



Grafické a multimediální procesory (GMU)
Nalezení minimální obalové koule koulí pomocí
GPU

v Brně
30. prosince 2019

Václav Martinka (xmarti76)
Petr Buchal (xbucha02)

1 Zadání

Popis varianty:

- Implementujte metodu pro nalezení minimální obalové koule koulí pomocí GPU
- Srovnejte výsledek s jednoduchou CPU implementací
- Popište způsob akcelerace na GPU

Existuje několik metod řešení tohoto problému. Například *Ritterova metoda*, která sice nenajde nejmenší obalovou kouli, zato funguje velice rychle. Bohužel je obtížné ji implementovat na GPU, jelikož jednotlivé iteraci výpočetní smyčky jsou na sobě navzájem závislé a není možné je spustit paralelně.

My jsme se proto rozhodli pro primitivní, ale snadno škálovatelný výpočet hrubou silou. Na začátku určíme relevantní prostor, kde lze očekávat řešení - obdélník obalující všechny koule. Ten posléze rozdělíme na několik podprostorů (v případě CPU implementace 8, u GPU experimentálně i 512) a zjistíme poloměr obalové koule se středem v tomto podprostoru. Vybereme podprostor s nejmenší koulí a postup opakujeme, dokud se řešení stále zmenšuje.

Tento algoritmus lze velmi snadno paralelizovat. Nevýhodou je, že výsledky se liší v závislosti na počtu podprostorů (čím více, tím optimálnější řešení) kvůli uvážnutí v lokálním maximu.

2 Použité technologie

Použité technologie:

- openCL - pro paralelní výpočty na GPU
- python - tvorba podpůrných dat a vizualizace výsledků
- C++ - příprava dat pro výpočet a řízení výpočtu

Projekt je ve formě *řešení* pro Visual Studio.

3 Použité zdroje

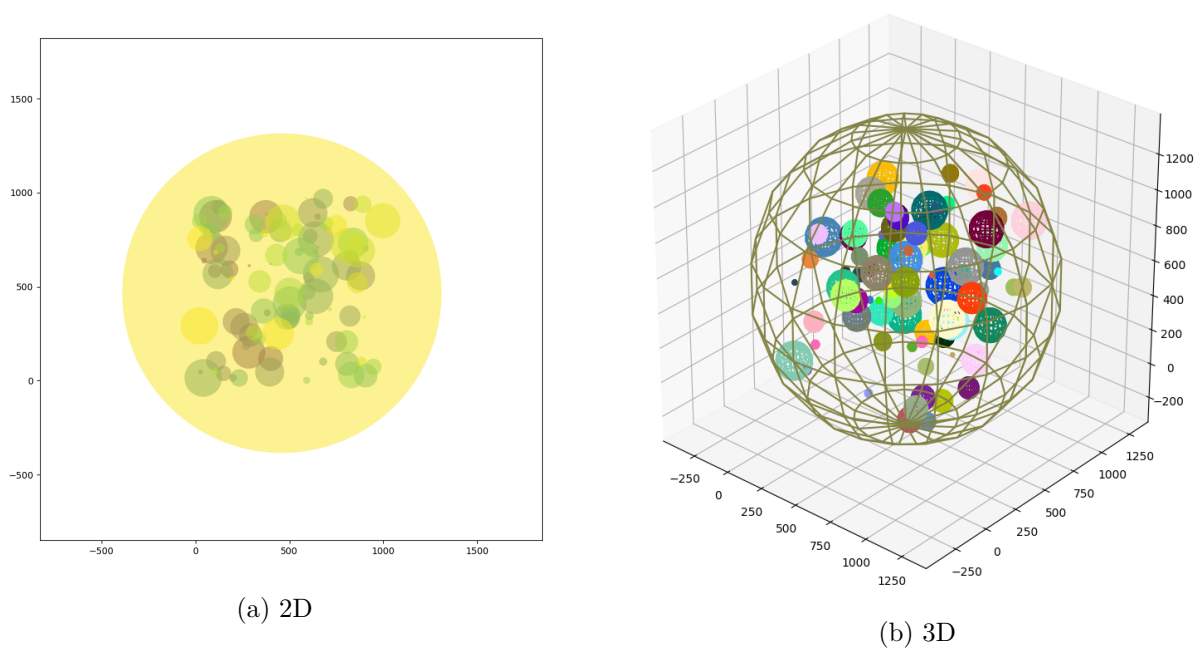
Jako kostru programu jsme využili soubory ze cvičení.

4 Způsob akcelerace

V případě CPU implementace jsou jednotlivé podprostory prohledávány postupně. Akcelerace bylo dosaženo paralelním prohledáním všech prostorů zároveň.

5 Způsob vizualizace

Vizualizační skript poskytuje 2 způsoby vizuální kontroly, zdali obalová koule obsahuje všechny požadované koule. Prvním způsob, který je nepostačující, je 2D pohled ze 3 stran prostoru (obrázek [1a](#)). Pokud by zde nějaká koule překračovala obalovou kouli, je zjevné, že algoritmus nepracuje správně. Druhý způsob vizualizace je pohled na 3D prostor (obrázek [1b](#)), kterým lze otáčet a tak zkontrolovat, že všechny požadované koule jsou uprostřed obalové koule.



Obrázek 1: Vizualizace minimální obalové koule koulí.

6 Ovládání vytvořeného programu

- Generování náhodných koulí obstarává skript `sphere_generator.py`. Povinný argument `-count` ovlivňuje počet generovaných koulí. Argumenty `-start` a `-end` ovlivňují rozsah, ve kterém se budou generovat středy koulí. Na závěr `-maximal_size` definuje maximální poloměr koule.
- Vizualizace výsledků je možná pomocí `-visualize.py`. Argumenty `-x`, `-y`, `-z` a `-radius` definují obalovou kouli. Dále je povinný argument se jménem vstupního souboru.
- Nejdůležitější částí projektu je program `GMU.exe`, který jako jediný argument přijímá název souboru s koulemi. Postará se o výpočet na CPU i GPU. Porovná a zobrazí výsledky včetně času potřebného pro výpočet.

7 Vyhodnocení

Algoritmus jsme testovali na notebookovém procesoru *i5* šesté generace a na jeho vestavěné grafické kartě. Na malém množství (desítky koulí) dat je CPU implementace výrazně rychlejší (přibližně o dva řády). S rostoucím množstvím dat (miliony koulí) se rozdíly stírají, byť CPU implementace je stále rychlejší.

Důvodem je nízká časová náročnost samotného výpočtu. Proto je algoritmus brzděn přesunem dat před a po výpočtu. Bylo by proto nutné přesunout přípravu dat a zpracování výsledku z CPU na GPU.

8 Rozdělení práce v týmu

- *Václav Martinka* - návrh výpočetního algoritmu, optimalizace základní implementace
- *Petr Buchal* - generující a vizualizační skript, základní implementace algoritmu