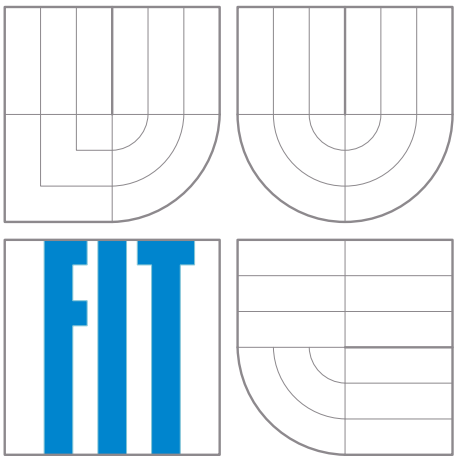

Projekt do předmětu GMU – Grafické a multimediální procesory

Raytracer pomocí GPU

31. prosince 2014

Řešitelé: Pavel Macenauer (xmacen02@stud.fit.vutbr.cz)
Jan Bureš (xbures19@stud.fit.vutbr.cz)
Fakulta Informačních Technologií
Vysoké Učení Technické v Brně

1 Zadání

- Implementace Raytraceru pomocí technologie CUDA v následujícím rozsahu:
  - Geometrická primitiva: roviny, koule
  - Phongův osvětlovací model
  - Bodové zdroje světla a stíny
  - Odlesky
- Akcelerace výpočtu scény
  - průzkum akcelerace raytracingu na GPU
  - subsampling
  - využití akcelerační struktury pro výpočet (BVH, KD-tree, octree, ...)
- Vygenerování demonstrační scény a její vykreslení

2 Použité technologie

Tady bych asi vepsal to jak jsme vyřešili tu paralelizaci a rozvržení paměti.

Zde vypište, jaké technologie vaše řešení používá – co potřebuje k běhu, co jste použili při tvorbě, atd. Text strukturujte, použijte odrážky, číslování. . .

Rozsah: cca 7 odrážek

3 Použité zdroje

Zde vypište, které zdroje jste použili k tvorbě: hotový kód, hotová data (obrázky, modely, . . .), studijní materiály. Pokud vyplyne, že v projektu je použit kód nebo data, která nejsou uvedena tady, jedná se o závažný problém a projekt bude pravděpodobně hodnocen 0 body.

Rozsah: potřebný počet odrážek

4 Nejdůležitější dosažené výsledky

Popište 3 věci, které jsou na vašem projektu nejlepší. Nejlépe ukažte a komentujte obrázky, v nejhorším případě vypište textově.

5 Ovládání vytvořeného programu

Stručně popište, jak se program ovládá (nejlépe odrážky rozdělené do kategorií). Pokud se ovládání odchyluje od zkratek a způsobů obvykle používaných v okénkových nadstavbách operačních systémů, zdůvodněte, proč se tak děje.

Rozsah: potřebný počet odrážek

6 Zvláštní použité znalosti

6.1 NVidia CUDA

Hlavní co jsme museli nastudovat jsou vědomosti ohledně technologie CUDA tedy struktura pamětí, jak je používat pomocí jejího C++ API a v kombinaci s programováním na CPU.

K debugování a programování jsme následně využívali vývojové prostředí Microsoft Visual Studio 2012 s NVidia NSight.

Museli jsme nastudovat i témata neprobíraná, resp. jen z lehka naťuklá na přenáškách. Jedním z nich je akcelerační struktura zvaná BVH.

6.2 Krátce o Bounding Volume Hierarchy

Jedná se o akcelerační strukturu při výpočtech v počítačové grafické.

Konstrukce

- Určí se konstanta pro počet objektů v nejnižší vrstvě stromu, tj. kolik listů bude mít každý předposlední uzel stromu.
- Řadíme objekty podle svých souřadnic (zda-li X, Y nebo Z záleží na programátorovi) a dělíme prostor vždy na 2.
- Skončíme ve chvíli, kdy v daném podprostoru je méně objektů, než-li námi daná konstanta, tyto objekty pak přidáme do seznamu daného uzlu

Výhody V případě raytracingu nemusíme testovat zda-li paprsek protne všechny primitiva ve scéně, ale vždy testujeme zda-li trefíme některý z podprostorů: kořen stromu má 2 podprostory, každý z nich zase další 2. Ve chvíli kdy narazíme na podprostor, který již nemá žádné další, ale pouze jednotlivé listy, tak provedeme test na průnik pouze těch primitiv obsažených pod daným uzlem. Počet testů je tak roven `HLOUBKA STROMU * 2 + POČET LISTŮ NA UZEL`, což bývá podstatně méně u velkých scén, než-li celkový počet primitiv.

7 Rozdělení práce v týmu

- **Jan Bureš**: Phongův osvětlovací model, úprava bilineární interpolace, především raytracing-části raytraceru, hloubka ostrosti, neostré stíny
- **Pavel Macenauer**: Základní kostra programu, optimalizace využitých paměťových jednotek, základ pro bilineární interpolaci, především CUDA-části raytraceru

8 Co bylo nejpracnější

- **Jan Bureš**: Na celé práci bylo asi nejložitější upravit algoritmus sledování paprsku tak, aby fungoval na architektuře CUDA. Dále pak správné rozdělení vláken do warpů a rozhodnout, který druh paměti použít pro které proměnné. Mno času bylo zapotřebí také věnovat rekurzi, kterou by sice CUDA měla podporovat od verze 3.0, ovšem stále program havaroval kvůli přetečení zásobníku.

- **Pavel Macenauer**: Nejprve celé zprovoznění CUDy a způsob, kterým předávat data mezi OpenGL, CUDou a následně je vykreslit stálo mnoho nervů a několik šálků kávy navíc. Především proto, že nové CUDA 3.0 API obsahuje nové metody pro komunikaci s OpenGL, nicméně dokumentace a materiály k tomu jsou především v podobě vygenerované dokumentace od společnosti NVidia. Dále ještě za zmínku stojí implementace BVH a celkové nastudování o principů.

9 Zkušenosti získané řešením projektu

Naučili jsme se více o architektuře CUDA a získali představu o psaní paralelních algoritmů, vyzkoušeli si implementovat phongův osvětlovací model a raytracer v praxi. Dále jsme si prostudovali možné algoritmické optimalizace raytraceru a některé z nich implementovali.

10 Autoevaluace

Technický návrh (85%): Tvorbu programu jsme si naplánovali na jednotlivé iterace, a tak až na pár úprav, které nás napadli během implementace nebylo nutné přepisovat již implementované části. Především se jednalo o části týkající se správy paměti, kdy s každým typem paměti na GPU se pracuje trochu jinak.

Programování (75%): Kód je dobře strukturovaný, ale mohl by být více okomentován, např. pro vygenerování použitelné doxygen dokumentace. Implementovaný raytracer je možné dále rozšiřovat a bez sebevětších komplikací přidávat další doplňky.

Vzhled vytvořeného řešení (80%): Scéna vypadá celkem pěkně, velmi zřídka lze pozorovat tečky, které narušují plynulost obrazu. Způsobené jsou pravděpodobně používáním datového typu float a nepřesností, které způsobuje. Některé optimalizace kvalitu obrazu mírně zhorší, ale vždy dle očekávání. Není problém do budoucna do implementovat i další primitiva, která by umožnila tvorbu komplexnějších scén.

Využití zdrojů (90%): Hodně jsme využili již implementovaný raytracer Aurelius k lepšímu pochopení raytracingu a trochu méně pak dostupnou literaturu. Zdroje o CUDě a jejím zapojení jsme získali především z vyhledávače Google.

Hospodaření s časem (70%): Začali jsme hned jak jsme obdrželi zadání, nicméně chvíli trvalo, než jsme vůbec měli něco co by něco vypočítalo na CUDě a předalo k zobrazení. Uprostřed semestru naše snaha mírně opadla kvůli jiným povinnostem ve škole. Na závěr jsme se snažili vše dokončit v čas, což se i povedlo.

Spolupráce v týmu (95%): Od začátku jsme komunikovali ohledně podmínek spolupráce. Následné programování pak probíhalo bez problémů a o všem jsem se navzájem informovali přes instantní mluvítka jako Skype nebo Facebook. Veškeré změny jsme pak evidovali v repositáři na serveru GitHub.com, kde jsme i vybudovali společné vývojové prostředí.

Celkový dojem (90%): Celý projekt se nám jevil mírně obtížnější vzhledem k ostatním projektům na škole, především z důvodu dostupnosti materiálů, kterých je na internetu spousty, nicméně vyfiltrovat z nich použitelné informace je časově náročné. Získané vědomosti a zkušenosti jsou určitě přínosem. Vybrali jsme si téma projektu kvůli architektuře CUDA, kterou jsme předtím v praxi nikdy nepoužívali a tímto jsem si udělali nejen lepší představu o samotné technologii ale i o moderních grafických kartách a naučili se pracovat s další technologií, která má své místo v softwarové budoucnosti. Celý projekt byl zajímavý také tím že jsme si vyzkoušeli vytvořit grafickou aplikaci úplně od začátku.

11 Doporučení pro budoucí zadávání projektů

Následující body jsou spíše nápady pro vytvoření možnosti v daném časovém rozsahu zpracovat komplexnější projekt a přeskočit onu fundamentální část projektu ”aby to něco dělalo”:

- Specifikovat požadavky k zadanému tématu (která primitiva implementovat, do jaké míry implementovat raytracer, případně vypsat, co změřit ...)
- Sestavit kostru projektu, např. projekt v MS Visual Studiu/QMake/Makefile s potřebnými hlavičkovými soubory obsahující prototypy metod

12 Různé

Ještě něco by v dokumentaci mělo být? Napište to sem! Podle potřeby i založte novou kapitolu.