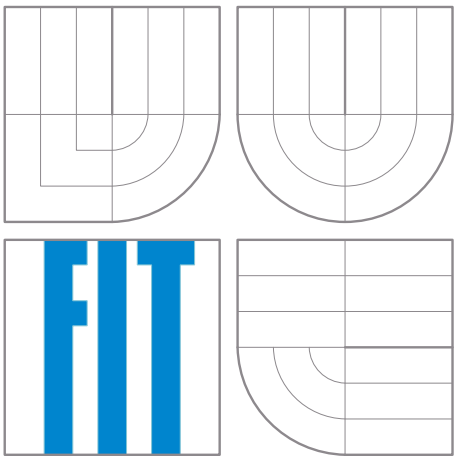


Projekt do předmětu GMU – Grafické a multimediální procesory

Nalezení minimálního orientovaného bounding boxu point cloudu pomocí GPU

21. prosince 2016

Řešitelé: Adam Jež (xjezad00@stud.fit.vutbr.cz)
Ivan Ševčík (xsevci50@stud.fit.vutbr.cz)
Fakulta Informačních Technologií
Vysoké Učení Technické v Brně

1 Zadání

Zvolené zadání projektu dávalo za úkol najít minimální orientovaný bounding box pomocí výpočtu na grafické kartě. Pro výpočet byli zvolené dva algoritmy, které jsou dále popsané v sekci 4 – přímočařejší Princip Component Analysis a pokročilejší exaktní metoda. První metoda byla díky nižší náročnosti na pochopení implementována celá samostatně bez použití jiných řešení, zatímco u druhé metody byla vybraná jenom pod část vhodná k paralelizaci a akceleraci, a zbytek algoritmu byl převzat. U obou metod se potom zkoumá zrychlení využitím grafického procesoru a taktéž se porovnává objem výsledných obalových těles.

2 Použité technologie

Při řešení jsme využili standardní knihovny `SDL2` a `GLEW` pro vytvoření grafického okna aplikace, ovládání a kontextu OpenGL. Dále jsme využili OpenCL pro akceleraci samotných výpočtů.

Potřebné knihovny byli pro bezproblémové spuštění přibalené do archívu. V případě potřeby je možné je stáhnout z následujících zdrojů:

- SDL – 2.0.4 – `https://www.libsdl.org/release/SDL2-devel-2.0.4-VC.zip`
- GLEW – 2.0.0 – `https://sourceforge.net/projects/glew/files/glew/2.0.0/glew-2.0.0-win32.zip/download`

Dále je potřeba mít nainstalované OpenCL SDK pro požadovanou platformu a nadefinované systémové proměnné obvyklým způsobem. Projekt byl vyvíjený ve Visual Studio 2015.

3 Použité zdroje

V rámci projektu byli vypracované 2 metody pro hledání minimálního orientovaného bounding boxu – pomocí Princip Component Analysis (PCA) [1] a na základě článku pro exaktní zjištění boudning boxu [2]. PCA metoda byla implementovaná samostatně na základě vzorce pro kovarianční matici, eigen vektorů a dalších běžně známých operací. Implementace exaktní metody byla převzata z knihovny MathGeoLib[3] a upravena pro naše potřeby. Původní licenční oznámení však byli i napříč rozsáhlým modifikacím v daných souborech ponechány kvůli lepší identifikaci převzatého kódu.

3D modely byli převzaté ze Stanfordské knižnice modelů[4] a následně upravené odstraněním nepotřebných topologických informací, čímž se zjednodušil formát a změnšila velikost dat.

Hlavním zdrojem při implementaci paralelního OpenCL kódu byl dokument NVIDIA OpenCL Programming Guide [5]. Při tvorbě prvotní OpenCL implementace byl použitý kód z cvičení předmětu GMU a teda výsledný kód obsahuje jeho fragmenty.

[1] `http://www.inf.fu-berlin.de/users/rote/Papers/pdf/On+the+bounding+boxes+obtained+by+principal+component+analysis.pdf`

[2] `http://clb.demon.fi/minobb/minobb.html`

[3] `https://github.com/juj/MathGeoLib`

[4] `https://graphics.stanford.edu/data/3Dscanrep/`

[5] `http://www.nvidia.com/content/cudazone/download/OpenCL/NVIDIA_OpenCL_ProgrammingGuide.pdf`

4 Použitý přístup

4.1 PCA

Metoda PCA má tyto hlavní pod části:

1. Výpočet centroidu a centrování dat
2. Výpočet kovarianční matice
3. Výpočet eigen vektorů
4. Projekce vrcholů na eigen vektory a nalezení krajních vrcholů

Výpočet centroidu je přímočarý, postačuje individuálně sčítat souřadnice všech bodů a podělit počtem těchto bodů. Při implementaci redukčního algoritmu pro výpočet sumy/součtu byl zvolený postup, který zjistí dostupné množství multiprocesorů na GPU, z něho odvodí přibližný počet dostupných vláken a spustí kernel s optimálním počtem vláken tak, že nejprve se provede sériová redukce v každém vláknu a nakonec se provede paralelní redukce s využitím lokální paměti. Implementovaný je kernelem `points_sum`. Centrování dat probíhá v samostatném kernelu `center_points`, jelikož výpočet kovarianční matice i projekce vyžaduje mít data vycentrované.

Kovarianční matice pro trojrozměrný prostor je matice 3x3, která na diagonále obsahuje variance dat v jednotlivých osách a v ostatních prvcích kovarianci mezi těmito osami. Jelikož je symetrická, postačuje vypočítat jenom kovarianci nad diagonálou, celkově se tedy jedná o 6 unikátních hodnot, které je nutné spočítat. V případě, že jsou data centrované, je jenom nutné vynásobit odpovídající souřadnice vrcholů, výsledky sčítat, a nakonec podělit počtem vrcholů. Násobení souřadnic a prvotní redukce pomocí lokální paměti se děje v kernelu `covariance_matrix`. Další redukce se taktéž řeší pomocí lokální paměti v samostatném kernelu `cov_reduce` a redukce se opakuje dokud výsledek není jenom 6 hledaných hodnot. Všech 6 hodnot se přitom počítá naráz v rámci jednoho spuštění kernelu.

Výpočet eigen vektorů není z pohledu paralelizace zajímavý, jedná se jenom o spočítání 9 hodnot (3 vektory v 3D). Výpočet nelze paralelizovat, ale i naproti tomu byl výpočet implementovaný jako kernel `compute_eigens`, aby se zabránilo zbytečným přenosům dat z GPU na CPU a zpět, jelikož režie spojená s takovýmto přenosem je velká a vyžadovala by synchronizaci.

Projekce je taktéž přímočará, jedná se jenom o projektování dat do prostoru, který je zadaný pomocí eigen vektorů, za účelem zjištění rozměrů bounding boxu. Jelikož při projekci vrcholu na eigen vektory je nutné znát všechny souřadnice bodu i všechny eigen vektory, jedno spuštění kernelu `projection_matrix` vypočítá všechny souřadnice projektovaného bodu. Vlákna spolupracují při načítání eigen vektorů do lokální paměti. Oboje šetří množství přístupů do globální paměti. Po projekci je nutné data opět agregovat, konkrétně najít minimum a maximum projektovaných bodů. Projekce a následná redukce, stejně jako nalezení středu a centrování bodů, používá nejprve sériový přístup a poté paralelní s využitím lokální paměti. Když po dané redukci zbude více výsledných bodů než je potřeba, je spuštěn kernel `reduction_minmax`, který pouze zredukuje výsledné hodnoty na požadovaný počet.

4.2 Exaktní metoda

V rámci exaktní metody byla na základě měření času CPU implementace zvolena k paralelizaci pod část výpočtu tzv. *sidepodal* hran se složitostí O^2. Sidepodal hrany k jiné hraně představují takové hrany, které můžou ve výsledném bounding boxu ležet na stěnách sousedích se stěnou dané hrany. Množina těchto hran je předpočítána před hledáním bounding boxu a právě tento výpočet byl

vhodný pro paralelizaci. Principem je vykonat pro každou dvojici hran test, jestli splňují podmínku, aby mohli být sidepodal. Samozřejmě stačí test vykonat jenom jednou pro každou dvojici.

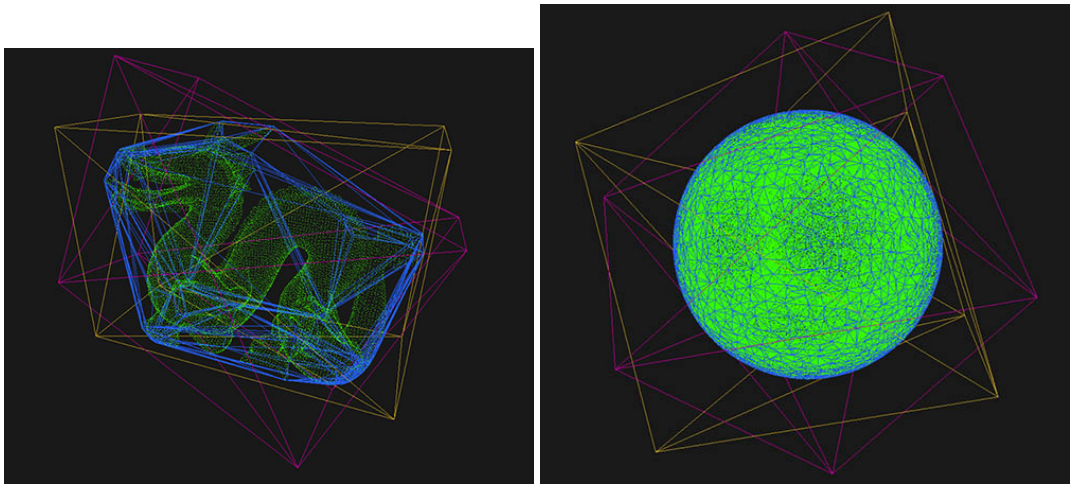
Výpočet, implementovaný kernelem `sidepodal`, probíhá ve 3 rozměrném prostoru, kde oba rozměry představují hrany konvexní obálky. Z toho plyne, že matice výpočtu je čtvercová. V paměti však nejsou uložené samotné hrany, ale normály stěn, které daná hrana spojuje. Pro každou hranu jsou tedy uložené dvě normály, které jsou při výpočtu použity. Všechny výpočty pod diagonálou je možné ignorovat, jelikož se jedná o stejný výpočet jen s obrácenými indexy hran oproti výpočtu nad diagonálou. Samotnou diagonálu je nutné počítat, jelikož hrana může patřit do své sidepodal množiny.

Prvně se tedy ukončí všechny výpočty, kdy celá skupina leží pod diagonálou. Následně se do lokální paměti načtou potřebné normály (celkově se jedná o $[4 * \text{šířka skupiny}]$ normál, 2 pro každou hranu a tyto seznamy jsou dva – jeden v každém rozměru). Následuje synchronizace a ukončení vláken, které patří do pracující skupiny, ale samy nic nepočítají (buď taky leží pod diagonálou a nebo za okrajem vstupních dat). Následně se už z lokální paměti načítají normály potřebné k výpočtu, vykoná se test a výsledek se uloží do paměti. Z paměti se už jenom přečte na straně CPU a po transformaci na požadovaný tvar se použijí výsledné hodnoty v původním exaktním algoritmu.

5 Nejdůležitější dosažené výsledky

Použití a porovnání 2 různých metod pro výpočet orientovaného bounding boxu, dosáhnutí zrychlení, přehledná vizualizace.

Obrázek 1 zobrazuje způsob, jakým je scéna vizualizovaná. Mračno bodů je vykreslované zelenou barvou, konvexní obálka modrou a jednotlivé bounding boxy fialovou (PCA metoda) a oranžovou (exaktní metoda). Viditelnost obálky a boxů je možné přepínat mezi vyplněným a drátovám modelem.

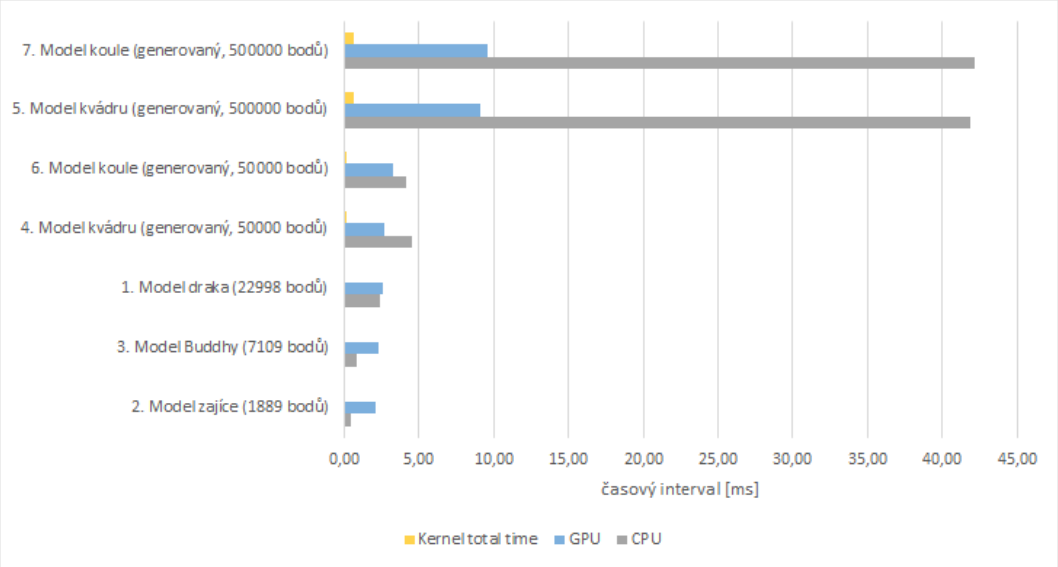


Obrázek 1: Vizualizácia scény a výsledkov

Všechny uskutečněné experimenty byly provedeny na počítači s následujícími specifikacemi:

- CPU – Intel(R) Core(TM) i5-4670K CPU @ 3.40GHz
- Operační paměť– DDR3 8 GB PC3-10700 (667 MHz)
- GPU – NVIDIA GeForce GTX 970 GDDR5 4 GB

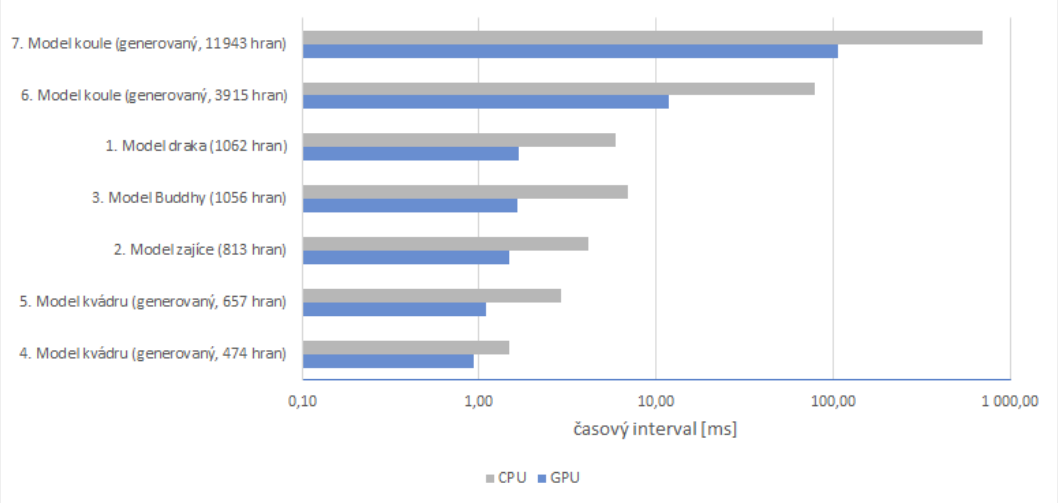
Na následujícím grafu 2 je zobrazenu porovnání mezi CPU a GPU implementací PCA algoritmu. Veškeré časové intervaly byly naměřeny 10krát a zprůměrovány. Pro porovnání jsou zde zobrazeny, kolik času trvaly pouze spuštěné kernely.



Obrázek 2: Porovnání CPU a GPU implementace metody PCA

Z grafu 2 lze vyčíst, že použití GPU implementace se vyplatí pouze při větším počtu bodů (přibližně od 30 tisíc). Zpracování jednotlivých kernelů trvá pouze zlomek času oproti spojené režii, mezi kterou patří: čtení a zapisování dat do paměti grafické karty, průchod ovladačem a transformace dat do požadované formy.

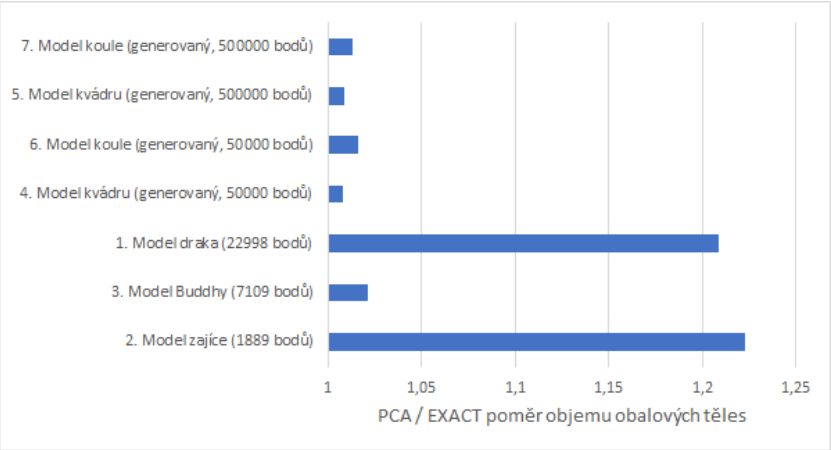
Na následujícím grafu 3 je porovnání částí pro výpočet sidepodal množin pro exaktní algoritmus.



Obrázek 3: Porovnání CPU a GPU implementace části algoritmu pro exaktní výpočet

V grafu lze vidět porovnání obou zmíněný algoritmů pro výpočet bounding boxu z hlediska objemu vygenerovaného kvádru. Podle očekávání vytváří exaktní metoda menší a tedy lepší bounding

boxy. Kvalita bounding boxu vygenerovaného pomocí metody PCA je velmi závislá na tvaru, který představuje dané mračno bodů.



Obrázek 4: Porovnání vygenerovaných bounding boxů pomocí obou zmíněných metod

6 Ovládání vytvořeného programu

Program sa ovláda pomocí grafického okna aplikace. Pokuď je toto okno v popředí, přijímá následující klávesové zkratky:

- **q, e** – zmenšit / zvětšit velikost vykreslovaných bodů.
- **w** – přepíná viditelnost orientovaných bounding boxů.
- **h** – přepíná viditelnost konvexní obálky.
- **1-7** (vrchní řada klávesnice) – přepíná modely:
    1. Model draka (22998 bodů)
    2. Model zajíce (1889 bodů)
    3. Model Buddhy (7109 bodů)
    4. Model kvádru (generovaný, 50000 bodů)
    5. Model kvádru (generovaný, 500000 bodů)
    6. Model koule (generovaný, 50000 bodů)
    7. Model koule (generovaný, 500000 bodů)
- **esc** – ukončení aplikace.

Dále je možné pomocí myší rotovat kameru kolem modelu (při držení pravého tlačítka) a model přibližovat / oddalovat kolečkem.

7 Rozdělení práce v týmu

- Ivan: Vizualizace, přizpůsobení kódu pro exaktní výpočet, kovarianční a *sidepodal* kernel, odvození výpočtu *eigen* vektorů, získaní platforem, načítání modelů.
- Adam: Implementace PCA výpočtu na CPU, implementace kernelů potřebných pro PCA, napojení kernelů a bufferů pro PCA výpočet, měření časů výpočtů.

8 Co bylo nejpracnější

Správně pochopit a implementovat PCA algoritmus nejdříve na CPU a následně ho vhodným způsobem paralelizovat. Neustále uvažovat o zarovnání z důvodu coalescingu na architekturách s OpenCL 1.1. Další aspekty výpočtů – volit vhodné datové struktury, použití pokročilejších redukčních technik. Použít pouze potřebný kód z knihovny MathGeoLib, která měla množství závislostí, jeho následné pochopení a vymyšlení vhodného způsobu paralelizace jeho podčástí. Vizualizace scény s různými daty a kamerou byla taktéž netriviální. Ladění kernelů bez možnosti jednoduchého debugování, na které jsme zvyklí při programování na CPU. Vyzkoušených bylo množství debugovacích nástrojů od NVIDIA, Intel i AMD, napříč tomu se po mnohých pokusech nepodařilo žádný úspěšně použít a tak jediná možnost byla přenášet vhodné mezi-výpočty na CPU ke kontrole. Profilování a měření všech potřebných časových intervalů. Dalším z problémů, na které jsme narazili, bylo podivné chování OpenCL při použití *UserEvent*, kdy při uvolňování kontextu aplikace zamrzla. Řešení bylo použití pouze *Event*, které pro naše profilovací potřeby stačilo.

9 Zkušenosti získané řešením projektu

Jak postupovat při paralelizaci a akceleraci kódu na GPU, které výpočty se vyplatí akcelerovat a které ne, jaké jsou možnosti a limity jednotlivých výpočetních architektur. Detailnější obeznámení se s CUDA architekturou grafických karet NVIDIA. Seznámení se s metodami pro výpočet obalovacích těles. Procvičení si znalostí C++.

10 Autoevaluace

Technický návrh (80%): Problematika byla nejdříve zanalyzovaná, byli nalezené vhodné přístupy a ty byly implementovány. Při implementaci paralelních výpočtů byly brány ohledy na všechny známé možnosti a omezení. Celková snaha op optimální implementaci.

Programování (60%): Kód je vhodně členěný do tříd a funkcí, přehledný, na potřebných místech komentovaný a využívá vícero rozšíření z C++11 a pozdějších verzí. Chybí však sofistikovanější ošetření chybových stavů a okomentování funkcí v hlavičkových souborech (doxygen). Parametry spuštění kernelů cílí na grafiky NVIDIA, jelikož jiné nám při testování nebyly dostupné.

Vzhled vytvořeného řešení (80%): Vizualizace zobrazuje všechny relevantní data vhodnou formou. Barvy byli volené tak, aby byli navzájem kontrastní a dobře viditelné. Kamera umožňuje pohled na scénu pod různými úhly.

Využití zdrojů (70%): Aplikace využívá především teoretických znalostí vyplývajících z literatury. Výjimkou je knihovna MathGeoLib, ze které byli použity relevantní části, které byly dále modifikované.

Hospodaření s časem (60%): Aplikace vzhledem k povinnostem v ostatních předmětech vznikala převážně v poslední 3 týdnech výuky. Vzhledem na omezené časové možnostem se čas využil hlavně na návrh a implementaci podstatných částí řešení a upozadily se aspekty, které by mohli přispět k celkové ucelenosti aplikace.

Spolupráce v týmu (100%): Komunikace téměř vždy v reálném čase, společné řešení problému v chatu a nebo hovorem přes Skype, aktivní přistup k řešení a dělbě práce.

Celkový dojem (70%): Vzhledem k tomu, že se jednalo o projekt, který neměl být dále využitý (například u diplomové práce, v jiném předmětu) a řešení ve dvou lidech, byla jeho náročnost mírně vyšší než se při volbě zadání předpokládalo. Samotné zadání bylo však zajímavé a poskytlo dostatek prostoru pro vlastní realizaci nápadů. Dodatečné upřesnění zadání, které přišlo s konzultací, však mohlo být součástí zadání v IS. Řešení je užitečné, jelikož obalovací tělesa se často využívají jako optimalizace při kontrole kolizí a orientovaný bounding box umožňuje obalit množství těles mnohem menším tělesem než jeho jednodušší alternativa – osově zarovnaný bounding box (AABB). Získané znalosti bude možné uplatnit v budoucnu pro snadnější orientaci v problematice a při volbě vhodného způsobu optimalizací časově náročných výpočtů.

11 Doporučení pro budoucí zadávání projektů

V případě, že je známá vhodná metoda pro implementaci daného problému, mohla by být tato metoda uvedená k zadání pro představu, jak daný problém lze řešit. Při výběru z desítek možných zadání je totiž časově náročné ke každému hledat materiály, pochopit je a vyhodnotit, jestli je metoda vhodná pro akceleraci na grafických procesorech. K lepšímu hospodaření s časem by mohla pomoct dřívější registrace varianty zadání.