdiv/docs/intro.html`

OpenEXR OpenEXR je dostupná knihovna pro práci s obrazovým formátem exr vyvíjená společností Industrial Light & Magic. Knihovna je dostupná na stránkách `http://www.openexr.com` a zároveň ji můžeme nainstalovat příkazem brew install openexr.

LibJpeg LibJpeg je knihovna pro práci s obrazovým formátem Jpeg dostupná přes balíčkový manager Homebrew. Lze ji nainstalovat pomocí příkazu brew install libjpeg.

LibPng Knihovna LibPng je knihovna pro práci s obrazovým formátem png. Knihovna je dostupná přes balíčkový manager Homebrew pomocí příkazu brew install libpng.

Boost Boost je známá knihovna fungující na většině operačních systémů poskytujících velké množství funkcí a dobře fungujících se standardní C++ knihovnou. Boost je na velkém množství systémů předinstalován, ale pokud chybí, lze ho nainstalovat přes Homebrew pomocí příkazu brew install boost. Tato knihovna by měla být zkompilovaná se standardní knihovnou -libc++, ale v některých případech může být linkovaná s knihovnou -libstdc++, a v tom případě je třeba ji opět zkompilovat se správnou standardní knihovnou.

UnitTest++ UnitTest++ je multiplatformní knihovna pro psaní unit testů. Používá CMake jako build systém a díky tomu je jeho instalace velice přímočará. Repozitář stáhneme z těchto stránek `https://github.com/unittest-cpp/unittest-cpp` a postupujeme dle přiloženého návodu.

Kapitola 3

Implementace prototypu

Po rozdělení projektů, zjištění návaznosti jednotlivých projektů na sebe a návrhu přepisu bylo třeba jednotlivé části přepsat. Jejich současná implementace s sebou přinesla spoustu zajímavých překážek, které bylo třeba vyřešit. V této kapitole jsou popsány největší problémy, které trvaly nejvíce času.

3.1 Problémy s prototypem

3.1.1 Vnitřní reprezentace znaků na Windows a OS X

První problém, který se dal očekávat, je rozdílná reprezentace řetězců znaků na Windows a OS X. Problém, jak se dá vytušit, spočívá v kódování, které používá Windows, OS X a ostatní Unixové systémy.

3.1.1.1 Vysvětlení problému

Jelikož je toto téma obsáhlé a bez pochopení některých základních pojmů je jeho vysvětlení nemožné, nadefinujeme si zde ty nejdůležitější.

Kódování znaků Kódování znaků je proces namapování jednotlivých znaků na sekvence bitů.

Znaková sada Znaková sada je množina všech znaků, které lze zakódovat do sekvence bitů.

Tabulka znaků Tabulka všech znaků, která mapuje znaky na čísla nebo sekvence bitů.

ASCII Kódování ASCII je základní kódování, ve kterém najdeme původ všech ostatních kódování. V kódování ASCII lze zakódovat až 256 znaků, což zahrnuje všechy znaky od A-Z, a-z, všechny cifry 0-9 a několik kontrolních znaků. Když se zamyslíme nad počtem různých znaků všech

jazyků, 256 je poměrně malé číslo a rozhodně zde nedokážeme uložit všechny potřebné znaky.

Unicode Unicode[42] je technická norma, která definuje rozsáhlou tabulku obsahující 1 114 112 tzv. codepointů, neboli kódových značek, které definují jednotlivé znaky. Tato tabulka zahrnuje všechny znaky, které jsou v dnešní době známé. Důležité je pochopit, že Unicode není kódování, ale pouze norma, která definuje výše uvedenou tabulku. Existuje několik kódování, které dokáží zakódovat Unicode kódové značky do bitů. Pro fixní šířku existuje UTF-32, pro proměnnou šířku na znak pak UTF-8 a UTF-16.

UCS-2 UCS-2 je starší 16 bitová implementace Unicode standardu ve verzi 1.1, v dnešní době nepoužívaná. Nedokáže zakódovat znaky větší délky než 16 bitů.

UTF-8 UTF-8 je kódování s proměnnou délkou znaku. Díky proměnné délce dokáže zakódovat všechny znaky z Unicode standardu. Pokud lze znak zakódovat v jednom bytu, UTF-8 použije jeden byte. Pokud je třeba použít více bytů, UTF-8 jich použijí více a pro rozpoznání, kolik bytů obsahuje jeden znak, používá nejvyšší bity. UTF-8 dokáže ušetřit místo v případě, že řetězec znaků neobsahuje velké množství dvou a více bytových znaků.

UTF-16 UTF-16 na druhou stranu používá 16 bitů na jeden znak, je tím pádem úspornější než předchozí UTF-8 a taktéž dokáže zakódovat všechny znaky z Unicode standardu. UTF-16 je novější implementace staršího kódování UCS-2, které také používalo 16 bitů na jeden znak.

V tuto chvíli, kdy máme definované všechny potřebné zkratky, si můžeme přiblížit problém, který nastal při implementaci třídy String v aplikaci FurryBall. Aplikace FurryBall používá formát JSON[2] pro serializaci hodnot a některé konfigurační soubory. Ve formátu JSON jsou všechny informace uložené do řetězce znaků. Pokud bychom otevírali stejný soubor na jednom systému, nevznikne žádný problém, ale při práci na více různých operačních systémech můžeme narazit na rozdíly v kódování znaků.

Konkrétně na platformě Windows se pro všechny API volání do systému používá kódování UTF-16 nebo UTF-8 a na platformě OS X pouze UTF-8. Druhým problémem byla rozdílná velikost datového typu wchar_t.

Datový typ wchar_t je typ, který svým názvem signalizuje, že se jedná o větší datový typ než normální char veliký 1B, v datovém typu wchar_t stojí pro wide, tudíž má větší rozsah než char. Operační systém Windows používá wchar_t pro systémové API s kódováním UTF-16 a má k tomu přízpůsobenou standardní knihovnu. Standardní knihovna v Microsoft Visual Studiu obsahuje několik přidaných konstruktorů pro třídy wifstream, wofstream a jejich stringové ekvivalenty. Tyto konstruktory ve standardní knihovně, kterou používá

Clang, neexistují, a proto bylo nutné vymyslet řešení, které by uspokojilo obě platformy.

3.1.1.2 Řešení

Po delším bádání bylo nalezeno řešení, které by dokázalo oba problémy odstranit. Zároveň se pravděpodobně jednalo o nejjednodušší způsob. Aplikace bude vnitřně pracovat v jakémkoliv kódování je spuštěná a bude kontrolovat pouze všechny řetězce, které bude otevírat a ukládat. Všechny tyto soubory musí být uloženy v kódování UTF-8. Díky tomu zaručíme správné otevření a uložení na obou platformách. Tento převod byl zajištěn několika málo kroky.

```
std::string loc(setlocale(LC_ALL, NULL));
setlocale(LC_ALL, "");
mbstowcs_s(&size, NULL, s, 0);
mbstowcs_s(&size, mbstr, size + 1, s, size);
mbstr[size] = 0;
```

- Nastavením správného locale voláním funkce setlocale[14].
- Nastavením codecvt při čtení a zapisování do souborů.
- Používáním datového kontejneru wstring v celé aplikaci, pouze při API volání a otevírání souborů převést wstring na běžněji používaný string funkcemi mbstowcs[13] a wcstombs[15].

Následně při čtení a zapisování souborů pak používat:

```
finText.imbue(
        std::locale(std::locale::clasic(),
        new std::cdecvt_utf8<wchar_t>())
);

foutText.imbue(
        std::locale(std::locale::classic(),
        new std::codecvt_utf8<wchar_t>())
);
```

3.1.2 Rozdíly mezi MSVC a Clangem

Další problémy, se kterými bylo nutné se potýkat, byly ve výsledku pouze rozdíly mezi kompilátorem použitým ve Visual Studiu a použitým Clangem. Těchto problémů bylo několik, ale žádný z nich nebyl tak závažný jako předešlá práce s řetězci znaků.

3.1.2.1 Různá hloubka zanoření pří řešení přetěžování operátorů v šablonách

Tento problém se objevil při kompilaci a projevuje se chybovou hláškou:

```
ambiguous call to overloaded function
```

Vyskytuje se při vytvoření operátoru přetypování z C++ šablony. Problémem je, že kompilátor je schopný najít dvě různé cesty, kterými jde přetížit šablona a není schopný rozhodnout, které přetížení použít. Bohužel současná implementace používá Visual Studio 2012, které má slabší kontrolu přetížení operátorů a z tohoto důvodu dokáže konverzi provést, kdežto Clang má kontrolu přísnější a díky tomu kompilace skončí s touto chybovou hláškou.

Tento problém nebylo možné uspokojivě vyřešit bez většího zásahu do zdrojového kódu. Z toho důvodu byl zvolen postup reimplementace problémových míst. Bohužel to znamenalo menší přehlednost v kódu z důvodu potřebné explicitní implementace některých konverzních metod.

3.1.2.2 Bezpečné verze základních funkcí

Microsoft se snaží napomoci vývojářům poskytováním bezpečnějších verzí obvyklých funkcí, jako je memcpy a memset. Vytvořili pak funkce memcpy_s a memset_s, které testují, zdali je zdroj stejně velký jako cíl. Z důvodu použití těchto funkcí bylo třeba vytvořit kopie těchto funkcí v podmíněném překladu. Výsledná funkce zavolala pouze obvyklou funkci s méně parametry. Z funkce memcpy_s se pak ve výsledku stala pouze memcpy.

Těchto funkcí se v kódu vyskytovalo několik, ale jejich přepis byl vcelku přímočarý. Z tohoto důvodu byl vytvořen hlavičkový soubor, který implementuje tyto problémové funkce a při nalezení další se přidá do hlavičkového souboru její implementace.

3.1.2.3 Primitivní funkce z Windows API

Aplikace FurryBall obsahovala i některé funkce z Windows API. Mezi tyto funkce patřila například funkce Sleep nebo ZeroMemory. Tyto funkce bylo zapotřebí implementovat, ale jejich implementace byla velice přímočará. Jediným problémem zde byly rozdílné jednotky u funkce Sleep.

3.2 Struktura balíčku prototypu

Zdrojem informací k této sekci byl [5]

V současné době je prototyp dostupný pouze na vývojářském stroji v podobě spustitelného binárního souboru a nalinkovaných knihoven.

Pokud bychom chtěli mít prototyp jako funkční aplikaci, bylo by třeba vytvořit aplikační strukturu. Struktura aplikace FurryBall by pak vypadala takto:

```
FurryBall.app
└── Contents
    ├── Info.plist
    ├── MacOS
    │   └── FurryBallExecutable
    └── Resources
        └── FurryBallIcon.icns
```

Soubor Info.plist je soubor strukturou podobný XML, ve kterém se nachází důležité informace pro spuštění aplikace systémem. Mezi základní a požadované informace se řadí:

CFBundleName Pro název aplikace.

CFBundleIcon Pro ikonu zobrazenou v prohlížeči souborů.

CFBundleExecutable Pro cestu k vstupnímu bodu do aplikace, v případě prototypu spustitelný binární soubor konzolové aplikace FurryBall Standalone.

Po vytvoření požadovaného souboru Info.plist potřebujeme ještě zařídit obsažení veškerých potřebných knihoven a hlavičkových souborů pro vytvoření výsledného balíčku aplikace.

3.3 Instalace prototypu

Prototyp aplikace FurryBall není veřejně dostupný a je dostupný pouze pro vnitřní potřeby dalšího vývoje. Jeho instalace je z tohoto důvodu obtížnější, než by se dalo očekávat.

Pro spuštění prototypu na jiném než vývojářském stroji je zapotřebí mít dostupné zdrojové kódy pro přeložení a zkompilování. Všechny knihovny musí být dostupné na potřebných místech, jak je popsáno v kapitole 2.2, a je povinností mít nainstalovaný CMake a kompilátor Clang.

Pro spuštění konzolové aplikace je zapotřebí existence emulátoru terminálu, všechny dostupné příkazy jsou dostupné z webové adresy: `http://aaa-studio.cz/furryballRThelp/` v záložce Command line.

3.4 Současný stav prototypu

Z celé aplikace FurryBall je v současné době funkční pouze konzolová aplikace, která dokáže podle zadaných příkazů provádět jednoduché i více pokročilé operace. Konkrétně jsou pak z aplikace FurryBall přepsané knihovna Utils, Common a Render a konzolová aplikace Standalone, kterou budeme spouštět jako prototyp.

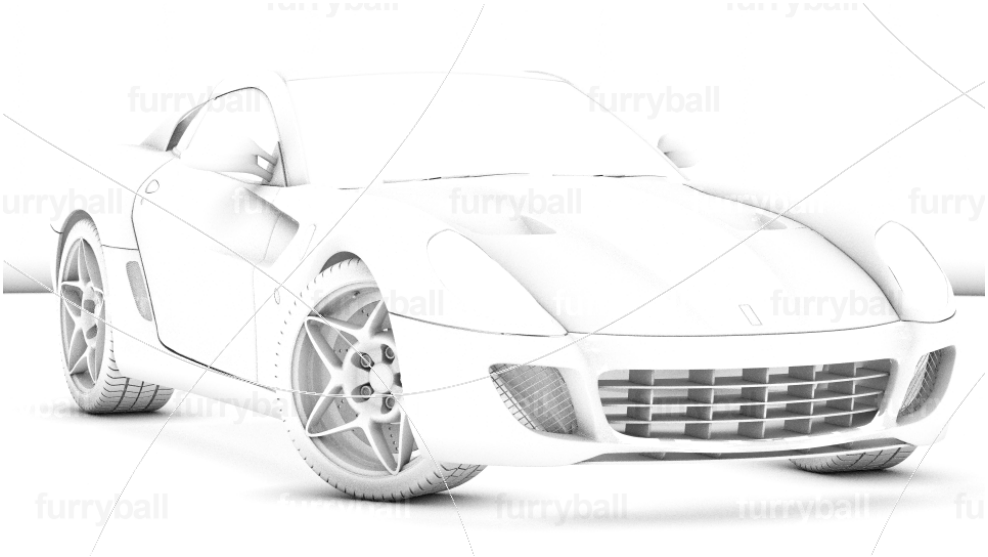
Přes konzolovou aplikaci pak lze vyrenderovat zadaná scéna s vlastním nastavením ve zvlášním souboru s příponou .fbo.

Pro představu jsou zde uvedeny příkazy, které dokáže konzolová aplikace zpracovat:

- **render** Příkaz pro vyrenderování scény v zadané složce.
- **info** Příkaz pro získání informací o FurryBallu, jako je počet aktivních karet, verze aplikace.
- **settings** Příkaz pro získání nebo nastavení aplikace FurryBall.

Všechny uvedené příkazy jsou podporované i v prototypu aplikace. Z hlediska vykreslování dokáže prototyp aplikace vykreslit menší scénu s veškerým nastavením i texturama. Na ilustraci 3.1 můžeme vidět vykreslenou scénu se zapnutým, pouze ambientním, osvětlením. Na dalším obrázku 3.2 vidíme zapnuté textury a přímé osvětlení. Na poslední ilustraci 3.3 je zapnuté větší množství efektů včetně nepřímého osvětlení, textur a různých post-process efektů jako simulace hloubky obrazu a bloom.

Obrázek 3.1: Benchmark scéna se zapnutým ambient occlusion



Obrázek 3.2: Benchmark scéna s přímým osvětlením a texturama

Obrázek 3.3: Benchmark scéna s veškerým nastavením a texturama

Kapitola 4

Testování

Zachování veškeré funkcionality při takto velké aplikaci je náročná věc. Pro jednoho programátora je prakticky nemožné si při přidání nových funkcí uvědomit všechny následky, které to může způsobit. Z tohoto důvodu existují různé druhy testů, které mohou případnou chybu jednodušeji odhalit. V současné implementaci aplikace FurryBall jsou dostupné pouze jednotkové testy a to navíc pro velice malé množství kódu. Z tohoto důvodu bylo zapotřebí přidat další jednotkové testy pro novou funkcionalitu a navrhnout integrační, regresní a uživatelské testy.

4.1 Jednotkové testy

Jednotkové testy se používají pro testování základní funkcionality, vlastních tříd nebo metod. Nepoužívají žádný kontejner, běží pouze krátkou dobu a na stroji vývojáře. V současné době existuje pouze malé pokrytí kódu jednotkovými testy. Z tohoto důvodu bylo třeba napsat některé testy, převážně pro řetězce znaků a práci se soubory.

Pro jednotkové testy se v aktuální implementaci používá knihovna UnitTest++. UnitTest++ je multiplatformní knihovna pro jazyk C++ s jednoduchým rozhraním pro psaní testů. Dovoluje nám napsat jednotlivé testy a jednoduše testovat výsledky pomocí zprostředkovaných maker.

Jednotlivé testy můžeme rozdělit do různých kategorií, každá kategorie může obsahovat neomezené množství testů, výsledky můžeme porovnat několika různými způsoby. UnitTest++ zároveň podporuje využití vlastních mock tříd pro zjednodušení práce s testy. Všechny identifikátory používané v testech jsou C++ makra a jsou velice jednoduché na použití.

V této kapitole jsou popsány již implementované jednotkové testy a navrženy některé nové, které by bylo vhodné naimplementovat.

4.1.1 Řetězce znaků

Práce s řetězci znaků byla značně problémová část implementace a bylo důležité ji mít otestovanou na obou systémech. Z tohoto důvodu bylo napsáno několik testů testujících různou funkcionalitu třídy starající se o řetězce.

Stručný výpis z naimplementovaných testů:

Základní porovnání Základní porovnání obsahuje vytvoření řetězce jak pomocí normálního 8B char řetězce, tak přes wchar_t 16B řetězec. Všechny řetězce obsahují i některé české znaky.

Převody V tomto testu se snažíme ošetřit, zdali fungují všechny převody mezi řetězci typu char a řetězci typu wchar_t.

4.1.2 Serializace

Pro správné fungování prototypu bylo potřeba přepsat třídu starající se o serializaci dat tak, aby data ukládala v kódování UTF-8.

Základní porovnání Základní porovnání obsahuje vytvoření řetězce jak pomocí normálního 8B char řetězce, tak přes wchar_t 16B řetězec. Všechny řetězce obsahují i některé české znaky.

Vytvoření Tento test vytvoří novou Serialized entitu a uloží jí do souboru ve formátu UTF-8.

Porovnání Tento test vytvoří novou Serialized entitu a uloží jí do souboru. Poté otevře tuto stejnou entitu a porovná jí s druhou vytvořenou na druhé platformě.

4.1.3 Práce s registry

Na operačním systému Windows se používají registry pro ukládání různého nastavení stavu prototypu. Z tohoto důvodu bylo potřeba nalézt odpovídající ekvivalent na systému OS X. Bohužel takový ekvivalent neexistuje a bylo zapotřebí ho vytvořit za pomocí souborů s uloženými preferencemi.

Tyto testy v současné době neexistují, ale jsou zde napsané jako návrh do budoucna. Kvůli neustálým iteracím nad některými funkcemi je možné, že se některé části rozbijí a z tohoto důvodu by bylo vhodné tyto testy napsat.

Ukládání Prototyp otevře registry a dokáže do nich zapsat.

Čtení Prototyp otevře registry a přečte z nich požadovanou hodnotu. Tu následně porovná a vrátí výsledek.

Ukládání do neexistujících registrů Prototyp se pokusí otevřít a uložit do neexistujícího registru. Pokud registry neexistují, vytvoří nové a nainicializuje je na základní hodnoty. Poté zapíše hodnotu.

Čtení z neexistujících registrů Prototyp se pokusí otevřít a přečíst hodnotu z neexistujícího registru. Pokud registry před čtením neexistují, prototyp je vytvoří a vrátí prázdný řetězec, protože hodnotu nelze najít.

4.2 Integrační testy

Integrační testy můžeme rozdělit do dvou různých kategorií. Vnitřní integrační testy si berou za cíl otestovat chování jednotlivých komponent aplikace ve vzájemné spolupráci. Druhá kategorie integračních testů jsou vnější integrační testy. Ty mají za úkol otestovat chování aplikace vzhledem k různým systémům, konfiguracím a sestavám. V průběhu vývoje prototypu nás budou zajímat vnitřní integrační testy pro otestování správné spolupráce všech částí aplikace společně.

V současné verzi aplikace FurryBall neexistují integrační testy v pravém slova smyslu, nikde se totiž bohužel netestuje vzájemná funkcionalita všech částí dohromady. Nejblíže k integračním testům se můžeme dostat za pomoci benchmarku.

Benchmark je jedna scéna s přesně určeným nastavením. Konkrétně se jedná o 3 různé nastavení scény, první pouze s ambientním osvětlením, druhá s přímým osvětlením a poslední s nepřímým osvětlením a dalšími post process efekty. Údaje z tohoto benchmarku jsou poté odeslány na server, kde je prováděno zařazení do statistik, vždy ke správné grafické kartě.

Nejjednodušší variantou, jak vytvořit integrační testy, by bylo rozšířit již stávající benchmark o další funkcionalitu.

Více scén Pro zpřesnění výsledků bude zapotřebí vytvořit větší množství scén, které by byly renderovány v rámci benchmarku. Každá scéna by pak testovala jinou část, jiný efekt nebo jiný druh nastavení.

Více informací Na server bude zapotřebí odesílat více informací, pro příklad všechny vyrenderované obrázky a soubory s výstupem z renderu.

Lepší vyhodnocování na serveru Server bude porovnávat odeslané obrázky s již uloženými v databázi a vyhodnocovat rozdíly a výsledky.

Interpretace výsledků Je třeba změnit interpretaci výsledků a přidat filtrování podle operačního systému.

V závislosti na těchto změnách bude třeba rozšířit funkce serveru o porovnávání obrázku, zpracování a zobrazení dat. Na druhé straně půjde díky těmto změnám použít benchmark zároveň i jako vnější integrační testy. Benchmark je možné pouštět jak na vývojářském stroji tak i na stroji uživatelů. Zároveň si může každý uživatel tímto způsobem jednoduše ověřit, že jeho instalace pracuje správně a fungují všechny důležité části.

Pro zobrazení výsledků porovnání bude zapotřebí přidat pole pro více informací do výsledné GUI aplikace v plné verzi FurryBallu.

4.3 Regresní testy

Regresní testy se v současné době provádějí manuálně. Po přidání nové funkcionality některý z programátorů (nejlépe ten, kdo funkci nepřidával) otestuje ostatní části aplikace a ujistí se, že nová funkce nezměnila chování starého kusu kódu.

Po přidání nové funkcionality se vždy pustí jíž implementované jednotkové testy, které by v nejlepším případě měly otestovat zpětnou kompatibilitu. Bohužel pokrytí však není úplné, a proto by bylo vhodné přidat ještě další vrstvu testů.

Poměrně jednoduchou variantou regresních testů by bylo použití knihovny Boost::Serialize[12], která by nám umožnila vytvořit například XML soubor pro každou verzi, který by obsahoval výstupy z jednolivých částí aplikace. Tyto XML soubory bychom pak pomocí programu diff porovnali a zjistili rozdíly. Pokud by tyto rozdíly nebyly chtěné, našli jsme tak chybnou část aplikace.

4.4 QA & testování

Přestože jednotkové i integrační testy mohou odhalit velké množství chyb, stále se mohou vyskytnout jisté problémy, které tyto testy neodhalí. Ovšem hlavním důvodem nevyužívání jednotkových a integračních testů ve všech případech, je převážně nutná předchozí příprava.

Při vývoji aplikace FurryBall byl prováděn převážně typ testování, kdy programátor nebo jiný člen týmu otestoval manuálně přidanou funkcionalitu.

Tento způsob testování není optimální, ale při kombinaci s jednotkovými, integračními a regresními testy poskytuje dostatečnou kontrolu nad možnými chybami.

Můžeme říci, že k tomuto druhu testování patří i zpětná vazba, která přichází od uživatelů aplikace. Pro zpětnou vazbu má FurryBall zřízené fórum, na kterém mu jeho uživatelé poskytují cenné informace o problémech a chybách, které je třeba opravit. Zároveň je fórum i místem, kde se programátoři dozvídají o chtěných nových funkcích.

4.5 Uživatelské testování

Uživatelské testování není v současné chvíli v aplikaci FurryBall zapotřebí. FurryBall App je jednoduchá aplikace a je připravena pro potřeby graficky odborné veřejnosti, která je zvyklá na podobné rozložení.

Oba pluginy pro Autodesk Maya i Cinema 4D používají standartní GUI pro každý program, se kterým je dobře obeznámen každý jeho uživatel. Zároveň existuje podrobná dokumentace pro oba tyto pluginy i vlastní FurryBall App a rychlost odezvy na fórech je velice dobrá.

Závěr

Tato práce se zabývá analýzou, návrhem, implementací a výsledným testováním prototypu aplikace FurryBall na operačním systému OS X. Analytická část obsahuje rozbor obou operačních systémů a současně rozbor aplikace FurryBall fungující na operačním systému Windows. V návrhu je popsaný zvolený postup přepisu a v implementační části zvolené knihovny a vypsané překážky, které se objevily při přepisu. Z oblasti testování jsou v práci popsané použité jednotkové a návrh integračních, regresních a QA testů.

V této práci se podařilo ověřit návrh procesu přepisu z operačního systému Windows na OS X vytvořením prototypu aplikace FurryBall. Přepsání celé aplikace FurryBall je mimo rozsah této bakalářské práce.

Výsledkem práce je prototyp, aplikace, která v současné době funguje pouze na vývojářském stroji, ale je možné vytvořit standardní OS X aplikaci s malým či žádným úsilím. Prototyp zároveň funguje s omezenými možnostmi, dokáže vykreslit malé scény bez většího množství textur. Nefungují zde některé příkazy do konzolové aplikace, převážně z důvodu chybějícího licencovacího modulu.

Jelikož je hotový jen prototyp a né celá aplikace, je pro vytvoření plné verze některé části, zejména modul starající se o licencování a samotnout GUI aplikaci. Kvůli povaze aplikace a jejímu napojení na operační systém Windows je možné, že se při přepisování těchto dvou částí vyskytnou problémy, které mohou tvorbu výsledné aplikace zpomalit.

Literatura

[1] Boost 1.61.0 Library Documentation [online]. [cit. 8.5.2016]. Dostupné z: http://www.boost.org/doc/libs/1_61_0/

[2] Introducing JSON [online]. [cit. 15.5.2016]. Dostupné z: http://www.json.org/

[3] Linux Programmer's Manual [online]. [cit. 15.4.2016]. Dostupné z: http://man7.org/linux/man-pages/man2/syscalls.2.html

[4] UnitTest++ [online]. [cit. 8.5.2016]. Dostupné z: https://github.com/unittest-cpp/unittest-cpp

[5] Apple, Inc.: Bundle Structures [online]. [cit. 18.6.2016]. Dostupné z: https://developer.apple.com/library/mac/documentation/CoreFoundation/Conceptual/CFBundles/BundleTypes/BundleTypes.html#//apple_ref/doc/uid/10000123i-CH101-SW19

[6] Apple, Inc.: HFS Plus Volume Format [online]. [cit. 14.4.2016]. Dostupné z: https://developer.apple.com/legacy/library/technotes/tn/tn1150.html

[7] Apple, Inc.: Xcode [online]. [cit. 20.4.2016]. Dostupné z: https://developer.apple.com/library/ios/documentation/ToolsLanguages/Conceptual/Xcode_Overview/

[8] Art And Animation studio: GPU vs CPU - GPU rendering vs Software CPU rendering - Arnold [online]. [cit. 3.6.2016]. Dostupné z: http://furryball.aaa-studio.eu/images/Arnold41.jpg

[9] Art And Animation studio: GPU vs CPU - GPU rendering vs Software CPU rendering - FurryBallRT [online]. [cit. 3.6.2016]. Dostupné z: http://furryball.aaa-studio.eu/images/FurryBall-RT-14_1.jpg

[10] Bikker, J.: Ray tracing in real-time games. 2013, [cit. 17.4.2016].

[11] Bikker, J.: Ray Tracing for Games, 2014, [cit. 17.4.2016].

[12] Boost Software: Serialization [online]. [cit. 28.6.2016]. Dostupné z: http://www.boost.org/doc/libs/1%5F39%5F0/libs/serialization/doc/index.html

[13] cplusplus.com: C++ Reference - mbstowcs [online]. [cit. 1.5.2016]. Dostupné z: http://www.cplusplus.com/reference/cstdlib/mbstowcs/

[14] cplusplus.com: C++ Reference - setlocale [online]. [cit. 1.5.2016]. Dostupné z: http://www.cplusplus.com/reference/clocale/setlocale/

[15] cplusplus.com: C++ Reference - wcstombs [online]. [cit. 1.5.2016]. Dostupné z: http://www.cplusplus.com/reference/cstdlib/wcstombs/

[16] GCC team: GNU gcc [online]. [cit. 23.4.2016]. Dostupné z: https://gcc.gnu.org/onlinedocs/gcc-5.4.0/gcc/

[17] GDB Developers: GDB: The GNU Project Debugger [online]. [cit. 24.4.2016]. Dostupné z: https://www.gnu.org/software/gdb/

[18] Greg Roelofs: Lib PNG [online]. [cit. 8.5.2016]. Dostupné z: http://www.libpng.org

[19] Industrial Light & Magic: OpenEXR [online]. [cit. 8.5.2016]. Dostupné z: http://www.openexr.com

[20] JetBrains s.r.o.: CLion [online]. [cit. 20.4.2016]. Dostupné z: https://www.jetbrains.com/clion/documentation/

[21] Khronos OpenCL Working Group and others: The opencl specification. *Version*, ročník 2, č. 1, 2015, [cit. 10.5.2016].

[22] Kitware, Inc.: CMake [online]. [cit. 22.4.2016]. Dostupné z: https://cmake.org/Wiki/CMake

[23] LLVM Project: The LLDB Debugger [online]. [cit. 24.4.2016]. Dostupné z: http://lldb.llvm.org

[24] LLVM Project: LLVM [online]. [cit. 22.4.2016]. Dostupné z: http://llvm.org/docs/

[25] Ludvigsen, H.; Elster, A. C.: Real-time ray tracing using nvidia optix. *Eurographics Short Papers*, 2010: s. 65–68, [cit. 10.5.2016].

[26] Max Howell: Homebrew [online]. [cit. 15.5. 2016]. Dostupné z: https://github.com/Homebrew/brew/blob/master/share/doc/homebrew/README.md

[27] Microsoft, Inc.: Chapter 1: Functional Comparison of UNIX and Windows [online]. [cit. 14.4.2016]. Dostupné z: https://technet.microsoft.com/en-us/library/bb496993.aspx

[28] Microsoft, Inc.: The evolution of the Windows family of operating systems [online]. [cit. 14.4.2016]. Dostupné z: https://technet.microsoft.com/en-us/library/bb496993.uamv1c1_01_big(l=en-us).gif

[29] Microsoft, Inc.: Microsoft Visual Studio [online]. [cit. 20.4.2016]. Dostupné z: https://msdn.microsoft.com/en-us/library/dn707591.aspx

[30] Microsoft, Inc.: What Is NTFS? [online]. [cit. 14.4.2016]. Dostupné z: https://technet.microsoft.com/en-us/library/cc778410(v=ws.10).aspx

[31] Microsoft, Inc.: Windows API Index [online]. [cit. 15.4.2016]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/ff818516(v=vs.85).aspx

[32] MinGW.org: MinGW gcc [online]. [cit. 24.4.2016]. Dostupné z: http://www.mingw.org/wiki

[33] Ni, Tianyun: Direct Compute: Bring GPU computing to the mainstream. In *GPU Technology Conference*, 2009, str. 23, [cit. 10.5.2016].

[34] NVIDIA Corporation: CUDA [online]. [cit. 30.4.2016]. Dostupné z: http://www.nvidia.com/object/cuda_home_new.html

[35] Oracle Corporation: NetBeans [online]. [cit. 20.4.2016]. Dostupné z: https://netbeans.org/kb/

[36] Pixar: OpenSubdiv Introduction [online]. [cit. 8.5.2016]. Dostupné z: http://graphics.pixar.com/opensubdiv/docs/intro.html

[37] Red Hat, Inc.: Cygwin [online]. [cit. 24.4.2016]. Dostupné z: https://www.cygwin.com

[38] Red Hat, Inc.: Debugging Cygwin Programs [online]. [cit. 24.4.2016]. Dostupné z: https://cygwin.com/cygwin-ug-net/gdb.html

[39] SmartSVN GmbH: SmartSVN [online]. [cit. 12.5.2016]. Dostupné z: http://www.smartsvn.com

[40] The Qt Company: Qt Documentation [online]. [cit. 4.5.2016]. Dostupné z: http://doc.qt.io/qt-5/index.html

[41] Tom Lane: Lib Jpeg[online]. [cit. 8.5.2016]. Dostupné z: http://libjpeg.sourceforge.net

[42] Unicode, Inc.: The Unicode Consortium [online]. [cit. 6.6.2016]. Dostupné z: http://www.unicode.org

[43] Wikipedia: Vývoj unixových systémů. [online]. [cit. 14.4.2016]. Dostupné z: https://cs.wikipedia.org/wiki/Unix{#}/media/File:Unix_history-simple.png

[44] Wine authors: WINE HQ [online]. [cit. 13. 5. 2016]. Dostupné z: https://www.winehq.org

Příloha A

Seznam použitých zkratek

CPU Central Processing Unit

GPU Graphic Processing Unit

CUDA Compute Unified Device Architecture

GUI Graphical User Interface

SPP Sample Per Pixel

HTML HyperText Markup Language

XML Extensible Markup Language

CSS Cascade Style Sheets

TCP/IP Transmission Control Protocol/Internet Protocol

UPD User Datagram Protocol

API Application Programming Interface

ABI Application Binary Interface

FIT Fakulta Informačních Technologií

JSON JavaScript Object Notation

MSVC Microsoft Visual C++

QA Quality Assurance (Zaručení kvality)