Projekt do předmětu GMU – Grafické a multimediální procesory

Simulace vodních ploch na GPU

30. prosince 2017

Řešitelé: Roman Čižmarik (xcizma04@stud.fit.vutbr.cz)
Tomáš Mlynarič (xmlyna06@stud.fit.vutbr.cz)
Fakulta Informačních Technologií
Vysoké Učení Technické v Brně

1 Zadání

Obecným zadáním tohoto projektu bylo vytvoření simulátoru vodních ploch, který by byl akcelerovaný na GPU. Po prostudování problému, bylo zjištěno, že je několik možných způsobů, jak docílit výsledného simulátoru, zejména simulace pomocí dynamiky tekutin (Eulerův přístup), nebo simulace pomocí částicového systému (Lagrangeův přístup). Pro tento projekt byl zvolen Lagrangeův přístup, neboli simulace pomocí částicového systému. Konkrétně byla zvolena metoda *Smoothed Particle Hydrodynamics*.

2 Použité technologie

- CMake
- Qt (Qt3D)
- OpenCL 1.2

Pro řešení jádra aplikace a vykreslování byl zvolen framework Qt, který umožňuje práci s GUI, ale také práci s 3D vykreslováním. Toto se ze začátku jevilo jako velký plus (díky jednoduchému zacházení s entitami), nicméně v pozdějších fázích bylo zjištěno, že pro framework (pro práci s 3D) není pořádná dokumentace, ani mnoho návodů.

Pro implementaci akcelerovaných výpočtů byl zvolen framework OpenCL, konkrétně verze 1.2 a to z důvodu nepodporování novější verze na systémech macOS.

3 Použité zdroje

Projekt byl z velké části inspirován prací [4], která obsahuje teorii k simulaci *Smoothed Particle Hydrodynamics (SPH)*, obsahuje také pseudokódy k potřebným výpočtům a dokonce i teorii k paralelizaci výpočtů. Dalším zdrojem pro pochopení *SPH* bylo [7], ve kterém jsou popsány principy i uniformní mřížka a kolize. Části kódu byly použity z cvičení GMU, ale také z [6], odkud byla převzatá myšlenka pro wraper na OpenCL, inspirace pro pochopení rovnic pro simulaci byla použita z [5]. OpenCL návod a informace byly brány z [9]. Pro implementaci integračních rovnic byl použit zdroj [2]. Pro implementaci prefixové sumy byly použity zdroje [1] a [3].

4 Přístup

Jak již bylo zmíněno, pro výpočet simulace byla použita metoda *Smoothed Particle Hydrodynamics*. Metoda je vhodná k paralelním výpočtům, jelikož je možné každou částici vody spočítat samostatně, s příspěvkem pouze několika okolních částic. Algoritmus výpočtu byl převzat z [4]. Pro výpočet simulace bylo implementováno několik kernelů, které na sebe navazují při výpočtech.

4.1 Výpočet pozic v mřížce

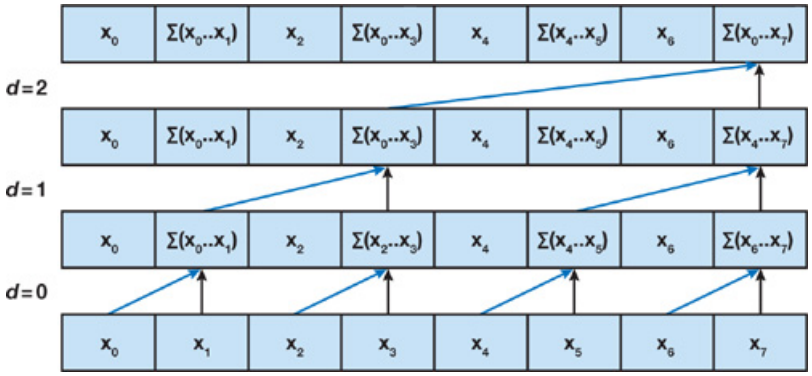
Uniformní mřížka (*uniform grid*) je jednou z nejjednodušších akceleračních struktur na vyhledávání v 3D prostoru. Prohledávaný prostor je rozdělen na buňky uniformní velikosti, zarovnaných podle os scény. Tato struktura neobsahuje žádnou hierarchii buněk.

V implementaci je samotná mřížka reprezentována 1D vektorem. Implementace na GPU se skládá z celkem tří kernelů. V prvním kroku každá částice atomicky inkrementuje počítadlo v mřížce na základě své pozice v prostoru a zároveň si poznamená do které buňky patří. Výsledkem tohoto kroku je počet částic v jednotlivých buňkách mřížky. V následujícím kroku jsou vypočteny prefixové sumy mřížky, k tomu se používá algoritmus *Blelloch scan* [1] [3], který je dobře paralelizovatelný. Prefixové suma jak ji definoval Blelloch je binární asociativní operátor $\oplus$ s identitou I, nad polem o n prvcích:

$$\text{Vstup: } [a_0, a_1, \ldots, a_{n-1}] \tag{1}$$

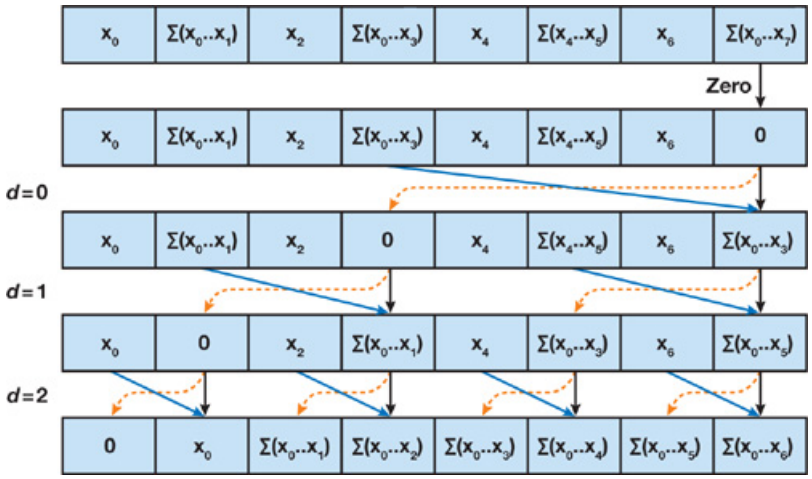
$$\text{Výstup: } [I, a_0, (a_0 \oplus a_1), \ldots, (a_0 \oplus a_1 \oplus \ldots \oplus a_{n-2})] \tag{2}$$

Základní algoritmus se skládá ze dvou fází: *reduce* (nebo *down-sweep*) a *up-sweep*. Tento algoritmus využívá algoritmický vzor často využíván v paralelním programování: vyvážený binární strom. V první fázi (*reduce*) je budován binární strom ze vstupu (listů), jak je možné vidět na obrázku 1. Na konci této fáze poslední prvek (kořen) stromu, obsahuje sumy všech prvků ve vstupním poli a ostatní prvky obsahují částečně sumy, případně jsou listy stromu.



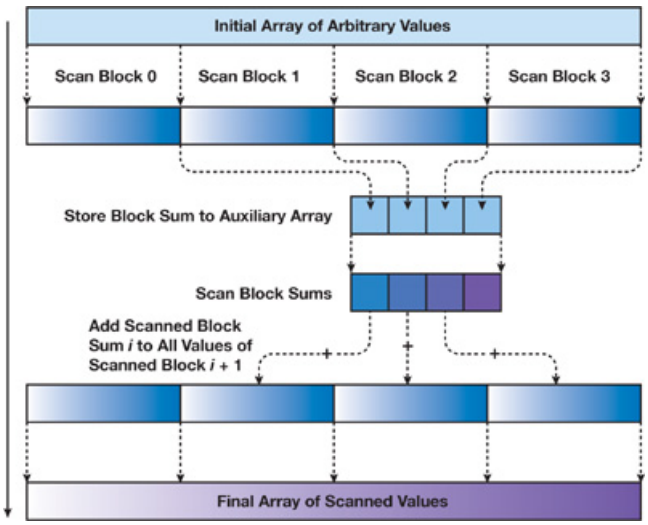
Obrázek 1: Fáze *reduce*

Ve druhé fázi (*down-sweep*), je strom procházen od kořene k listům a využívá částečné sumy z předchozího kroku. Na začátku této fáze je nutné vložit do kořene stromu hodnotu identity, v tomto případě 0. Následně v každém kroku, každý uzel v dané úrovni stromu, vypočítá sumu vlastní hodnoty a hodnoty levého potomka, zapíše tuto hodnotu do pravého potomka a svou hodnotu předá na levého potomka, jak je vidět na obrázku 2.



Obrázek 2: Fáze *down-sweep*

Tento základní algoritmus však není dostačující, protože je zřejmě, že velikost pracovní skupiny musí být rovná velikosti pole a velikost pole musí být mocnina 2. Vstupní pole proto musí být rozšířeno na nejbližší mocninu 2. Následně je vstupní pole rozděleno na bloky o velikosti pracovní skupiny. Na tyto bloky se použije základní algoritmus, který navíc může pracovat s lokální pamětí. Výsledná suma z první fáze algoritmu je zapsána do pomocného pole `sums` a pokračuje se fází druhou. Na pole `sums` se následně aplikuje stejný algoritmus. Výsledné sumy z tohoto kroku se přičtou ke všem prvkům odpovídajícího bloku vstupního pole, viz obrázek 3. Výhodou tohoto přístupu je využití lokální paměti. Z principu algoritmu stačí pouze polovina vláken velikosti vstupního pole. Limitací však zůstává velikost pole `sums`. To může byt maximálně tak velké jako je maximální velikost jedné pracovní skupiny, protože musí být zpracovány jednou skupinou.



Obrázek 3: Algoritmus *Blelloch scan* s využitím lokální paměti

Výsledkem tohoto kroku je pole prefixové sumy počtu částic v uniformní mřížce. Z mřížky je možné tedy snadno zjistit kolik částic je v dané buňce a kolik částic se nachází v předchozích buňkách.

V následujícím kroku je potřeba seřadit pole částic podle indexu buňky, do které patří. Na to je

využita funkce `sort` standardní knihovny, protože implementace řadících algoritmů na GPU není úplně triviální a nebyla cílem tohoto projektu. Po tomto kroku je připraveno vše, co je potřebn pro rychlé vyhledávání v uniformní 3D mřížce.

4.2 Výpočet hustoty a tlaku

Tento krok se počítá paralelně pro každou částici. Za použití připravené mřížky se pro každou částici najde okolí 3x3x3 sousedních buněk. Následně se vypočítá hustota pro danou částici z okolních částic pomocí vyhlazovacího jádra W_{poly6} [4]. Z vypočítané hustoty v daném místě a tuhosti simulované kapaliny, můžeme vypočítat tlak v tomto místě.

4.3 Výpočet sil

Podobně jako v předchozím kroku se nejprve vezme okolí dané částice a pro všechny tyto částice se zjistí podle vzdálenosti příspěvky sil. Pro výpočet sil se použijí vyhlazovací jádra (viz [4]) a spočítají se všechny příspěvky okolních částic. Tento výpočet probíhá paralelně pro všechny částice najednou pouze z globální pamětí.

4.4 Výpočet kolizí

I přes to, že prvotně v projektu byly připraveny algoritmy pro výpočet kolizí s objektem obecného typu, ve finální aplikaci je výpočet kolizí pouze s hranicemi scény, tedy stěnami kvádru. Pro jednoduchost výpočet probíhá pro všechny částice a všechny stěny kvádru. Nejprve se stěny kvádru zkopírují do lokální paměti a poté se pro každou částici vypočítá vzdálenost mezi bodem stěny a danou částicí a přičte se malé číslo. Pokud by se částice nacházela na druhé straně stěny, přičte se odraz od stěny k akceleraci částice.

4.5 Výpočet pozice a rychlosti

Poslední výpočet v daném iteraci je vypočtení nové pozice částice a její rychlost. Algoritmus počítá paralelně pro každou částici zvlášť a nejsou žádné závislosti mezi částicemi. Konkrétně byl aplikován Verletův algoritmus [2], který z parametrů vypočtených v předchozích krocích (pozice, rychlosti a akcelerace) vypočte nové.

4.6 Vykreslení scény

Po výpočtu nových pozic je potřeba scénu vyrenderovat. Toto je bohužel kvůli použití Qt3D děláno sekvenčně, jelikož každá částice je představena jako entita v grafu scény a tudíž není možné (nebyl nalezen způsob jak) pozici entity měnit přímo daty z grafické karty. Tento krok trvá při výpočtech nejdéle (viz sekce 5.1).

5 Nejdůležitější dosažené výsledky

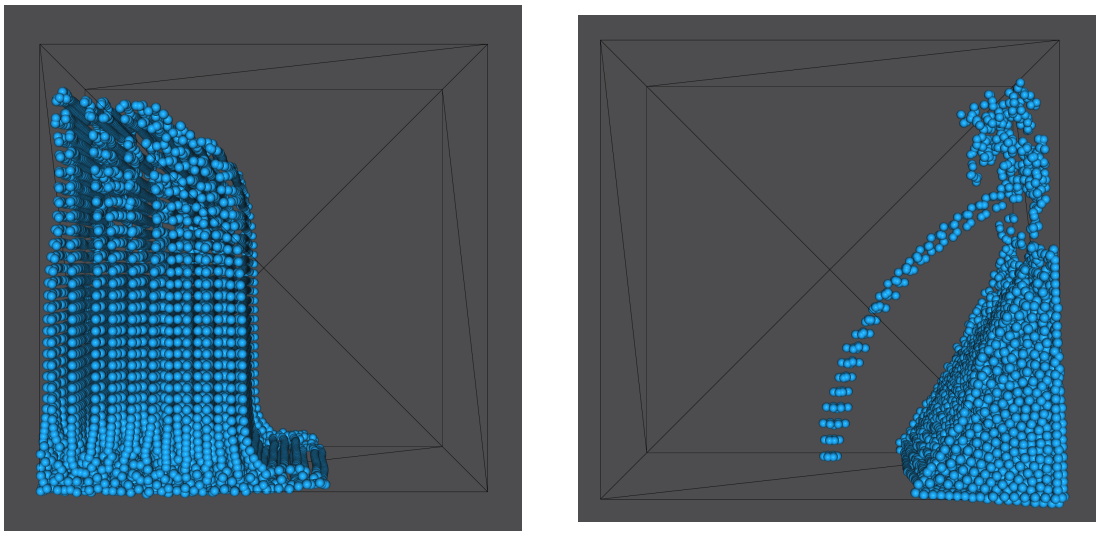
V projektu byla implementována simulace 2 různých scénářů:

- *Dam Break* – kvádr vody, který se rozteče (viz obrázek 4a)
- *Fountain* – vodotrysk ze spodu scény (viz obrázek 4b s upraveným vektorem gravitace)

Také byly naimplementovány 3 různé metody simulace:

- *CPU Grid* – na procesoru využívající akcelerační uniformní mřížky
- *GPU Brute Force* – OpenCL prohledávající všechny částice
- *GPU Grid* – OpenCL využívající uniformní mřížku

Profilování akcelerovaného kódu je na systémech macOS velmi problematické, jelikož neexistuje oficiální nástroj, který by toto nějakým způsobem umožňoval. Pro jednoduché profilování a zjištění problémů ohledně špatných zápisů do paměti či *data races* byl použit open source nástroj *Oclgrind*, který nad spuštěným programem vytvoří virtuální OpenCL stroj a umožňuje s ním práci (za cenu výkonu) [8].



(a) Dam Break

(b) Fountain (nakloněná gravitace)

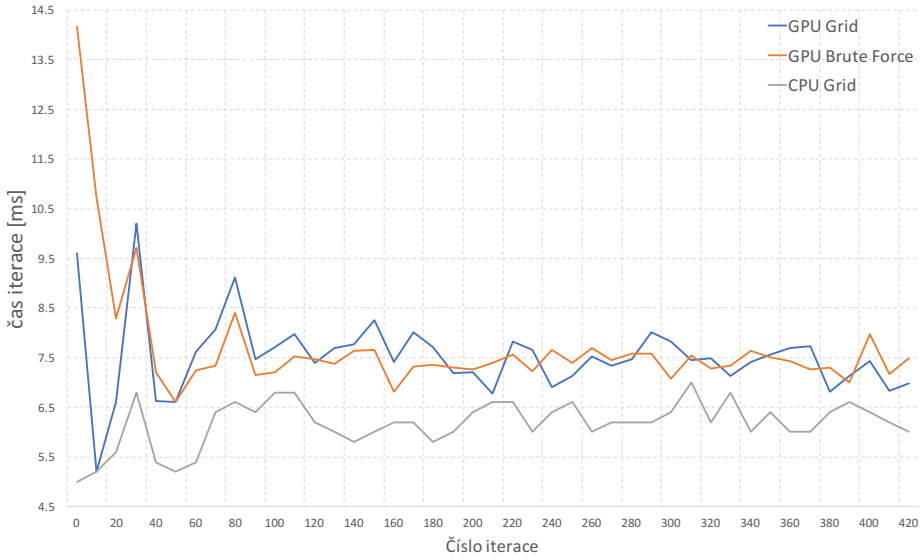
Obrázek 4: Scénáře pro 16000 částic

OS	macOS High Sierra 10.13.2
CPU	2.8 GHz Intel Core i7
GPU	AMD Radeon Pro 555
RAM	16 GB 2133 MHz LPDDR3

Tabulka 1: Specifikace počítače použitého k měření

5.1 Dosažené výsledky

Měření výkonu aplikace proběhlo na počítači se specifikacemi v tabulce 1. Jelikož se výsledky často lišily, bylo měření provedeno 5x a výsledky zprůměrovány tak, aby odchylka, způsobená jinými procesy v systému, byla co nejmenší. Měření proběhlo pomocí profilování `cl::Event` pro kód na GPU a pomocí `QElapsedTimer` pro CPU kód. Pro některé úseky kódu metody simulace *GPU Grid* byl použit kombinovaný způsob (jelikož řazení indexů částic bylo implementováno pomocí C++ funkce `sort`, viz sekce 4.1). Výsledné časy jsou v milisekundách.

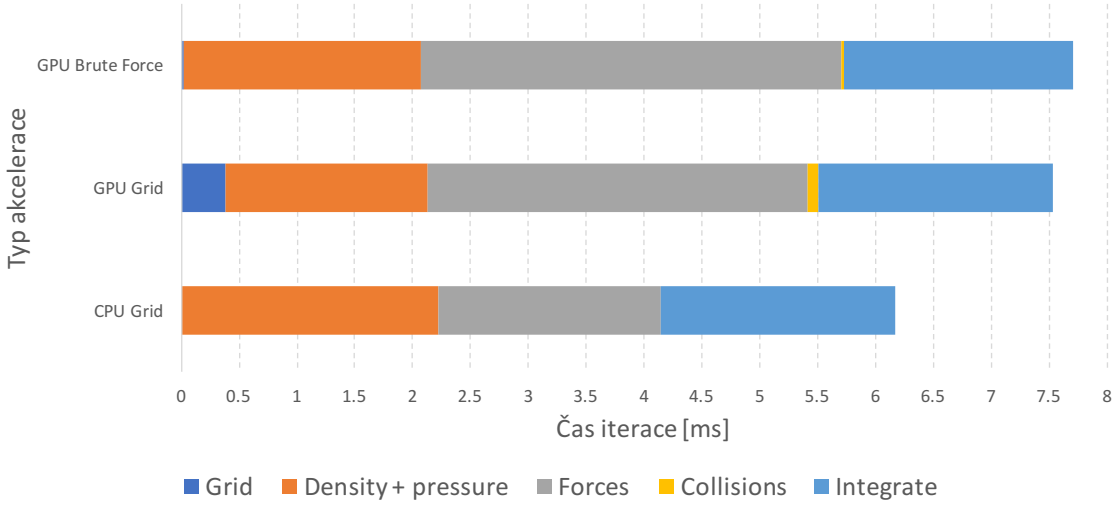


Obrázek 5: Průběh simulace (1620 částic, scénář *Dam break*)

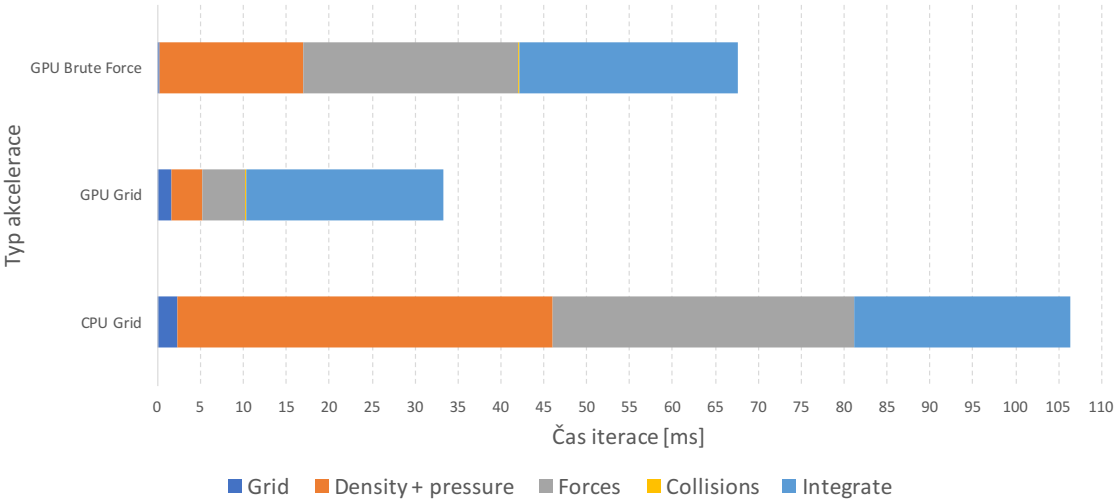
Na obrázku 5 lze vidět průběh simulace pro scénář *Dam Break* pro 1620 částic. V tomto případě je nejrychlejší simulace *CPU Grid*. Implementace využívající OpenCL jsou pomalejší kvůli větší režii přenosu dat než výkonu grafické karty.

Obrázek 6: Průběh simulace (16000 částic, scénář *Dam break*)

Obrázek 6 představuje stejný scénář pro 16000 částic. Zde je již možné vidět předpokládaný výsledek, a to zvýšení rychlosti pro OpenCL implementace. Velký nárůst výkonu je možné sledovat již při *GPU Brute Force* metodě, nicméně použití uniformní mřížky zrychlilo výpočet přibližně 4-násobně oproti CPU. Na obrázcích 7 a 8 je možné vidět zprůměrovaný průběh jedné iterace pro 1620 a 16000 částic. Jak bylo představeno na obrázku 5, metoda *CPU Grid* je pro malé množství částic nejrychlejší[1].



Obrázek 7: Rozbor iterace (1620 částic, scénář *Dam break*)



Obrázek 8: Rozbor iterace (16000 částic, scénář *Dam break*)

Problematická část jednoho kroku je výpočet nových pozic částic a následné překreslení scény. Ve všech metodách tento výpočet trvá přibližně stejnou dobu (na obrázku 8 znázorněno jasně modrou barvou). Důvod pro tento efekt je, že kvůli frameworku Qt3D je potřeba cyklicky projet všechny částice a aktualizovat jejich polohu ve scéně pomocí nově získaných pozic. Pro OpenCL implementace to znamená vykopírovat data z GPU a v cyklu aktualizovat všechny částice.

[1] pro *CPU Grid* není na grafu znázorněn čas pro *Collisions*, jelikož je součástí času *Forces*

6 Ovládání vytvořeného programu

Program se ovládá pomocí grafického okna, které je možné ovládat buď to myší, nebo klávesnicí. Před samotným spuštěním simulace je také možno vybrat hardware, scénář simulace, typ akcelerace a velikost mřížky, ve které se simulace odehrává.

6.1 GUI

Typ akcelerace

- CPU Grid (pomocí uniformní mřížky
- GPU Brute Force (každý s každým)
- GPU Grid (pomocí uniformní mřížky)

Typ scénáře

- **Dam break** – všechny částice v levé části scény
- **Fountain** – generování částic ze středu scény (á la fontána)

6.2 Myš

- **levé tlačítko** – pohled do scény
- **pravé tlačítko** – rotace kolem středu
- **kolečko** – zoom in/out

6.3 Klávesnice

- **g** – vypnutí gravitace
- **o** – posun gravitačního vektoru vlevo
- **p** – posun gravitačního vektoru vpravo
- **r** – vycentrování kamery
- **s** – krok simulace
- **space** – pauza/pokračování simulace

7 Rozdělení práce v týmu

- Roman: kostra projektu, rozdělení scény, wireframe, kolize, akcelerovaný grid, GUI
- Tomáš: základní algoritmus, kostra simulace, export logovacích dat, měření výsledků

8 Co bylo nejpracnější

- pochopeni algoritmu, CPU implementace
- pochopení a práce s Qt3D
- GPU uniformní mřížka
- multiplatformnnost (Windows + macOS | NVIDIA + AMD) a bohužel častá střelba do vlastních řad
- naměření vhodných výsledků

9 Zkušenosti získané řešením projektu

- vyzkoušení pro nás neznámého oboru – fyzikální simulace (vody)
- zkušenost s tvorbou částicových simulátorů
- zkušenost s HW akcelerací na GPU
- procvičení C++ a Qt

10 Autoevaluace

Technický návrh (70%): Aplikace byla vytvářena s ohledem na obecnost a tímpádem je možné vytvářet různé scénáře, používat různé typy simulátorů a dokonce i různé operační systémy s různým HW. Zvolení frameworku Qt se zdalo být jako dobrá volba, nicméně zpomaluje vykreslování.

Programování (85%): Kód byl vhodně členěn do tříd a abstrahován, aby bylo možné měnit pouze části pro specifický problém. Některé části byly připraveny příliš obecně, což ve výsledku bylo zjednodušeno pro specifický problém. Kód by mohl být více okomentován.

Vzhled vytvořeného řešení (70%): Simulace zobrazuje všechny potřebné informace, GUI umožňuje manipulaci jak myší, tak pomocí klávesnice. Práce s kamerou je trochu složitější a mohla by být lepší. Vykreslení by mohlo být realistické místo pouze vykreslování kuliček.

Využití zdrojů (60%): Aplikace využívá jak teoretických znalostí z literatury, tak částečnou inspirací jinými projekty. Framework Qt3D poskytuje značnou část práce se scénou, vykreslování i GUI.

Hospodaření s časem (70%): Aplikace postupně vznikala v druhé části semestru (s ohledem na jiné projekty). Větší důraz mohl být kladen na optimalizaci na GPU, nicméně po změření rychlosti bylo zjištěno, že největší problém je vykreslování, které se děje sekvenčně.

Spolupráce v týmu (100%): Komunikace často v reálném čase (Skype, osobně). Používání verzovacího systému *git*. Často diskuse a rozdělení práce před každou iterací implementace.

Celkový dojem (70%): Projekt byl z našeho hlediska docela pracný, jelikož jsme neměli zkušenost jak s programováním na GPU, tak s 3D simulátory částicových systémů. Navíc komplikace byla cílení na 2 systémy zároveň (*Windows* a *macOS*), jak i různé výrobce GPU (*NVIDIA* a *AMD*).

Zadání bylo zvoleno díky teoretické představě, později prodiskutováno a upřesněno v rámci konzultace.

Reference

[1] Guy E Blelloch. Scans as primitive parallel operations. *IEEE Transactions on computers*, 38 (11):1526–1538, 1989.

[2] Jonathan Dummer. A Simple Time-Corrected Verlet Integration Method [online], [cit. 2017-12-28]. URL http://lonesock.net/article/verlet.html.

[3] Mark Harris, John D. Owens, and Shubhabrata Sengupta. GPU Gems 3: Parallel Prefix Sum (Scan) with CUDA [online], [cit. 2017-12-28]. URL https://developer.nvidia.com/gpugems/GPUGems3/gpugems3_ch39.html.

[4] Zsolt Horváth. Částicové simulace v reálném čase. Master's thesis, Vysoké učení technické v Brně, Fakulta informačních technologií, 2012. URL http://www.fit.vutbr.cz/study/DP/DP.php?id=14098.

[5] Saeed Mahani. SPH 3D Real-Time Fluid Simulation [online], [cit. 2017-12-28]. URL https://github.com/saeedmahani/SPH-3D-Fluid-Simulation.

[6] Sai Narayan Natarajan. SmoothedParticleHydrodynamics [online], [cit. 2017-12-28]. URL https://github.com/Nightmask3/SmoothedParticleHydrodynamics.

[7] Yalmar Ponce and Claudio Esperança. Particle Based Simulations Using GPUs. URL http://www.ucsp.edu.pe/sibgrapi2013/eproceedings/tutorials/T7-handouts.pdf.

[8] James Price and Simon McIntosh-Smith. An opencl device simulator and debugger, [cit. 2017-12-28]. URL https://github.com/jrprice/Oclgrind.

[9] McIntosh-Smith Simon and Deakin Tom. Hands On OpenCL. 2014. URL https://github.com/HandsOnOpenCL/Lecture-Slides/releases/download/v1.2/KITE-OpenCL-course.pptx.