

Na tomto místě bude oficiální zadání vaší práce

- Toto zadání je podepsané děkanem a vedoucím katedry,
- musíte si ho vyzvednout na studijním oddělení Katedry počítačů na Karlově náměstí,
- v jedné odevzdané práci bude originál tohoto zadání (originál zůstává po obhajobě na katedře),
- ve druhé bude na stejném místě neověřená kopie tohoto dokumentu (tato se vám vrátí po obhajobě).

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačové grafiky a interakce



Diplomová práce

Efektivní zobrazování rozsáhlých nočních scén na mobilních zařízeních

Bc. Luboš Vonásek

Vedoucí práce: Ing. Jiří Bittner, Ph.D.

Studijní program: Otevřená informatika, Navazující magisterský

Obor: Počítačová grafika a interakce

19. února 2014

Poděkování

Zde můžete napsat své poděkování, pokud chcete a máte komu děkovat.

Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 15. 5. 2014

.....

Abstract

Translation of Czech abstract into English.

Abstrakt

Abstrakt práce by měl velmi stručně vystihovat její podstatu. Tedy čím se práce zabývá a co je jejím výsledkem/přínosem.
Očekávají se cca 1 – 2 odstavce, maximálně půl stránky.

Obsah

1	Zadání	1
2	Specifikace cílů	3
2.1	Předzpracování	3
2.2	Simulátor	3
2.3	Existující implementace	4
2.3.1	Asphalt Urban od Gameloftu	4
2.3.2	Need for Speed Most Wanted od EA Games a Firemonkeys	4
2.3.3	GT Racing 2 od Gameloftu	5
2.3.4	Sports Car Challenge od Fishlabs	5
2.3.5	Track Racing od vývojáře Soulkey	5
3	Rešerše	7
3.1	Předvypočítané osvětlení	7
3.1.1	Vytvoření texturovacích souřadnic	7
3.1.2	Generování lightmap	8
3.1.3	Oprava chyby sousedících trojúhelníků	9
3.2	Odlesky povrchů	9
3.3	Screen-space přístup	10
4	Realizace	13
4.1	Realizace na platformě Android	13
4.1.1	Java native interface	13
4.1.2	Využití vláken	14
4.1.3	Práce se soubory	14
4.2	Nahrazení víceprůchodových přístupů	14
4.2.1	Rozměr textur na OpenGL ES 2.0	15
4.2.2	Efekty pomocí screen-space přístupu	15
4.3	Předvypočítané osvětlení	17
4.4	Světla vozidel	18
5	Testování	19
6	Závěr	23
	Literatura	25

A Testování zaplnění stránky a odsazení odstavců	27
B Pokyny a návody k formátování textu práce	31
B.1 Vkládání obrázků	31
B.2 Kreslení obrázků	32
B.3 Tabulky	32
B.4 Odkazy v textu	33
B.4.1 Odkazy na literaturu	33
B.4.2 Odkazy na obrázky, tabulky a kapitoly	34
B.5 Rovnice, centrovaná, číslovaná matematika	34
B.6 Kódy programu	35
B.7 Další poznámky	36
B.7.1 České uvozovky	36
C Seznam použitých zkratek	37
D UML diagramy	39
E Instalační a uživatelská příručka	41
F Obsah přiloženého CD	43

Seznam obrázků

2.1 Asphalt 7	4
2.2 Asphalt 8	4
2.3 Need for Speed Most Wanted mobilní verze Zdroj:www.androidpolice.com . .	4
2.4 GT Racing 2 Zdroj:www.youtube.com	5
2.5 Sports Car Challenge	5
2.6 Track Racing	5
3.1 kD strom v metodě Packing Lightmaps	8
3.2 Model města transformován do souřadnic lightmapy	8
3.3 Princip stínových map	9
3.4 Diagram odrazu vodní hladiny	10
3.5 Odraz objektů v screen-space přístupu	10
4.1 Vykreslovací řetězec simulátoru	15
4.2 Odlesky na povrchu vozovky	16
4.3 Odlesky na vozidle. Bílé plochy znázorňují pozice, ze kterých se čte barva odlesku.	17
4.4 Scéna s aplikovanými lightmapami lamp	18
B.1 Popiska obrázků	32

Seznam tabulek

5.1	Parametry zařízení, na kterých jsem projekt testoval	19
5.2	Závislost rychlosti vykreslování scény na zařízení a poměru rozlišení oproti nativnímu(v závorce je uvedena konfigurace v nastavení simulátoru). Z každého měření je uvedená minimální a maximální hodnota	20
5.3	Závislost rychlosti vykreslování scény na platformě a rozlišení textur lightmap při vizuálních detailech normal(poměr 0.6x ku nativnímu rozlišení). Z každého měření je uvedená minimální a maximální hodnota	20
5.4	Závislost využití paměti na rozlišení textur lightmap. Testováno na Acer TravelMate P253-e (laptop)	20
B.1	Ukázka tabulky	32

Kapitola 1

Zadání

Prostudujte metody efektivního zobrazování rozsáhlých scén se zaměřením na low-end mobilní zařízení. Soustředěte se na metody využívající předpočtenou informaci o osvětlení ve formě map osvětlení (light map). Navrhněte metodu pro aktualizaci map osvětlení pro dynamické scény s omezeným počtem měnících se světel. Konkrétní testovací scénou bude model nočního města s několika pohybujícími se vozidly a několika zhasínajícími lampami pouličního osvětlení. Navrženou metodu implementujte v jazyce C++ s využitím OpenGL. Výsledky implementace otestujte a zhodnoťte její paměťovou a výpočetní efektivitu..

Kapitola 2

Specifikace cílů

Samotnou realizaci projektu lze rozdělit do dvou dílčích částí. První částí je předzpracování, ve kterém se vytváří mapy osvětlení. To znamená vytvoření koordinát map a vykreslení hodnot do lightmap. Druhou částí je samotný simulátor, který s těmito předzpracovanými hodnotami pracuje.

2.1 Předzpracování

Ve fázi předzpracování se provede konverze 3D modelů do vlastního formátu, který bude umožňovat mít v sobě jak souřadnice standardních textur, tak i souřadnice v lightmapách. Formát by neměl zahazovat informace o modelu. I když je třeba výsledný simulátor nevyužije, je dobré tyto informace ponechat pro případ dalšího vývoje. Formát by měl být schopen uložit některé další informace o materiálu a shaderu.

Souřadnice lightmap je nutné co nejvíce nahustit z důvodu očekávané malé paměti vymezeny pro velké textury. Aby bylo možné určit nejvhodnější rozlišení těchto textur, je třeba umožnit jejich škálování.

Vykreslení hodnot do lightmap se provede pomocí OpenGL, bude zde řešena viditelnost(pomocí stínových map), vypočteno utlumení a výpočet difuzního odrazu světla.

2.2 Simulátor

Simulátor bude navržen tak, aby fungoval na X11-Linuxu a Androidu. Znamená to výběr multiplatformních knihoven nebo ve výjimečných případech rozdílná implementace pro jednotlivé platformy. K vykreslování jsem vybral OpenGL ES 2.0(jedná se o odlehčené OpenGL 3.0) a GLSL verze 1.0(to zaručí maximální možnou kompatibilitu).

Fyzika bude realizována pomocí Bullet Physics(tento engine používá mnoho známých firem zabývajících se vývojem mobilních her). Zvuk bude realizován na mobilních zařízeních pomocí Android API a na Linuxu pomocí knihovny FMOD API. Správce oken je na Androidu třeba doprogramovat, aby fungovalo dotykové ovládání a pozastavení simulátoru(např při hovoru). Na Linuxu bude použitý freeglut.

2.3 Existující implementace

2.3.1 Asphalt Urban od Gameloftu



Obrázek 2.1: Asphalt 7

Asphalt Urban je série mobilních závodních simulátorů, jejichž první díl byl publikován spolu s herním smartphonem Nokia N-Gage. Jedná se o úspěšnou sérii, která přitahuje hráče všech možných platform.

Na obrázku 2.1 můžete vidět 7.díl simulátoru. Osvětlení z pouličních lamp je vykresleno přímo do textur, reflektor vozidla je tvořen zřejmě pomocí projektivní textury. Povrch zrcadlově odráží 3D objekty. Po bližším zkoumání lze zjistit, že některé tyto odražené objekty jsou odlišné. Domnívám se, že zde byl použit duplicitní objekt.

Další díl ze série Asphalt Urban už odlesky řeší lépe. Odráží se hlavně světla a odlesk je lépe přizpůsoben povrchu. Vzniká zde dojem mokré silnice. Dochází i k rozmanitému brzdovým světel, od kol cáká voda. Osvětlení od lamp je opět řešeno texturou. Nepříjemnou změnou je absence reflektoru vozidla a přidání rušivého chvění kamery během jízdy.



Obrázek 2.2: Asphalt 8

2.3.2 Need for Speed Most Wanted od EA Games a Firemonkeys



Obrázek 2.3: Need for Speed Most Wanted mobilní verze
Zdroj:www.androidpolice.com

Simulátor, který je spíše známý z PC, na mobilním trhu takový úspěch nemá. Od PC verze je velice odlišný, ale rozumně se nejedná o nějakou lacinou napodobeninu. Jedná se o první závodní simulátor pro mobilní zařízení, který disponuje modelem ničení vozidla (je možné vozidlo poškrábat, rozbít mu okna apod.).

Oproti Asphalt 8 má navíc efekt Depth-of-field, to znamená, že vzdálené modely jsou rozmazané a zabarveny do barvy pozadí. Tento efekt vytváří přijemný mlhovitý efekt. Jinak bych řekl, že tyto dva simulátory jsou si sobě velice podobné. Na stejném enginu funguje i známý simulátor Real Racing 3, který ale noční jízdu nemá.

2.3.3 GT Racing 2 od Gameloftu



Obrázek 2.4: GT Racing 2
Zdroj: www.youtube.com

Mým oblíbeným závodním simulátorem je v současné době GT Racing 2, je to hlavně díky jeho zpracováním a tím, že je zdarma. Jako jediný z uvedených simulátorů má při noční jízdě nízké ambientní osvětlení a scéna je osvětlena hlavně reflektorem vozidla.

Nejsem si jist, jestli osvětlení lampami je vykresleno přímo do textury nebo jestli jsou zde použity lightmapy. Řekl bych, že se jedná o lightmapy.

2.3.4 Sports Car Challenge od Fishlabs



Obrázek 2.5: Sports Car Challenge

Asi nejdetailejší model vozidla má právě Sports Car Challenge. Vývojář spolupracuje s předními výrobci automobilů a to se odrazilo právě na modelech vozidel. Modely jsou realistické a to včetně interiérů.

Projekt trpí slabší kompatibilitou s mobilními zařízeními, i když náročnost enginu je nízká. Co se týká noční jízdy, je realizována stejnými technikami jako denní jízda. Jsou zde použity pouze tmavé textury a halo efekt pouličních lamp.

2.3.5 Track Racing od vývojáře Soulkey



Obrázek 2.6: Track Racing

I když tento projekt není závodní simulátor s noční jízdou, zmiňuji ho hlavně kvůli ovládání. Jedná se o neoficiální remake hry Trackmania známé z PC. Projekt běží na Unity3D enginu a je dostupný na mnoha platformách, ale nedá se stáhnout přímo z Marketu či Storu.

U výše uvedených simulátorů vozidlo automaticky zrychluje a u některých i samo brzdí. Hráč nemá tedy nad vozidlem takovou kontrolu. V případě Track Racing má hráč plnou kontrolu a zážitek ze hry se dá srovnat se zážitkem z PC hry.

Kapitola 3

Rešerše

3.1 Předvypočítané osvětlení

Předvypočítané osvětlení je z hlediska vykreslování velice jednoduchá technika. Pouze se aplikuje další textura či textury, které mají vlastní texturovací souřadnice a nesou v sobě informaci o osvětlení jednotlivých trojúhelníků. Těmto texturám se říká lightmapy a s drobným nadhledem by se dalo říct, že se jedná o techniku, která rozšiřuje techniku předvypočítaných stínů.

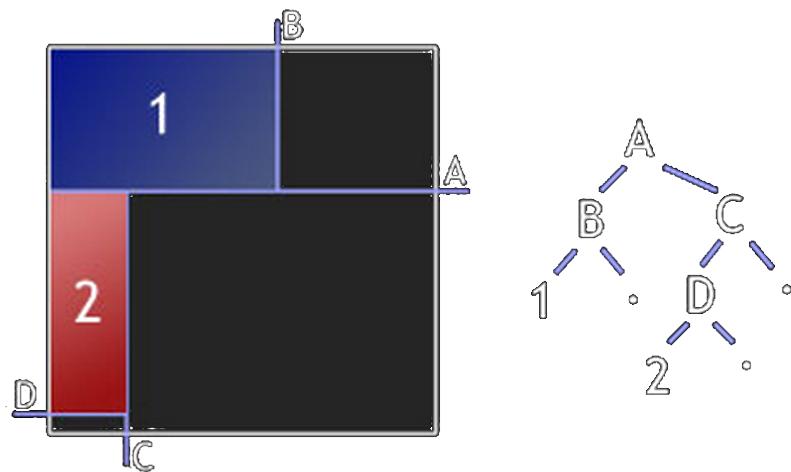
Z hlediska generování těchto textur je tato technika náročnější, skládá se z několika kroků. V prvním kroku je třeba vytvořit texturovací souřadnice pro jednotlivé trojúhelníky.

3.1.1 Vytvoření texturovacích souřadnic

Jeden z velmi efektivních algoritmů řešící tuto problematiku je Packing Lightmaps [Sco02]. Za využití datové struktury kD stromu, která u tohoto algoritmu reprezentuje rozložení geometrie v lightmapě, je možné řešit rozložení pro obdélníky. Aby bylo možné pracovat s trojúhelníky, je nutné algoritmus rozšířit.

Práci s trojúhelníky lze realizovat tak, že vždy dva trojúhelníky spolu vytvoří jeden obdélník. Nejdříve se provede drobná deformace tak, aby všechny trojúhelníky byly pravoúhlé (je nutné hlídat, aby se nesnižoval počet texelů připadající na daný trojúhelník). Poté už stačí trojúhelníky vhodně spárovat (vytvořit z nich obdélníky) a je možné spustit původní algoritmus.

Jak je kD strom použit je vidět na následujícím obrázku. V uzlech jsou provedeny řezy, kterými se rozděluje rovina, v listech jsou pak jednotlivé objekty.



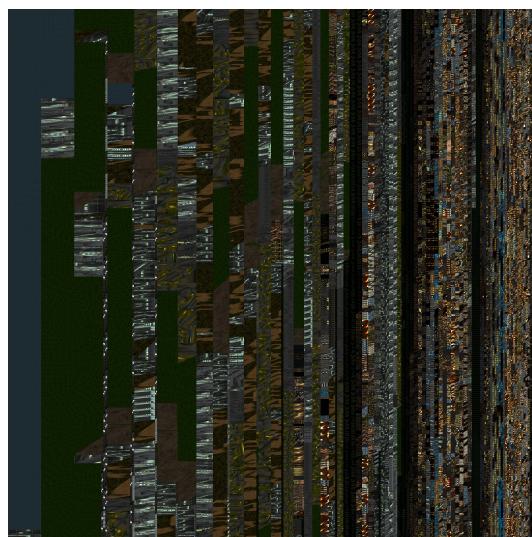
Obrázek 3.1: kD strom v metodě Packing Lightmaps

Zdroj: [[Sco02](#)]

3.1.2 Generování lightmap

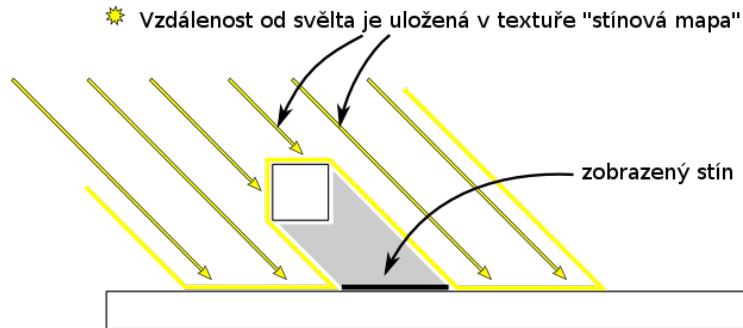
Generování lightmap lze provádět pomocí rasterizace nebo vrhání paprsku. Vrhání paprsku je v porovnání s rasterizací daleko složitější metoda. V rámci předmětu softwarový projekt se zabývám pouze rasterizací.

Generování se provádí pomocí standardního rasterizačního řetězce. Geometrie se netransformuje do souřadnic kamery, ale do souřadnic lightmapy. Otexturovanou geometrii do souřadnic lightmapy můžete vidět na obrázku 3.2.



Obrázek 3.2: Model města transformován do souřadnic lightmapy

Dále je potřeba vyfiltrovat objekty, které nejsou viditelné. K tomu slouží stínová mapa. Stínová mapa je textura, která v sobě má uloženou vzdálenost od světla pro velké množství bodů. Při vykreslování se spočítá souřadnice bodu ve stínové mapě, z toho se získá vzdálenost bodu od světla. Porovnáním vzdálenosti od světla vykreslovaného bodu se zjistí, zda je bod osvětlený daným světlem. Viz obrázek níže.



Obrázek 3.3: Princip stínových map

Zdroj: [Mas11]

Tímto způsobem vyfiltrujeme neosvětlené objekty. Výsledek lze vylepšit využitím dalších technik jako je například útlum světla, PCF filtrování a další.

3.1.3 Oprava chyby sousedících trojúhelníků

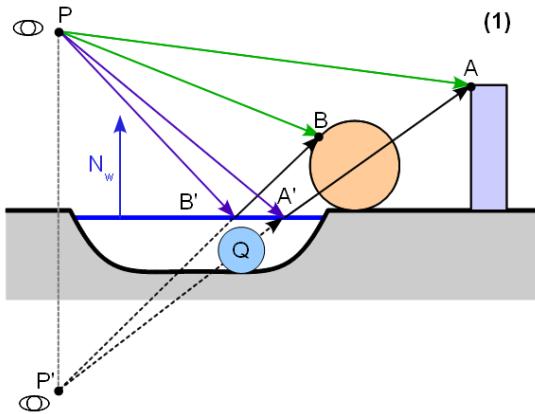
Pokud bychom použili přímo souřadnice získané metodou Packing Lightmaps, vznikly by ve výsledné scéně chyby (buď v důsledku zaokrouhlování souřadnic nebo v případě lineárního filtrování vlivem sousedních texelů). Je proto nutné už při generování lightmap souřadnice posunout nepatrнě směrem ke středu každého trojúhelníku. Po dokončení generování lightmap se provede postprocessing, který doplní mezery v lightmapách vždy nejbližší možnou hodnotou.

3.2 Odlesky povrchů

Odlesky dodávají scéně lepší vzhled, dodávají dojem buď kovového nebo mokrého povrchu. V rasterizačním řetězci jsou odlesky drobet problematické, protože grafická karta vždy zpracovává pouze aktuální geometrii a nemá informaci o okolí.

Standardně se odlesk realizuje dalším průchodem, ve kterém se vykreslí okolí do textury a tato textura se pak aplikuje na model s odleskem. U menších objektů jako je třeba auto se vykreslí okolí ve formě cubemapy (krychle, která má na svých stěnách obrazy z okolí objektu). Cubemapa se následně „nalepí“ na objekt a tím se získá lesklý povrch.

V případě rovných povrchů je nutné použít jiný přístup, scéna se vykreslí do textury z pohledu, který je na obrázku 3.4 označen jako P' . Pohled snímá objekty jakoby z povrchu (všechny objekty pod povrchem je třeba vyfiltrovat). Při vykreslování na obrazovku se spočítá pozice daného pixelu povrchu ve vykreslené textuře a z té se aplikuje barva pixelu.



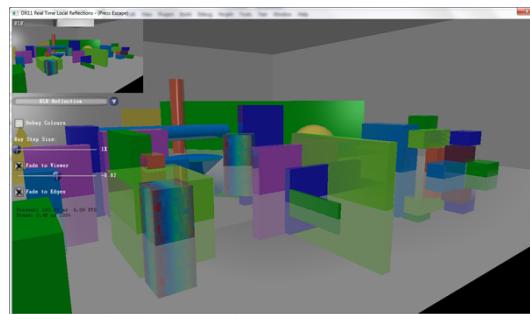
Obrázek 3.4: Diagram odrazu vodní hladiny

Zdroj: [\[Kap11\]](#)

3.3 Screen-space přístup

Základem screen-space přístupu je vykreslení scény do takzvaného geometry bufferu, který umožňuje na rozdíl od frame bufferu uložit daleko více informací o jednotlivých bodech. Dalšími informacemi typicky bývá normála, hloubka nebo informace k výpočtu spekulární složky světla. Geometry buffer je součástí OpenGL 4 a vyšší, ale některé poznatky o screen-space přístupu se dají použít i za pomocí staršího OpenGL.

Pomocí screen-space přístupu se dá implementovat mnoho efektů. V podstatě by bylo možné si v hlavním průchodu pouze vykreslit scénu pomocí velice jednoduchého shaderu a osvětlovací model, případně další efekty, řešit až v průchodu pro screen-space data. Problém je, že screen-space průchod je typicky realizován pomocí jednoho shaderu a tedy musí být veškeré efekty a materiály implementovány v tomto shaderu.



Obrázek 3.5: Odraz objektů v screen-space přístupu

Zdroj: [\[Pfa12\]](#)

Problém nastává, pokud jsou potřeba číst informace, které jsou mimo rozmezí geometry bufferu. Tento případ nastává typicky při pokusu o vykreslení odrazivých ploch. Problém se často řeší absencí tohoto efektu pokud nejsou k dispozici potřebné informace. Aby absence nebyla tak rušivá, provádí se plynulý přechod mezi místy, kde efekt je a kde není(tedy jakoby na kraji geometry bufferu byl přechod do ztracená).

Odrazy zrcadel v budově se pochopitelně pomocí screen-space nedají řešit. Obecně lze říct, že odrazy pomocí screen-space lze realizovat jen u objektů, na které se kamera nebude dívat kolmo.

Kapitola 4

Realizace

Základním problémem realizace simulátoru na mobilních zařízení je jejich nízký výkon. Uživatelé očekávají kvalitu grafiky srovnatelnou s počítačovou grafikou, nicméně mobilní zařízení zatím takového výkonu nedosahují.

4.1 Realizace na platformě Android

Platforma Android je postavená na Linuxovém jádře a využívá mnohé komponenty z projektu GNU, nicméně celé Android API je v jazyku Java. Android využívá svůj Java virtuální stroj Dalvik. Pomocí Android NDK(native development kit) je možné zkompilovat C/C++ kód jako knihovnu, kterou je možné zavolat z jazyku Java. Toto je prováděno pomocí JNI(Java native interface).

Snažší by bylo naprogramovat simulátor přímo v Javě, ale tím bych ztratil možnost portovat simulátor na další platformy, které virtuální stroj Dalvik nemají. Dále by byl problém s použitím fyzikálního engine, který je napsaný v jazyce C++. Musel bych napsat interface pro fyzikální engine, který by mi umožnil komunikaci mezi C++ a Java kódem.

4.1.1 Java native interface

Java native interface je rozhraní, které umožňuje spustit C++ kód z jazyku Java a obráceně. Nicméně komunikaci nelze realizovat nějak jednoduše a je proto vhodné tuto komunikaci minimalizovat.

```
jclass cl = env->FindClass(className);
jmethodID m = env->GetStaticMethodID(cl, methodName, paramFormat);
env->CallStaticVoidMethod(cl, m, id, volume);
```

Ukázka 4.1: Volání Java metody z jazyku C++

Udělal jsem tedy rozhraní, které odpovídá handlerům knihovny freeglut. Tedy nahrazuji veškeré funkce této knihovny svou implementací. Knihovna implementuje správu oken, vstup klávesnice a myši a volání idle a display funkcí. Klávesnice a myš byla pochopitelně plně nahrazena dotykovým ovládáním, ale pokud do Android zařízení připojíme klávesnici nebo myš, bude plně funkční. Funkčnost myši zařizuje sám systém, klávesnice stačila namapovat.

4.1.2 Využití vláken

Většina zařízení, které dnes už spadají do střední třídy, je vybavena aspoň dvěma procesory, je proto vhodné rozdělit činnost do více vláken, aby bylo dosaženo maximálního výkonu. Třída GLSurfaceView, která umožňuje vykreslovat OpenGL scénu z C++ umožňuje volat OpenGL příkazy pouze z jednoho vlákna. Z tohoto důvodu jsem rozdělil kód na grafické a negrafické vlákno.

V menu simulátoru se používá pouze grafické vlákno, negrafické vlákno je primárně použito pro fyzikální výpočty, které zařízení vytěžuje nejvíce. Vlákna jsou synchronizovaná, aby nedocházelo k použití transformace vozidla z předešlého snímku (to by bylo velmi rušivé).

Dále provádím regulaci rychlosti simulátoru v závislosti na výkonu hardwaru. Ukázalo se, že je daleko příjemnější hru zpomalit, než aby docházelo k trhanému zobrazení. Problém je, že nelze zpomalovat rychlosť simulátoru do nekonečna, proto je zde nastavena minimální hodnota, na kterou se může simulátor zpomalit. Je zde i horní limit, aby nedocházelo k zrychlené jízdě a tedy i ke zvýšené spotřebě energie.

4.1.3 Práce se soubory

V Androidu se aplikace publikují v APK balíčku. Jedná se o ZIP soubor, který má nějakou danou strukturu. V Java kódu je možné na veškeré soubory uvnitř balíčku přistupovat, v C++ tato možnost není. Některí vývojáři toto řeší, že po prvním spuštění si aplikace stáhne ze serveru přídavná data a ty si uloží například na paměťovou kartu. Lze také rozbalit soubory z APK balíčku a s těma pak v C++ pracovat. To se moc nepoužívá, protože jsou potom data v zařízení dvakrát.

Další možnou variantou je přistupovat k APK balíčku jako k ZIP souboru. Knihovna libzip umožňuje zpracovávat data ze ZIP souboru jako kdyby byla uložena normálně na disku nebo na paměťové kartě. To má výhodu, že data jsou komprimovaná a zároveň přístupná. Tato varianta je tedy lepší než stahovat data ze serveru.

4.2 Nahrazení víceprůchodových přístupů

Při běhu jakéhokoliv simulátoru je největší zátěž mobilního hardware způsobena vykreslováním 3D scény, proto je nutné tuto část co nejvíce zefektivnit. Pro mnoho efektů se používají takzvané víceprůchodové přístupy (scéna se vykreslí víckrát z různých pohledů nebo pomocí různých shaderů), ale na mobilním zařízení je zatím z hlediska výkonu možné použít pouze jednoprůchodový přístup.

Abych částečně nahradil ve svém simulátoru víceprůchodový přístup, použil jsem screen-space přístup (to je obecně jakákoliv technika využívající data z pohledu kamery, jinými slovy máme pouze výsledný snímek a na něm provádíme další operace). V praxi se používá hlavně pro zrcadlové plochy. Potřeboval jsem přistupovat do aktuálně vykreslovaného snímku. To je ale problém, protože tento snímek není kompletní a kdybych vykresloval přímo do něj a zároveň z něj četl způsobil by to chyby ve výsledném obraze.

Řešením je vykreslování strídavě do dvou různých textur. To znamená, že při lichém snímku kreslím do textury 1 a čtu z textury 2. Při sudém snímku je tomu obráceně. Tímto

způsobem získám přístup do vykreslené scény, sice s drobným zpožděním, ale při dostatečně rychlém vykreslování si toho uživatel nevšimne(za předpokladu, že textura bude použita na efekty jako jsou odlesky nebo stíny).



Obrázek 4.1: Vykreslovací řetězec simulátoru

4.2.1 Rozměr textur na OpenGL ES 2.0

Verze OpenGL ES 2.0 neumožnuje používat obdélníkové textury. Textury musí být čtvercové a jejich strana musí být o velikosti mocniny 2. Pro vykreslování do textury, která bude výsledně umístěna na obrazovku je tedy nutné zvolit vyšší rozlišení. Např. u tabletu Google Nexus 7 2012 s rozlišením 1280x720 se zvolí textura 2048x1024. To znamená, že by se ve výsledku vykreslilo víc než 2x více pixelů než je potřeba.

Není ale nutné vykreslovat do celé textury, pomocí příkazu *glViewport* se dá nastavit část textury, do které má OpenGL vykreslovat a nedojde tedy k plýtvání výkonem(bude se plýtvat pouze pamětí).

Dále je možné zvolit nižší rozlišení viewportu než je rozlišení displeje. Mobilní zařízení obvykle disponují obrovskou hustotou pixelů a pokud zařízení nebude stíhat vykreslovat scénu v plném rozlišení, lze vykreslovat v rozlišení menším. U počítačových 3D her se také setkáváme s možností změnit rozlišení, tam řeší změnu přímo hardware displeje nebo jeho ovladač.

4.2.2 Efekty pomocí screen-space přístupu

Jedním z velmi rychlých efektů ve screen-space je **motion-blur**.

```

uniform sampler2D color_texture , pFrm;
uniform float res , speed;
varying vec2 v_Coords;

void main() {
    //apply color texture
    gl_FragColor = texture2D(color_texture , v_Coords);

    //apply motion blur
    gl_FragColor *= (1.0 - speed);
    gl_FragColor += speed * texture2D(pFrm , gl_FragCoord.xy * res);
}

```

Ukázka 4.2: Motion blur fragment shader dle rychlosti vozidla

Speed je rychlosť vozidla od 0 do 1, pFrm je textura předchozího snímku a res je 1 / plné rozlišení textury(pro jednoduchosť je použitý čtvercový rozměr textury).

Tento kód je použit přímo při renderování modelů, nikoliv při postprocesu. Výhodou je, že rozmaznutí se projeví po několik snímků za sebou pomocí jednoho čtení textury navíc. Nevýhodou je, že tento kód musí být vložen do každého shaderu vykreslující 3D model.

Dále je možné ve screen-space řešit **odlesky**. Existuje obecný(složitý) postup jak tyto odlesky vypočítat, který by mobilní hardware nezvládl. Problém jsem rozdělil podle typu ploch. Vzhledem k tomu, že v simulátoru dochází pouze k rotaci pohledu podle osy y(rotace yaw), je možné řešit pro odlesky pro různé směry normál zvlášť.

Například u vozovky stačí nalézt, kde vozovka končí, a podle toho obraz převrátit. Korektně to pro zatím z hlediska výkonu není možné vypočítat, proto dochází k velké nepřesnosti odlesku. Odhad zlomu je prováděn pomocí normály a pozice daného fragmentu, do kterého se odlesk započítává.



Obrázek 4.2: Odlesky na povrchu vozovky.

U vozidel se odlesk provádí pouze podle normály. Pro výpočet pozice odraženého obrazu se vezme střed vozidla a přičte se k němu normála vynásobena určitým faktorem(funguje pouze u vozidla, které uživatel ovládá).



Obrázek 4.3: Odlesky na vozidle. Bílé plochy znázorňují pozice, ze kterých se čte barva odlesku.

Pokud máme ve scéně **předvypočítané stíny**, je vhodné tyto statické stíny aplikovat i na **dynamické objekty**. V případě vozidla lze přečíst intenzitu osvětlení z některých bodů na silnici pod vozidlem. Intenzitu osvětlení lze jednoduše ukládat do alfa kanálu textury, ve které se nachází aktuální scéna.

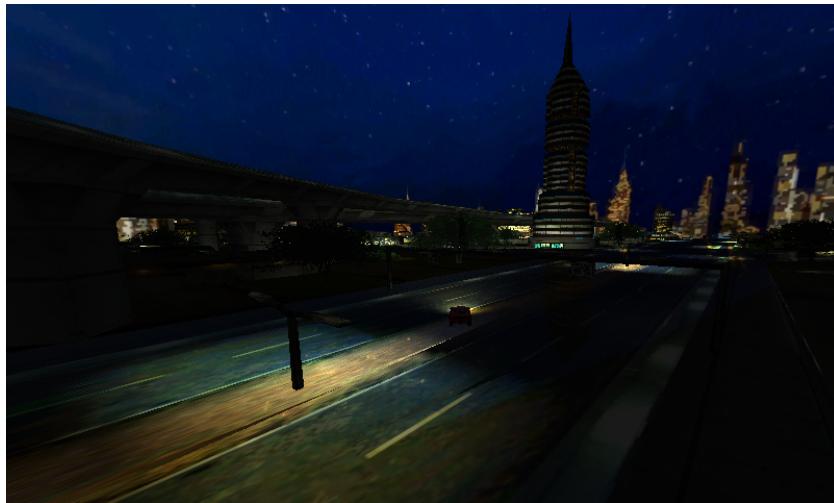
Zde vzniká problém s barevným světlem. Jsou zde dvě možnosti řešení. Prvním je přidání další textury, ve kterém by byla barevná intenzita světla (tím by došlo ke zvýšení náročnosti na hardware a ke zvýšení paměťové náročnosti). Druhým možným řešením je omezit barevná světla do nějaké sytosti (pokud se slabě zabarvené světlo aplikuje jako bílé, není to totíž rušivé).

4.3 Předvypočítané osvětlení

Předvypočítané osvětlení umožňuje velmi rychle zobrazovat stínování a stíny na statických objektech. Problémem je, že je potřeba aplikovat tento efekt i na dynamických objektech. Úroveň osvětlení lze přečíst z nejbližšího statického objektu, tedy za předpokladu, že budeme mít tuto hodnotu někde k dispozici. Tuto hodnotu lze mít uloženou v alfa kanálu aktuálního snímku a jen jí přečíst před tím, než se vykreslí dynamické objekty.

Pro generování lightmap jsou k dispozici dva druhy světel: bodové světlo a reflektor. Tyto světla jsou definované přímo v 3D modelovacím software. Dosah světel je omezen na 300metrů (je to z důvodu chyb, které vznikají u osvětlení na velkou vzdálenost). Světla pochopitelně nepoužívají spekulární složku, protože tato složka je závislá na pohledu kamery a tudíž jí nelze předpočítat.

I maximální možné rozlišení lightmap pro danou scénu není dostatečné a stíny jsou kostičkovány. Z tohoto důvodu se používají dva druhy filtrování. Prvním je filtrování PCF, které se provádí přímo při generování lightmapy a druhým je lineární filtrování, které se aplikuje až při aplikování lightmapy. Tato kombinace filtrování značně zvýší výslednou kvalitu stínů.



Obrázek 4.4: Scéna s aplikovanými lightmapami lamp

Dalším problémem je rozsah intenzity osvětlení v lightmapě. Například pouliční lampa na obr.4.4 nedělá očekávané stíny od stojanu lampy. Tento stín zmizí, protože při připočtení druhého světla se úroveň osvětlení v daných bodech rovná maximální možné hodnotě(dochází zde k ořezu vyšších hodnot).

4.4 Světla vozidel

Dle předběžných měření jsem zjistil, že na mobilních zařízeních lze za běhu počítat jedno až dvě světla(bez řešení viditelnosti světla). U vozidla ovládané hráčem není potřeba řešit viditelnost světla, prakticky zde nedochází k situacím, že by světlo osvětlovalo neviditelné objekty(je to z důvodu těsné vazby světlo hráčova vozidla-kamera).

U ostatních vozidel je třeba osvětlení zjednodušit, aby to mobilní hardware zvládal. Zvolil jsem metodu vykreslování kuželů do scény. Při vykreslování je použita OpenGL funkce blending, která umožňuje kreslit objekty s částečnou průhledností. Dále je také zapnuté ořezávání odvrácených trojúhelníku. Střed kužele je vyplněn několikrát zevnitř s odvrácenou normálou. Z toho důvodu je osvětlení silnější pokud světlo svítí do kamery než od kamery.

Kapitola 5

Testování

Testování probíhalo na smartphonu, tabletu a laptopu, které jsou blíže popsány v tabulce 5.1. Zařízení byla během testování plně nabita, byla připojena ke zdroji napájení a byly vypnuty volby úspory energie. Veškeré testy byly prováděny v režimu „free ride“ a za stejných podmínek.

	Samsung Galaxy S3 mini (mobil)	Google Nexus 7 2012 (tablet)	Acer TravelMate P253-e (laptop)
Procesor	NovaThor U8500	ARM Cortex-A9	Intel 1005M
Počet jader	2	4+1*	2
Frekvence CPU	1.0GHz	1.3GHz	2.4GHz
Operační paměť	1GB	1GB	4GB
Grafická karta	ARM Mali-400 MP	Nvidia Tegra 3	Intel HD Graphics
Rozlišení displeje	800x480	1280x800	1366x768
Operační systém	Android 4.1	Android 4.4	Kubuntu 13.10 32bit

*tablet má extra procesor pro nižší spotřebu během nečinnosti

Tabulka 5.1: Parametry zařízení, na kterých jsem projekt testoval

V prvním testu zjišťuji, jak je vykreslování rychlé při různém počtu vykreslovaných fragmentů. Jedná se v podstatě o snížení rozlišení displeje k zvýšení výkonu.

	Samsung Galaxy S3 mini (mobil)	Google Nexus 7 2012 (tablet)	Acer TravelMate P253-e (laptop)
0.2x (poor)	27.3ms - 29.9ms	22.4ms - 28.4ms	10.1ms - 10.6ms
0.4x (low)	39.7ms - 42.7ms	24.3ms - 33.2ms	12.5ms - 13.0ms
0.6x (normal)	47.6ms - 59.9ms	42.1ms - 45.4ms	15.5ms - 15.9ms
0.8x (high)	76.9ms - 85.1ms	62.1ms - 64.4ms	21.4ms - 21.7ms
1.0x (ultra)	91.3ms - 97.9ms	80.7ms - 81.8ms	26.7ms - 27.0ms

Tabulka 5.2: Závislost rychlosti vykreslování scény na zařízení a poměru rozlišení oproti nativnímu(v závorce je uvedena konfigurace v nastavení simulátoru). Z každého měření je uvedená minimální a maximální hodnota

Z výsledku je patrně vidět výkonový rozdíl mezi laptopem proti tabletu a mobilu. Laptop je přibližně 3x až 4x rychlejší, což je menší rozdíl než jsem očekával.

Dále jsem testoval závislost rozlišení textury lightmapy a rychlosti vykreslování. Naměřené údaje jsou v tabulce 5.3.

	Samsung Galaxy S3 mini (mobil)	Google Nexus 7 2012 (tablet)	Acer TravelMate P253-e (laptop)
512x512	45.9ms - 49.7ms	42.5ms- 44.4ms	13.1ms - 15.6ms
1024x1024	46.9ms - 48.6ms	41.5ms - 45.4ms	13.1ms - 13.3ms
2048x2048	47.6ms - 59.9ms	42.1ms - 45.4ms	15.5ms - 15.9ms

Tabulka 5.3: Závislost rychlosti vykreslování scény na platformě a rozlišení textur lightmap při vizuálních detailech normal(poměr 0.6x ku nativnímu rozlišení). Z každého měření je uvedená minimální a maximální hodnota

Z měření vyplývá relativně malá závislost na rozlišení textury. Pokud pominu nepřesnost v měření, jedná se o rozdíl přibližně 2ms u všech zařízení.

V dalším testu jsem se zaměřil na paměťovou náročnost. Tento test bohužel nebylo možné provést na platformě Android, ale zřejmě bych došel ke stejnemu výsledku pro obě platformy.

	Paměť	Sdílená paměť
512x512	29780kB	35424kB
1024x1024	29776kB	56928kB
2048x2048	29776kB	159328kB

Tabulka 5.4: Závislost využití paměti na rozlišení textur lightmap. Testováno na Acer TravelMate P253-e (laptop)

Nakonec jsem otestoval kompatibilitu s dalšími zařízeními, v testu uspěl tablet Google Nexus 7 2013, Samsung Galaxy S4 a smartTV set-top-boxu eGreat U8. V případě set-top-boxu bylo nutné ještě implementovat ovládání hardwarovými tlačítky. Další zařízení nebyla testována. Uvedená zařízení byla testována pouze na kompatibilitu, měl jsem je zapůjčená pouze na několik desítek minut.

Kapitola 6

Závěr

Vzniklý simulátor sice není po grafické stránce schopen konkurovat komerčním simulátorům na platformě Android. Nicméně disponuje několika grafickými technikami, které mnoho komerčních simulátorů postrádá. Předzpracované osvětlení dodává scéně lepší vzhled a vedlo se dokonce částečně aplikovat předzpracované osvětlení i na dynamické objekty.

Zadání bylo splněno, pouze poblikávající lampy nebyly realizovány ideálně. Jsou realizovány pomocí střídání více lightmap, což je paměťově zbytečně náročné a není zde možnost nastavení více stavů světla. Práce je poměrně rozsáhlá a nestihl jsem zhasnající světla realizovat efektivněji.

Práci do budoucna hodlám rozšířit o generování lightmap pomocí vrhání paprsku. Výsledek provedený pomocí rasterizace je sice uspokojivý, ale pomocí vrhání paprsku by šly realizovat i plošné zdroje světla a tím by byl výsledný efekt daleko kvalitnější. Aby scéna nebyla příliš tmavá, bylo nutné nastavit poměrně velkou hodnotu pro ambientní složku osvětlovacího modelu.

Literatura

- [Kap11] Lauris Kaplinski. Reflections. <http://khayyam.kaplinski.com/2011/09/reflective-water-with-glsl-part-i.html>, 2011.
- [Mas11] Arnaud Masserann. Shadow mapping. <http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-16-shadow-mapping>, 2011.
- [Pfa12] James Pfaff. Realtime local reflections. <http://www.gamedev.net/blog/1323/entry-2254101-real-time-local-reflections/>, 2012.
- [Sco02] Jim Scott. Packing lightmaps. <http://www.blackpawn.com/texts/lightmaps/>, 2002.

Příloha A

Testování zaplnění stránky a odsazení odstavců

Tato příloha nebude součástí vaší práce. Slouží pouze jako příklad formátování textu.

Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili? Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili? Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili?

Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili? Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili? Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili?

Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili? Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili? Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili?

Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili? Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili? Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili? Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili? Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili?

Příloha B

Pokyny a návody k formátování textu práce

Tato příloha samozřejmě nebude součástí vaší práce. Slouží pouze jako příklad formátování textu.

Používat se dají všechny příkazy systému L^AT_EX. Existuje velké množství volně přístupné dokumentace, tutoriálů, příruček a dalších materiálů v elektronické podobě. Výchozím bodem, kromě Googlu, může být stránka CSTUG (Czech Tech Users Group) [?]. Tam najdete odkazy na další materiály. Většinou dostačující a přehledně organizovanou elektronikou dokumentaci najdete například na [?] nebo [?].

Existují i různé nadstavby nad systémy T_EX a L^AT_EX, které výrazně usnadní psaní textu zejména začátečníkům. Velmi rozšířený v Linuxovém prostředí je systém Kile.

B.1 Vkládání obrázků

Obrázky se umísťují do plovoucího prostředí **figure**. Každý obrázek by měl obsahovat **název** (`\caption`) a **návěstí** (`\label`). Použití příkazu pro vložení obrázku `\includegraphics` je podmíněno aktivací (načtením) balíku graphicx příkazem `\usepackage{graphicx}`.

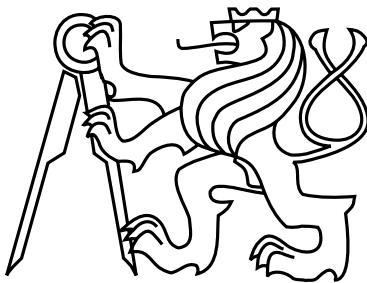
Budete-li zdrojový text zpracovávat pomocí programu `pdflatex`, očekávají se obrázky s příponou `*.pdf`¹, použijete-li k formátování `latex`, očekávají se obrázky s příponou `*.eps`².

Příklad vložení obrázku:

```
\begin{figure}[h]
\begin{center}
\includegraphics[width=5cm]{figures/LogoCVUT}
\caption{Popiska obrazku}
\label{fig:logo}
```

¹pdflatex umí také formáty PNG a JPG.

²Vzájemnou konverzi mezi snad všemi typy obrazku včetně změn vekostí a dalších výmožností vám může zajistit balík ImageMagic (<http://www.imagemagick.org/script/index.php>). Je dostupný pod Linuxem, Mac OS i MS Windows. Důležité jsou zejména příkazy `convert` a `identify`.



Obrázek B.1: Popiska obrázku

DTD	construction	elimination
	in1 A B a:sum A B in1 A B b:sum A B	case([_:A]a)([_:B]a)ab:A case([_:A]b)([_:B]b)ba:B
+	do_reg:A -> reg A	undo_reg:reg A -> A
*, ?	the same like and + with empty_el:empty	the same like and + with empty_el:empty
R(a,b)	make_R:A->B->R	a: R -> A b: R -> B

Tabulka B.1: Ukázka tabulky

```
\end{center}
\end{figure}
```

B.2 Kreslení obrázků

Zřejmě každý z vás má nějaký oblíbený nástroj pro tvorbu obrázků. Jde jen o to, abyste dokázali obrázek uložit v požadovaném formátu nebo jej do něj konvertovat (viz předchozí kapitola). Je zřejmě vhodné kreslit obrázky vektorově. Celkem oblíbený, na ovládání celkem jednoduchý a přitom dostatečně mocný je například program Inkscape.

Zde stojí za to upozornit na kreslící programe Ipe [?], který dokáže do obrázku vkládat komentáře přímo v latexovském formátu (vzorce, stejné fonty atd.). Podobné věci umí na Linuxové platformě nástroj Xfig.

Za pozornost ještě stojí schopnost editoru Ipe importovat obrázek (jpg nebo bitmap) a kreslit do něj latexovské popisky a komentáře. Výsledek pak umí exportovat přímo do pdf.

B.3 Tabulky

Existuje více způsobů, jak sázet tabulky. Například je možno použít prostředí `table`, které je velmi podobné prostředí `figure`.

Zdrojový text tabulky B.1 vypadá takto:

```
\begin{table}
\begin{center}
\begin{tabular}{|c|l|l|}
\hline
\textbf{DTD} & \textbf{construction} & \textbf{elimination} \\
\hline
\$mid\$ & \verb+in1|A|B a:sum A B+ & \verb+case([_:A]a)([ _:B]a)ab:A+\\
&\verb+in1|A|B b:sum A B+ & \verb+case([_:A]b)([ _:B]b)ba:B+\\
\hline
\$+\$&\verb+do_reg:A->reg A+&\verb+undo_reg:reg A->A+\\
\hline
\$*,\$ the same like \$mid\$ and \$+\$ & the same like \$mid\$ and \$+\$\\
& with \verb+empty_el:empty+ & with \verb+empty_el:empty+\\
\hline
R(a,b) & \verb+make_R:A->B->R+ & \verb+a: R->A+\\
&& \verb+b: R->B+\\
\hline
\end{tabular}
\end{center}
\caption{Ukázka tabulky}
\label{tab:tab1}
\end{table}
\begin{table}
```

B.4 Odkazy v textu

B.4.1 Odkazy na literaturu

Jsou realizovány příkazem `\cite{odkaz}`.

Pozor: Sazba názvů odkazů je dána BibTeX stylem

(`\bibliographystyle{abrv}`). BibTeX tedy obvykle vysází velké pouze počáteční písmeno z názvu zdroje, ostatní písmena zůstanou malá bez ohledu na to, jak je napíšete. Přesněji řečeno, styl může zvolit pro každý typ publikace jiné konverze. Pro časopisecké články třeba výše uvedené, jiné pro monografie (u nich často bývá naopak velikost písmen zachována).

Pokud chcete BibTeXu napovědět, která písmena nechat bez konverzí (viz `title = "\{\LaTeX\ --- online manuál"` v předchozím příkladu), je nutné příslušné písmeno (zde celé makro) uzavřít do složených závorek. Pro přehlednost je proto vhodné celé parametry uzavírat do uvozovek (`author = "..."`), nikoliv do složených závorek.

Odkazy na literaturu ve zdrojovém textu se pak zapisují:

Podívejte se na `\cite{Chen01}`,
další detaily najdete na `\cite{latexdocweb}`

Vazbu mezi soubory `*.tex` a `*.bib` zajistíte příkazem `\bibliography{}` v souboru `*.tex`. V našem případě tedy zdrojový dokument `thesis.tex` obsahuje příkaz `\bibliography{reference}`.

Zpracování zdrojového textu s odkazy se provede postupným voláním programů `pdflatex <soubor>` (případně `latex <soubor>`), `bibtex <soubor>` a opět `pdflatex <soubor>`.³

Níže uvedený příklad je převzat z dříve existujících pokynů studentům, kteří dělají svou diplomovou nebo bakalářskou práci v Grafické skupině.⁴ Zde se praví:

...

j) Seznam literatury a dalších použitých pramenů, odkazy na WWW stránky, ...
 Pozor na to, že na veškeré uvedené prameny se musíte v textu práce odkazovat -- [1].
 Pramen, na který neodkazujete, vypadá, že jste ho vlastně nepotřebovali a je uveden jen do počtu. Příklad citace knihy [1], článku v časopise [2], statí ve sborníku [3] a html odkazu [4]:
 [1] J. Žára, B. Beneš;, and P. Felkel.
 Moderní počítačová grafika. Computer Press s.r.o., Brno, 1 edition, 1998.
 (in Czech).
 [2] P. Slavík. Grammars and Rewriting Systems as Models for Graphical User Interfaces. Cognitive Systems, 4(4--3):381--399, 1997.
 [3] M. Haindl, Š. Kment, and P. Slavík. Virtual Information Systems.
 In WSCG'2000 -- Short communication papers, pages 22--27, Pilsen, 2000.
 University of West Bohemia.
 [4] Knihovna grafické skupiny katedry počítačů:
 <http://www.cgg.cvut.cz/Bib/library/>

... abychom výše citované odkazy skutečně našli v (automaticky generovaném) seznamu literatury tohoto textu, musíme je nyní alespoň jednou citovat: Kniha [?], článek v časopisu [?], příspěvek na konferenci [?], www odkaz [?].

B.4.2 Odkazy na obrázky, tabulky a kapitoly

- Označení místa v textu, na které chcete později čtenáře práce odkázat, se provede příkazem `\label{navesti}`. Lze použít v prostředích `figure` a `table`, ale též za názvem kapitoly nebo podkapitoly.
- Na návěstí se odkážeme příkazem `\ref{navesti}` nebo `\pageref{navesti}`.

B.5 Rovnice, centrovaná, číslovaná matematika

Jednoduchý matematický výraz zapsaný přímo do textu se vysází pomocí prostředí `math`, resp. zkrácený zápis pomocí uzavření textu rovnice mezi znaky `$`.

³První volání `pdflatex` vytvoří soubor s koncovkou `*.aux`, který je vstupem pro program `bibtex`, pak je potřeba znova zavolat program `pdflatex (latex)`, který tentokrát zpracuje soubory s příponami `.aux` a `.tex`. Informaci o případných nevyřešených odkazech (cross-reference) vidíte přímo při zpracovávání zdrojového souboru příkazem `pdflatex`. Program `pdflatex (latex)` lze volat vícekrát, pokud stále vidíte nevyřešené závislosti.

⁴Několikrát jsem byl upozorněn, že web s těmito pokyny byl zrušen, proto jej zde přímo necituji. Nicméně příklad sám o sobě dokumentuje obecně přijímaný konsensus ohledně citací v bakalářských a diplomových pracích na KP.

Kód `\$ S = \pi * r^2 $` bude vysázen takto: $S = \pi * r^2$.

Pokud chcete nečíslované rovnice, ale umístěné centrováně na samostatné řádky, pak lze použít prostředí `displaymath`, resp. zkrácený zápis pomocí uzavření textu rovnice mezi znaky `$$`. Zdrojový kód: `||$S = \pi * r^2 ||$|`` bude pak vysázen takto:

$$S = \pi * r^2$$

Chcete-li mít rovnice číslované, je třeba použít prostředí `eqation`. Kód:

```
\begin{equation}
S = \pi * r^2
\end{equation}

\begin{equation}
V = \pi * r^3
\end{equation}
```

je potom vysázen takto:

$$S = \pi * r^2 \tag{B.1}$$

$$V = \pi * r^3 \tag{B.2}$$

B.6 Kódy programu

Chceme-li vysázet například část zdrojového kódu programu (bez formátování), hodí se prostředí `verbatim`:

```
(* nickname2 *)
Lego> Refine in1
      (do_reg (nickname1 h));
Refine by in1 (do_reg (nickname1 h))
?4 : pcdata
?5 : pcdata
      (* surname2 *)
Lego> Refine surname1 h;
Refine by surname1 h
?5 : pcdata
      (* email2 *)
Lego> Refine undo_reg (email1 h);
Refine by undo_reg (email1 h)
*** QED ***
```

B.7 Další poznámky

B.7.1 České uvozovky

V souboru `k336_thesis_macros.tex` je příkaz `\uv{}` pro sázení českých uvozovek. „Text uzavřený do českých uvozovek.“

Příloha C

Seznam použitých zkratek

2D Two-Dimensional

ABN Abstract Boolean Networks

ASIC Application-Specific Integrated Circuit

⋮

Příloha D

UML diagramy

Tato příloha není povinná a zřejmě se neobjeví v každé práci. Máte-li ale větší množství podobných diagramů popisujících systém, není nutné všechny umísťovat do hlavního textu, zvláště pokud by to snižovalo jeho čitelnost.

Příloha E

Instalační a uživatelská příručka

Tato příloha velmi žádoucí zejména u softwarových implementačních prací.

Příloha F

Obsah přiloženého CD

Tato příloha je povinná pro každou práci. Každá práce musí totiž obsahovat přiložené CD. Viz dále.

Může vypadat například takto. Váš seznam samozřejmě bude odpovídat typu vaší práce. (viz [?]):

Na GNU/Linuxu si strukturu přiloženého CD můžete snadno vyrobit příkazem:

```
$ tree . >tree.txt
```

Ve vzniklém souboru pak stačí pouze doplnit komentáře.

Z **README.TXT** (případne index.html apod.) musí být rovněž zřejmé, jak programy instalovat, spouštět a jaké požadavky mají tyto programy na hardware.

Adresář **text** musí obsahovat soubor s vlastním textem práce v PDF nebo PS formátu, který bude později použit pro prezentaci diplomové práce na WWW.