UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

GPU implementácia SPH metódy simulácie toku kvapaliny

Diplomová práca

2014 Bc. Tomáš Vajda

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

GPU implementácia SPH metódy simulácie toku kvapaliny

Diplomová práca

Študijný program:	Aplikovaná informatika
Študijný odbor:	2511 Aplikovaná informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	prof. RNDr. Roman Ďurikovič, PhD.
Konzultant:	prof. RNDr. Roman Ďurikovič, PhD.

Bratislava, 2014 Bc. Tomáš Vajda

47282565

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Tomáš Vajda
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: 9.2.9. aplikovaná informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský

Názov: GPU implementácia SPH metódy simulácie toku kvapaliny

Cieľ: Máme implementované SPH knižnice, ktoré simulujú dynamiku kvapalín časticovým systémom. Úlohou je implementovať časť tejto knižnice použitím GPU (grafického procesora) ako paralelného procesora, čo urýchli naše algoritmy. Mnohé algoritmy sú jedinečné a často používané svetovými lídrami.

Vedúci: prof. RNDr. Roman Ďurikovič, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: doc. PhDr. Ján Rybár, PhD.

Dátum zadania: 26.10.2011

Dátum schválenia: 26.10.2011 prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Čestne vyhlasujem, že magisterskú prácu som vypracoval samostatne s použitím uvedenej literatúry a pod dohľadom môjho školiteľa.

.....

Bratislava, 2014 Tomáš Vajda

Poďakovanie

Ďakujem môjmu školiteľovi prof. RNDr. Romanovi Ďurikovičovi, PhD. za jeho veľkú ochotu, pomoc, cenné rady a za trpezlivosť počas písania tejto magisterskej práce. Taktiež by som chcel poďakovať mojim kolegom zo seminára YACGS za užitočné implementačné rady a poskytnutie kódu na simuláciu toku kvapaliny pomocou SPH metódy na CPU. Ďakujem ešte aj mojej rodine a priateľom za ich podporu a povzbudzovanie počas môjho štúdia.

Abstrakt

V tejto práci sa zaoberáme simuláciou toku kvapalín pomocou metódy SPH. Metóda SPH simuluje kvapalinu pomocou častíc a Navier-Stokesových rovníc. Táto metóda je výpočtovo náročná vzhľadom na počet častíc v systéme. Je možné ju však veľmi dobre paralelizovať, pretože sily pre každú časticu môžu byť vypočítavané nezávisle na výpočte síl pre ostatné častice. My túto metódu implementujeme na GPU, čo urýchľuje výpočet a umožňuje, aby metóda v reálnom čase prepočítavala pozície častíc a simulovala tak tok kvapaliny. Výsledkom práce je funkčná implementácia SPH metódy pomocou OpenCL, vďaka čomu môže byť simulácia vypočítavaná nielen na GPU, ale aj na iných zariadeniach podporujúcich OpenCL. Vo výsledkoch práce sa zaoberáme aj rýchlosťami simulácie pre rôzne počty častíc a porovnávame ich aj s SPH implementáciou na CPU.

Kľúčové slová: 3D simulácia kvapalín, smoothed particle hydrodynamics, SPH, GPU, OpenCL, častice

Abstract

In this thesis we deal with fluid simulation using SPH method. SPH method simulates fluid using particles and Navier-Stokes equations. This method is computationally expensive with respect to a number of particles in the system. But it is suitable for parallelization, because forces for each particle can be computed independently from other particles. We implement this method on GPU, which accelerates computation and makes it possible to compute particles positions in real-time and simulate fluid flow. Result of this work is implementation of SPH method in OpenCL which enables to run simulation not only on GPU, but on all devices which support OpenCL. In results of this work, we concern with simulation speed for various numbers of particles and compare it to SPH implementation on CPU.

Keywords: 3D fluid simulation, smoothed particle hydrodynamics, SPH, GPU, OpenCL, particles

Obsah

1 Úvod **1**

1.1 Ciele a motivácia 2

1.2 Štruktúra práce 3

2 Prehľad dostupných prístupov **4**

2.1 OpenCL 4

2.1.1 Architektúra OpenCL 6

2.1.2 Vykonávanie OpenCL programu 9

2.2 Smoothed Particle Hydrodynamics 10

2.2.1 Aproximácia kernelu 11

2.2.2 Aproximácia častice 11

2.2.3 Vyhladzovacie jadrové funkcie 12

2.2.4 SPH pre tok kvapalín 14

2.2.5 Aproximácia hustoty 15

2.2.6 Aproximácia tlakovej sily 17

2.2.7 Slabo stlačiteľné SPH 19

2.2.8 Prediktívno-korektívne nestlačiteľné SPH 20

2.2.9 Implicitne nestlačiteľné SPH 22

2.3 Hľadanie k-najbližších susedov 24

2.3.1 Z-Indexing a triedenie 24

2.3.2 Indexovanie pomocou smerníkov 27

2.3.3 Brute-force metóda 28

3 Návrh softvérového diela **29**

3.1 Cieľ práce 29

3.2 Technológie 30

3.3 Architektúra 30

3.4 Simulačná časť 31

3.4.1 Štruktúra tried 31

3.5 OpenCL framework 34

3.5.1 Štruktúra tried 35

3.5.2 Diagram tried 40

4 Implementácia **42**

4.1 Štruktúra projektu 42

4.2 Použitie OpenCL frameworku 43

4.3 Simulačná časť 46

4.3.1 SPH metódy 46

4.3.2 Hľadanie k-najbližších susedov 51

5 Výsledky **54**

6 Záver **59**

A Slovník **60**

B Manuál k programu **62**

C Popis obsahu CD **64**

Zoznam obrázkov

1.1 Vizualizácia povrchu SPH kvapaliny pomocou volumetrickej metódy sledovania lúča [FAW10]. 3

2.1 Platformová časť OpenCL. 6
2.2 Pamäťová časť OpenCL. 9
2.3 Mortonov rozklad dvojrozmernej matice 8x8. 25
2.4 Dvojrozmerná mriežka pomocou lineárneho indexovania [Gre10]. 26

3.1 Diagram tried simulačnej časti. 32
3.2 Diagram tried OpenCL frameworku. 41

4.1 Obrázky z aplikácie 1 46
4.2 Obrázky z aplikácie 2 52
4.3 Obrázky z aplikácie 3 53

5.1 Obrázky z aplikácie 4 58

Zoznam tabuliek

5.1 Scéna 01, 4096 častíc kvapaliny, 8030 častíc tuhých telies, GPU 55
5.2 Scéna 01, 4096 častíc kvapaliny, 8030 častíc tuhých telies, CPU 55
5.3 Scéna 02, 13200 častíc kvapaliny, 18544 častíc tuhých telies, GPU 55
5.4 Scéna 02, 13200 častíc kvapaliny, 18544 častíc tuhých telies, CPU 56
5.5 Scéna 03, 167500 častíc kvapaliny, 90130 častíc tuhých telies, GPU 56
5.6 Scéna 03, 167500 častíc kvapaliny, 90130 častíc tuhých telies, CPU 56
5.7 Scéna 03, 167500 častíc kvapaliny, 90130 častíc tuhých telies, GPU 57
5.8 CPU implementácia, 4096 častíc kvapaliny, 8030 častíc tuhých telies 58

Zoznam algoritmov

4.1 Pseudokód - použitie OpenCL frameworku 45
4.2 Pseudokód - inicializácia 48
4.3 Pseudokód - WCSPH 49
4.4 Pseudokód - PCISPH 50
4.5 Pseudokód - IISPH 51
4.6 Pseudokód - Hľadanie k-najbližších susedov 53

Kapitola 1

Úvod

Simulácia kvapalín má na poli počítačovej animácie významnú úlohu. Používa sa nielen vo filmoch, kde je dôležitejšia skôr kvalita a hodnovernosť simulácie, ale aj v aplikáciách fungujúcich v reálnom čase, ktoré sa zameriavajú skôr na rýchlosť simulácie na úkor jej kvality. Jedna z používaných metód na simuláciu kvapalín je SPH (Smoothed-particle hydrodynamics). Metóda SPH simuluje kvapalinu pomocou častíc. Pre každú časticu vypočítava sily vzhľadom na častice v jej okolí. SPH metóda je vhodná na simuláciu kvapalín v reálnom čase, čím je však počet častíc vyšší, tým je simulácia pomalšia. Metóda je však vhodná na paralelizáciu, kedy sa môžu paralelne vypočítavať sily pre jednotlivé častice. Je to možné vďaka tomu, že výpočty síl pre jednotlivé častice sú od seba navzájom nezávislé. Paralelná implementácia metódy urýchľuje jej výpočet a je možné v reálnom čase simulovať veľké počty častíc, čo sekvenčná implementácia na CPU neumožňuje robiť v reálnom čase.

1.1 Ciele a motivácia

Naším cieľom v tejto práci je implementovať a paralelizovať SPH metódu tak, aby bola simulácia vykonávaná na GPU a bolo možné v reálnom čase simulovať veľké počty častíc. Na paralelizáciu SPH používame knižnicu OpenCL, ktorá umožňuje písať kód, ktorý môže byť vykonávaný na rôznych heterogénnych zariadeniach, nielen GPU. V tejto práci sa ale zamieravame iba na GPU. Hlavnou motiváciou práce je urýchlenie simulácie veľkého počtu častíc pomocou paralelizácie na výkonných GPU zariadeniach, čo umožňuje dosahovať oproti sekvenčnej implementácii na CPU násobné zrýchlenie. Vo výsledkoch práce uvádzame porovnania rýchlostí simulácie pre rôzne zariadenia a počty častíc a aj porovnanie so sekvenčnou verziou.

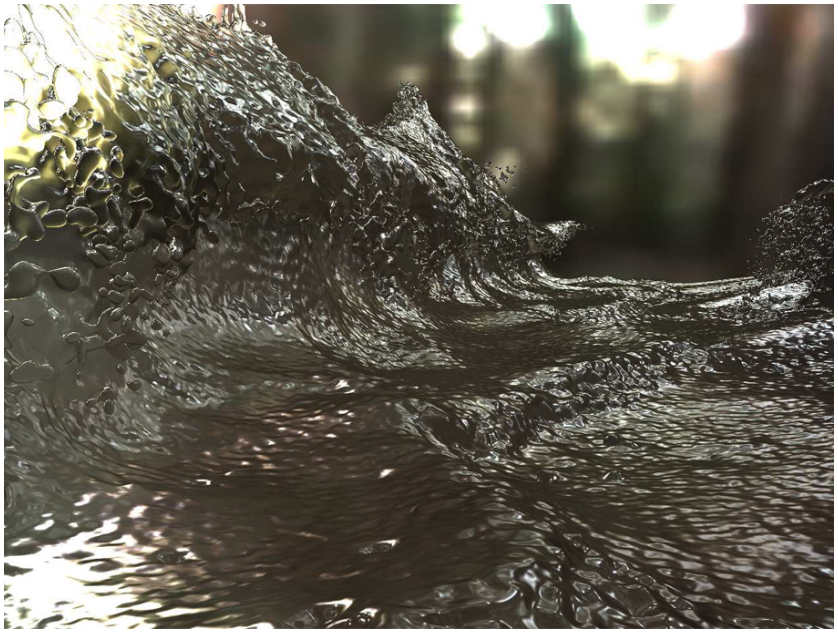
V práci implementujeme tri rôzne metódy SPH simulácie kvapalín pre čiastočne stlačiteľné a nestlačiteľné kvapaliny. Zamerali sme sa aj na použitie frameworku pre OpenCL knižnicu, ktorý nám umožňuje dostatočnú flexibilitu pri vytváraní kódu na simuláciu SPH. Framework je dostatočne flexibilný a dá sa využiť pri písaní ľubovoľného kódu, ktorý môže byť vykonávaný pomocou OpenCL na rôznych zariadeniach. V práci sa zameriavame aj na efektívne vyhľadávanie k-najbližších susedov, ktoré treba v OpenCL riešiť špecifickým spôsobom.

Z predchádzajúceho textu vyplývajú dva ciele, ktoré v tejto práci úspešne uskutočňujeme:

- Paralelizácia SPH metódy na simuláciu toku kvapalín
- Implementácia tejto metódy na GPU pomocou OpenCL knižnice

1.2 Štruktúra práce

Táto práca je rozdelená do šiestich kapitol. Kapitola **Prehľad dostupných prístupov** pojednáva o matematickom a fyzikálnom základe metódy SPH, zameriava sa aj na knižnicu OpenCL, popisuje metódu na vyhľadávanie k-najbližších susedov pre SPH simuláciu použitú v našej implementácii a uvádza aj prehľad iných metód. V ďalšej kapitole **Návrh softvérového diela** píšeme o architektúre našej implementácie. Kapitola **Implementácia** sa zaoberá technickými detailami implementácie, štruktúrou projektu a uvádza aj pseudokódy. V kapitole **Výsledky** meriame rýchlosti pre rôzne nastavenia simulácie a uvádzame aj porovnania. Nakoniec v kapitole **Záver** rekapitulujeme výsledky celej práce a píšeme aj o priestore pre možné vylepšenia našej implementácie.



Obr. 1.1: Vizualizácia povrchu SPH kvapaliny pomocou volumetrickej metódy sledovania lúča [FAW10].

Kapitola 2

Prehľad dostupných prístupov

V tejto kapitole stručne popisujeme knižnicu OpenCL, jej funkcionalitu a technické detaily. Ďalej sa zaoberáme metódou SPH, popisom metódy na hľadanie k-najbližších susedov použitej v našej práci a uvádzame aj iné metódy používané na hľadanie k-najbližších susedov.

2.1 OpenCL

OpenCL (Open Computing Language) je knižnica slúžiaca na vytváranie programov, ktorých kód môže byť vykonávaný na rôznych heterogénnych zariadeniach. Zameriava sa na superpočítače, osobné počítače, vložené systémy aj mobilné zariadenia. Umožňuje programovanie na rôznych výpočtových prostriedkoch ako CPU, GPU, DSP a FPG. Rovnaký kód môže byť vykonaný na CPU, GPU, DSP a FPGA, pričom OpenCL dynamicky sleduje záťaž systému a vyrovnáva ju medzi všetky dostupné procesory. OpenCL má nízkoúrovňovú flexibilitu, ktorá poskytuje rozsiahly prístup k výpočtovým prostriedkom pre vyššie úrovne abstrakcie a rôzne frameworky. Pozostáva z dvoch hlavných častí. Prvú časť tvorí programovací jazyk založený na C99

pre písanie kernel funkcií, ktoré sú potom vykonávané na OpenCL zariadeniach. Tento jazyk však má nové rozširujúce kľúčové slová a funkcie oproti C99 jazyku. Druhá časť zahŕňa API, ktorého úlohou je definovanie, spravovanie a ovládanie OpenCL zariadení a ktorá je vykonávaná na host zariadení. Stručne povedané OpenCL knižnica umožňuje písanie programov v jazyku C99 a potom ich následné spúšťanie pomocou API napríklad v jazyku C++.

Jednou z výhod OpenCL je prenositeľnosť kódu, takže kód napísaný v jeho jazyku môže byť vykonávaný na rôznych heterogénnych druhoch zariadení aj paralelných, ktoré majú viacero vykonávacích jednotiek aj takých, ktoré majú iba jednu vykonávaciu jednotku. Umožňuje vykonávať výpočty paralelne buď task-based alebo data-based paralelizmom. Je určený hlavne na paralelizáciu algoritmov, tak aby mohli byť vykonávané ďaleko rýchlejšie a efektívnejšie ako ich sekvenčné verzie. Pri algoritmoch, ktoré sú výpočtovo veľmi náročné je možné pomocou OpenCL na paralelných platformách dosiahnuť dramatické zrýchlenie oproti sekvenčným platformám.

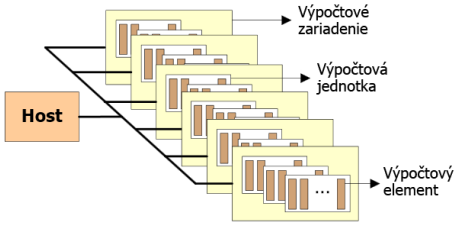
OpenCL sa skladá z množiny hlavičkových súborov a zdieľaných objektov, ktoré sú načítané počas run-time. Je to otvorený multiplatformový štandard, ktorého vývoj má na starosti Khronos Compute Working Group, pracovná skupina v rámci neziskového konzorcia Khronos Group, ktorého členmi sú napríklad AMD, Apple, Google, Intel, Motorola, Mozilla, NVIDIA, Samsung a ďalší z približne 100 členov.

OpenCL bolo pôvodne vyvíjané spoločnosťou Apple, ktorá ho v spolupráci so spoločnosťami AMD, Qualcomm, IBM, Intel a NVIDIA prepracovala do prvotného návrhu a predložila konzorciu Khronos Group. V roku 2008 bola sformovaná skupina Khronos Compute Working Group, ktorá pracovala pol roka na technickej špecifikácii pre prvú verziu OpenCL.

2.1.1 Architektúra OpenCL

OpenCL knižnica sa rozdeľuje na tri základné časti podľa ich logickej funkcionality. V ďalších podkapitolách postupne popíšeme jednotlivé časti, ich funkcionalitu a elementy, z ktorých sa tieto časti skladajú.

Platformová časť



Obr. 2.1: Platformová časť OpenCL.

Platformová časť OpenCL zastupuje výpočtové elementy, ktoré majú určitú funkcionalitu v rámci OpenCL. Slúži na pokrytie zariadení, ktoré môžu definovať, ktoré výpočty a funkcie majú byť spúšťané na ďalších zariadeniach. Na Obr. 2.1 je znázornená hierarchia jednotlivých elementov platformovej časti. Tieto elementy sú:

- *Host* je pripojený na jedno alebo viac výpočtových zariadení a môže na nich spúšťať vykonávanie programov. Host býva väčšinou CPU.
- *Výpočtové zariadenie* je to buď GPU, CPU, DSP alebo FPGA. Skladá sa z jednej alebo viacerých výpočtových jednotiek.
- *Výpočtová jednotka* pozostáva z jedného alebo viacerých výpočtových elementov.
- *Výpočtový element* vykonáva kód ako SIMD alebo SPMD.

Výpočtová časť

Výpočtovú časť tvoria všetky prvky, ktoré sa starajú o to, aby bol programový kód vykonávaný na OpenCL zariadeniach. V rámci tejto časti sa definujú kernel funkcie a programy, ktoré budú vykonávané na zariadeniach, definuje sa na akých zariadeniach majú byť vykonávané a kedy majú byť vykonávané. Umožňuje nastaviť ako majú byť jednotlivé zariadenia rozdelené na menšie celky, ktoré budú vykonávať kód. Výpočtovú časť tvoria tieto elementy:

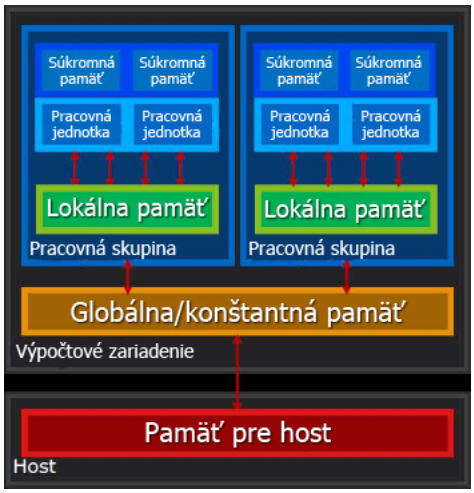
- *Kernel* je jediná funkcia, ktorá môže byť volaná z host zariadenia a bude vykonávaná na OpenCL zariadení. Kernel je kód napísaný v C99, ktorý je kompilovaný počas run-time.
- *Program* sa skladá z množiny kernel funkcií, funkcií a deklarácií. Pri vytváraní programu je potrebné špecifikovať, ktoré súbory tvoria program a potom ich skompilovať.
- *Command queue* je objekt do ktorého sú zaraďované príkazy, ktoré majú byť vykonané na OpenCL zariadení.
- *Pracovná jednotka* je najmenší výpočtový element. Po spustení kernel funkcie, začne jej kód vykonávať množstvo pracovných jednotiek (tento počet zadá programátor). Každá pracovná jednotka má identifikačné číslo prístupné z kernel funkcie, ktoré sa používa na rozlíšenie dát, ktoré majú byť touto jednotkou spracovávané.
- *Pracovná skupina* slúži na umožnenie komunikácie a spolupráce medzi pracovnými jednotkami. Vyjadruje ako sú pracovné jednotky organizované (jedná sa o n-dimenzionálnu mriežku pracovných skupín). Pra-

covná skupina má svoje unikátne identifikačné číslo, ktoré sa dá tiež získať z kernel funkcie.

Pamäťová časť

Pamäťová časť rozdeľuje a organizuje pamäť pre jednotlivé zariadenia na menšie celky. Pamäť je rozdelená na pamäť pre OpenCL zariadenie a pamäť pre host. Pamäť pre OpenCL zariadenie je rozdelená na menšie celky, pričom každý z nich má inú dobu prístupu k dátam. Manažment pamäte je explicitný, programátor teda musí v rámci OpenCL API sám posielať dáta z pamäti pre host do globálnej a lokálnej pamäti výpočtového zariadenia OpenCL a potom ich aj získať späť a zapísať do pamäti pre host. Na Obr. 2.2 sú graficky znázornené jednotlivé pamäťové celky a ich príslušnosť k jednotlivým výpočtovým jednotkám. Tieto pamäťové celky tvoria:

- *Súkromná pamäť* túto má každá pracovná jednotka svoju vlastnú. Je to pamäť s najrýchlejším prístupom a najmenšou kapacitou.
- *Lokálna pamäť*. Túto pamäť zdieľajú všetky pracovné jednotky v rámci pracovnej skupiny.
- *Globálna/konštantná pamäť* má najväčšiu kapacitu a zároveň je najpomalšia. Majú k nej prístup všetky pracovné skupiny avšak potrebuje synchronizačné primitíva. Je považovaná za streamovaciu pamäť pričom najväčší výkon je možné dosiahnuť používaním súvislých pamäťových adries alebo vzorov, ktoré využijú celú šírku pamäte.
- *Pamäť pre host* obvykle to je pamäť CPU teda operačná pamäť RAM.



Obr. 2.2: Pamäťová časť OpenCL.

2.1.2 Vykonávanie OpenCL programu

OpenCL umožňuje programátorom vykonávať kód na rôznych zariadeniach. Pre vykonanie tohto kódu je treba kód načítať, skompilovať a spustiť. Je to však proces, ktorý potrebuje ešte aj množstvo iných úkonov, ktoré je potrebné vykonať. Celý životný cyklus vykonávania programu rozdeľujeme na tieto jednotlivé kroky, ktoré musia byť urobené v nasledujúcom poradí, aby bol proces úspešný:

1. Získanie všetkých OpenCL zariadení na host zariadení.
2. Vytvorenie kontextu asociovaného s OpenCL zariadeniami.
3. Vytvorenie a skompilovanie programov, ktoré budú vykonané na jednom alebo viacerých OpenCL zariadeniach.
4. Výber kernel funkcií v programoch.
5. Vytvorenie pamäťových objektov prístupných z host a/alebo OpenCL zariadení.

6. Prekopírovanie pamäťových objektov na OpenCL zariadenia.

7. Zaradenie kernel funkcií do command queue.

8. Vykonanie kernel funkcií na OpenCL zariadeniach.

9. Prekopírovanie vypočítaných dát z pamätí OpenCL zariadení do pamäte pre host.

2.2 Smoothed Particle Hydrodynamics

V tejto časti sa zameriame na metódu Smoothed Particle Hydrodynamics a jej tri metódy, ktoré sme použili v našej implementácii. SPH metóda je široko používaná v počítačovej grafike kvôli svojej jednoduchosti a preto je vhodná aj pre potreby našej práce. Jedná sa o Lagrangeovu metódu na simuláciu toku kvapalín pomocou častíc. Pôvodne bola vyvinutá na simuláciu astrofyzického fenoménu v 3-dimenzionálnom vesmíre v [Gri77] a [Luc77]. Pretože pohyb simulovaného fenoménu je podobný toku kvapaliny a plynu, v simulácii môžu byť použité rovnice Newtonovskej hydrodynamiky.

Hydrodynamické problémy sú väčšinou popisované parciálnymi diferenciálnymi rovnicami premenných ako hustota, rýchlosť a energia. Až na jednoduché prípady, analytické riešenie nie je možné získať. Namiesto toho sa používajú numerické riešenia. V takomto prípade musí byť doména problému diskretizovaná a na parciálne diferenciálne rovnice musí byť použitá aproximácia vytvárajúca obyčajné diferenciálne rovnice v diskrétnej forme. V SPH je doména diskretizovaná množinou ľubovoľne distribuovaných častíc bez akejkoľvek spojitosti. Na aproximáciu parciálnych diferenciálnych rovníc je použitá metóda integrálnej reprezentácie. V SPH je táto metóda známa ako aproximácia kernelu. Po aproximácii kernelu je výsledok ďalej aproximovaný

pomocou častíc, čo je v SPH známe ako aproximácia častice.

2.2.1 Aproximácia kernelu

Prvý krok v aproximácii rovnice je takzvaná aproximácia kernelu. Počas tohto kroku je rovnica aproximovaná v integrálnej forme. Začneme s nasledujúcou identitou:

$$f(\mathbf{x}) = \int_{\Omega} f(\mathbf{x}')\delta(\mathbf{x} - \mathbf{x}')d\mathbf{x}', \tag{2.1}$$

kde $f \in \mathbb{R}^n \to \mathbb{R}^m$ je funkcia vektora pozície $\mathbf{x}$ a δ je Diracova delta funkcia. Ak je Diracova delta funkcia nahradená vyhladzovacou funkciou $W(\mathbf{x}-\mathbf{x}', h)$, dostaneme integrálnu reprezentáciu funkcie $f(\mathbf{x})$

$$f(\mathbf{x}) \doteq \int_{\Omega} f(\mathbf{x}')W(\mathbf{x} - \mathbf{x}', h)d\mathbf{x}', \tag{2.2}$$

Hodnota h vo vyhladzovacej funkcii W je vyhladzovací polomer definujúci oblasť vplyvu funkcie. Pokiaľ nie je funkcia W Diracovou delta funkciou, rovnica 2.1 je iba aproximáciou. V literatúre o SPH metóde je operátor aproximácie kernelu označovaný lomenými zátvorkami a rovnica 2.2 je prepísaná do tvaru:

$$\langle f(\mathbf{x})\rangle = \int_{\Omega} f(\mathbf{x}')W(\mathbf{x} - \mathbf{x}', h)d\mathbf{x}'. \tag{2.3}$$

2.2.2 Aproximácia častice

V SPH je celý systém reprezentovaný časticami, ktoré majú vlastnú hmotu a nachádzajú sa na rôznych miestach. Preto je aproximácia častice použitá na diskretizáciu integrálnej reprezentácie, ktorá je prevedená do sumácie. Sumácia je vykonávaná na všetkých časticiach v doméne nosiča vyhladzovacej funkcie. V aproximácii je objem $d\mathbf{x}'$ na pozícii častice j v rovnici 2.3 nahradený konečným objemom častice ΔV_j. Tento objem sa vzťahuje na hmotu

m_j častice j nasledujúcim spôsobom:

$$m_j = \Delta V_j \rho_j, \tag{2.4}$$

kde ρ_j je hustota častice j a $j \in 1, ..., N$ a N je počet častíc v doméne. Vďaka tejto výmene môže byť funkcia $f(\mathbf{x})$ z rovnice 2.3 prepísaná v diskretizovanej forme týmto spôsobom:

$$\begin{aligned} \langle f(\mathbf{x}) \rangle &= \int_\Omega f(\mathbf{x}') W(\mathbf{x} - \mathbf{x}', h) d\mathbf{x}' \\ &\cong \sum_{j=0}^{n} f(\mathbf{x}_j) W(\mathbf{x} - \mathbf{x}_j, h) \Delta V_j \\ &= \sum_{j=0}^{n} f(\mathbf{x}_j) W(\mathbf{x} - \mathbf{x}_j, h) \frac{1}{\rho_j} (\rho_j \Delta V_j) \\ &= \sum_{j=0}^{n} f(\mathbf{x}_j) W(\mathbf{x} - \mathbf{x}_j, h) \frac{1}{\rho_j} (m_j), \end{aligned} \tag{2.5}$$

takže dostaneme:

$$\langle f(\mathbf{x}) \rangle = \sum_{j=0}^{n} \frac{m_j}{\rho_j} f(\mathbf{x}_j) W(\mathbf{x} - \mathbf{x}_j, h). \tag{2.6}$$

Z predchádzajúcej rovnice je zrejmé, že pre aproximačný operátor SPH platí niekoľko pravidiel. Pre funkcie f_1, f_2 a konštantu c platí:

$$\begin{aligned} \langle f_1(\mathbf{x}) + f_2(\mathbf{x}) \rangle &= \langle f_1(\mathbf{x}) \rangle + \langle f_2(\mathbf{x}) \rangle \\ \langle f_1(\mathbf{x}) f_2(\mathbf{x}) \rangle &= \langle f_1(\mathbf{x}) \rangle \langle f_2(\mathbf{x}) \rangle \\ \langle c f_1(\mathbf{x}) \rangle &= c \langle f_1(\mathbf{x}) \rangle \\ \langle f_1(\mathbf{x}) + f_2(\mathbf{x}) \rangle &= \langle f_2(\mathbf{x}) \rangle + \langle f_1(\mathbf{x}) \rangle \\ \langle f_1(\mathbf{x}) f_2(\mathbf{x}) \rangle &= \langle f_2(\mathbf{x}) \rangle \langle f_1(\mathbf{x}) \rangle . \end{aligned} \tag{2.7}$$

2.2.3 Vyhladzovacie jadrové funkcie

Jedným z hlavných problémov metód, ktoré nie sú založené na trojuholníkových sieťach je ako urobiť aproximáciu funkcie založenej na ľubovoľne

rozptýlenej množine uzlov bez akejkoľvek spojitosti. V SPH to je zabezpečené použitím vyhladzovacej funkcie ako napríklad v rovnici 2.6. Táto funkcia W je v literatúre nazývaná *vyhladzovacia jadrová funkcia* alebo *jadrová funkcia*. Je to jedna z najdôležitejších častí SPH aproximácie, pretože nedefinuje iba dimenziu domény nosiča, ale určuje aj konzistentnosť a presnosť aproximácií.

V literatúre boli predstavené rôzne vyhladzovacie jadrové funkcie, ale každá z nich by mala spĺňať nasledujúce podmienky spomenuté v [LL03]:

Jednota. Vyhladzovacia jadrová funkcia musí byť normalizovaná nad svojou doménou nosiča - integrál domény nosiča sa musí rovnať 1:

$$\int_{\Omega} W(\mathbf{x} - \mathbf{x}', h) d\mathbf{x}' = 1. \tag{2.8}$$

Kompaktnosť nosiča domény. Vyhladzovacia jadrová funkcia by mala byť nenulová iba v malej doméne známej ako doména nosiča. Formálne:

$$W(\mathbf{x} - \mathbf{x}', h) = 0, \text{ pre } \|\mathbf{x} - \mathbf{x}'\| > kh, \tag{2.9}$$

kde h je vyhladzovacia dĺžka alebo vyhladzovací polomer a k je škálovací faktor. Táto vlastnosť transformuje aproximáciu z globálnej na lokálnu a zjednodušuje tak výpočet.

Kladnosť. Táto podmienka zaisťuje, aby vyhladzovacia jadrová funkcia mala kladné hodnoty v rámci celého nosiča domény. Táto podmienka nie je podmienkou pre konvergenciu a niektoré jadrové funkcie používané v SPH nespĺňajú túto podmienku. Napriek tomu je táto podmienka kľúčová pre hydrodynamiku, pretože zabezpečuje fyzikálny význam. Bez tejto podmienky môže aproximácia viesť k negatívnej hustote a energii.

Monotónnosť. Hodnota vyhladzovacej jadrovej funkcie by sa mala monotónne znižovať so zvyšujúcou sa vzdialenosťou od častice. Táto podmienka je založená na predpoklade, že bližšie častice by mali mať väčší vplyv ako tie, čo sú ďalej.

Vlastnosť delta funkcie. Vyhladzovacia jadrová funkcia by mala spĺňať Diracovu delta funkciu s polomerom nosiča približujúcim sa k nule:

$$\lim_{h\to 0} W(\mathbf{x}-\mathbf{x}',h) = \delta(\mathbf{x}-\mathbf{x}'). \tag{2.10}$$

Táto podmienka zaručuje, že $\langle f(\mathbf{x})\rangle = f(\mathbf{x})$ keď sa polomer nosiča približuje k nule.

Symetria. Vyhladzovacia jadrová funkcia by mala byť párna funkcia. Táto vlastnosť znamená, že častice s rovnakou vzdialenosťou, ale inými pozíciami by mali mať rovnaký vplyv. V niektorých metódach sa ale nedodržiava, aby bola zabezpečená vyššia konzistentnosť.

Rovnomernosť. Vyhladzovacia jadrová funkcia by mala byť dostatočne rovnomerná. Funkcia, ktorá spĺňa túto podmienku je menej citlivá na nerovnomernú distribúciu častíc a preto dáva zvyčajne lepšie výsledky.

Každá funkcia spĺňajúca predchádzajúce podmienky môže byť vyhladzovacia jadrová funkcia v SPH. Už bolo navrhnutých množstvo funkcií tak, aby boli použiteľné v SPH a líšia sa v mnohých aspektoch ako napríklad v kvalite výslednej aproximácie a výpočtovej náročnosti.

2.2.4 SPH pre tok kvapalín

Rovnica pre dynamiku kvapalín je založená na fyzikálnych zákonoch zachovania hmoty, hybnosti a energie, pričom my sa zaujímame o prvé dva. V tejto časti vysvetlíme ako riešiť dynamiku kvapalín podľa týchto dvoch zákonov. Rovnice popisujúce zachovanie hmoty a hybnosti môžu byť zapísané ako parciálne diferenciálne rovnice v Eulerovej forme. Tieto rovnice sú známe ako Navier-Stokesove rovnice. Pre izotermickú kvapalinu sú Navier-Stokesove rovnice vyjadrené nasledovne:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0, \tag{2.11}$$

$$\rho(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v}) = -\nabla p + \mu \nabla^2 \mathbf{v} + \mathbf{f}, \tag{2.12}$$

kde $\mathbf{v}$ je rýchlosť, ρ je hustota, p je tlak, μ je viskozita kvapaliny a $\mathbf{f}$ sú externé sily pôsobiace na kvapalinu. Pri uvažovaní nestlačiteľnej kvapaliny je rovnica 2.11 zjednodušená na:

$$\nabla \cdot (\rho \mathbf{v}) = 0, \tag{2.13}$$

pretože $\frac{\partial \rho}{\partial t} = 0$. Pre nestlačiteľnú kvapalinu je hustota kvapaliny konštantná a nakoniec dostaneme:

$$\nabla \cdot \mathbf{v} = 0. \tag{2.14}$$

V Lagrangeovom toku kvapalín sa pozorovateľ pohybuje s tokom, preto v kontraste s Eulerovou reprezentáciou, sú hodnoty vyhodnocované v pozíciách častíc. Lagrangeova reprezentácia súvisí s Eulerovou reprezentáciou v derivácii materiálu, preto na vyjadrenie zachovania hybnosti pre konkrétnu časticu je rovnica 2.12 upravená na:

$$\frac{D\mathbf{v}}{Dt} = \frac{-\nabla p}{\rho} + \frac{\mu \nabla^2 \mathbf{v}}{\rho} + \frac{\mathbf{f}}{\rho} = \mathbf{a}^{press} + \mathbf{a}^{visco} + \mathbf{a}^{ext}. \tag{2.15}$$

$\mathbf{a}^{press}$ je akcelerácia kvapaliny spôsobená tlakom v kvapaline, $\mathbf{a}^{visco}$ je akcelerácia kvapaliny spôsobená viskozitou kvapaliny a $\mathbf{a}^{ext}$ je akcelerácia kvapaliny spôsobená externými silami ako povrchové napätie a gravitácia. V nasledujúcich častiach je vysvetlené ako sa tieto akcelerácie a ich príslušné sily aproximujú.

2.2.5 Aproximácia hustoty

Ešte pred aproximáciou síl pôsobiacich v kvapaline, musí byť najprv aproximovaná hustota kvapaliny. Dobrá aproximácia hustoty je veľmi dôležitá, pretože ovplyvňuje tlakové sily a distribúciu častíc. Existujú dva prístupy

k aproximácii hustoty. Prvý je sumácia hustoty. Na získanie aproximácie v rovnici 2.6 jednoducho $f(x)$ nahradíme ρ:

$$\begin{aligned}\langle\rho\rangle &= \sum_{j=0}^{n} \frac{m_j}{\rho_j}\rho_j W(\mathbf{x}-\mathbf{x}_j, h) \\ &= \sum_{j=0}^{n} m_j W(\mathbf{x}-\mathbf{x}_j, h),\end{aligned} \tag{2.16}$$

čo vyjadruje, že hustota je jednoducho váhovaný priemer hmoty častíc v rámci domény nosiča. Iný prístup k aproximácii hustoty je spojitá hustota. Aproximácia spojitej hustoty je založená na Lagrangeovej forme spojitej rovnice. Túto formu môžeme dostať z Eulerovskej formy (rovnica 2.11):

$$\frac{D\rho}{Dt} = -\rho(\nabla \cdot \mathbf{v}). \tag{2.17}$$

Keď požijeme SPH aproximáciu na túto rovnicu, dostaneme aproximáciu spojitej hustoty, čo je:

$$\frac{D\rho}{Dt} = -\rho \sum_{j=0}^{n} \frac{m_j}{\rho_j}\mathbf{v}_j \nabla \cdot W(\mathbf{x}-\mathbf{x}_j, h). \tag{2.18}$$

Obidve metódy majú svoje výhody aj nevýhody. Sumácia hustoty zachováva hmotu, zatiaľ čo spojitá hustota nemá túto vlastnosť. Ďalší rozdiel medzi týmito metódami je poradie výpočtu. Pri použití sumácie hustoty, hustota musí byť vyrátaná ešte pred vypočítaním síl. Naopak, keď sa používa spojitá hustota, môže byť táto hustota vyrátaná v rovnakom čase ako ostatné sily, čo robí tento prístup vhodným pre paralelizáciu.

Počas výpočtu hustoty kvapaliny blízko okraja sa v doméne nosiča nachádza menej častíc. Toto vytvára nereálne výsledky pri prístupe sumácie hustoty - častice na konci kvapaliny majú menšiu hustotu. Na vyriešenie tohto problému bolo navrhnutých niekoľko riešení. Jedným z nich je aproximácia hustoty pomocou prístupu so spojitou hustotou, ktorý netrpí problémom s

nedostatkom častíc na okraji. Ďalší prístup je použite virtuálnych častíc, pri ktorom sú na okraj pridané nové častice iba kvôli výpočtu hustoty. Posledný prístup je ľahko upravená sumácia hustoty. Hustota je normalizovaná nasledujúcim spôsobom:

$$\rho = \frac{\sum_{j=0}^{n} m_j W(\mathbf{x} - \mathbf{x}_j, h)}{\sum_{j=0}^{n} \frac{m_j}{\rho_j} W(\mathbf{x} - \mathbf{x}_j, h)}. \tag{2.19}$$

Podľa [LL03] sumácia hustoty vedie k lepším výsledkom v simulácii všeobecného fenoménu kvapalín, zatiaľ čo spojitá hustota sa zvyčajne používa pri fenoméne s veľkou nespojitosťou napríklad výbuchy alebo nárazy s veľkou rýchlosťou.

2.2.6 Aproximácia tlakovej sily

Aby sme mohli aproximovať tlakové sily použitím SPH, musíme najprv aproximovať tlakovú časť v rovnici 2.12, čo je $\frac{\nabla p}{\rho}$. Je však potrebná aproximácia gradientu, ktorá vedie k symetrickým silám. Asymetrické sily by mohli viesť k nefyzikálnemu správaniu, pretože by bol porušený druhý Newtonovský zákon. V literatúre boli použité rôzne aproximácie vedúce k symetrickým silám. Najčastejšie používaná je z [Mon92]:

$$\mathbf{f}_i^{press} = m_i \mathbf{a}_i^{press} = -m_i \frac{\nabla p_i}{\rho_i} = -m_i \sum_{j=0}^{N} m_j (\frac{p_j}{\rho_j^2} + \frac{p_i}{\rho_i^2}) \nabla W(\mathbf{x}_i - \mathbf{x}_j, h), \tag{2.20}$$

a z [MCG03]:

$$\mathbf{f}_i^{press} = m_i \mathbf{a}_i^{press} = -m_i \frac{\nabla p_i}{\rho_i} = -m_i \sum_{j=0}^{N} m_j (\frac{p_j + p_i}{\rho_j \rho_i}) \nabla W(\mathbf{x}_i - \mathbf{x}_j, h). \tag{2.21}$$

Pred vypočítaním tlakovej sily musí byť vypočítaný ešte tlak v kvapaline. Kvapalina sa bude potom pohybovať v súlade s vypočítaným tlakom z miest s vysokým tlakom do miest kde je tlak nižší. Toto je však pravdivé iba pre

stlačiteľné kvapaliny, pretože tlak je závislý od hustoty. Ak je hustota konštantná v rámci celej kvapaliny, aj tlak zostáva konštantný. Preto je kľúčové vypočítať tlakové hodnoty, ktoré majú tendenciu produkovať tlakové sily, ktoré minimalizujú stlačiteľnosť kvapaliny. Dosiahnutie nestlačiteľnosti je jedným z hlavných problémov v simulácii nestlačiteľných kvapalín.

Jedným z najjednoduchších prístupov k vypočítaniu tlakových hodnôt je použiť zákon ideálneho plynu:

$$pV = nRT, \tag{2.22}$$

kde $V = \frac{1}{\rho}$ je objem na jednotku hmoty, n je počet častíc v jednom mole, R je univerzálna plynová konštanta $(8,3145 JK^{-1}mol^{-1})$ a T je termodynamická teplota. Za predpokladu izotermickej a nestlačiteľnej kvapaliny môže byť pravá strana rovnice 2.22 nahradená konštantou plynovej tuhosti k a prepísaná na:

$$\begin{aligned} pV &= k \\ p\frac{1}{\rho} &= k \\ p &= k\rho. \end{aligned} \tag{2.23}$$

Táto rovnica produkuje iba odpudivé sily. V astrofyzike môžu byť odpudivé sily vyvážené gravitačnou silou. Pri simulácii malých scén, čo je náš prípad, môže toto zapríčiniť nestabilitu a nerealistické výsledky. Preto v [DG96] bolo navrhnuté jednoduché riešenie, kde je zavedená nová hodnota hustoty ρ_0 nazývaná pokojová hustota a rovnica 2.23 je upravená na:

$$p = k(\rho - \rho_0). \tag{2.24}$$

Táto rovnica môže byť použitá na simuláciu všeobecného fenoménu ako vylievajúca sa voda, ale vytvára nerealistické výsledky, pretože je ťažké dosiahnuť nestlačiteľnosť.

2.2.7 Slabo stlačiteľné SPH

Podobný prístup využitia rovnice 2.24 známy ako WCSPH (Weakly compressible SPH - slabo stlačiteľné SPH) bol navrhnutý v [BT07] kde bola Taitova rovnica z [Bat67] použitá na vyhodnotenie tlakových hodnôt. V porovnaní s rovnicou 2.24 môže byť dosiahnutá vyššia úroveň nestlačiteľnosti a môžu byť aj zvolené väčšie časové kroky. Taitova rovnica má tvar:

$$p = B((\frac{\rho}{\rho_0})^\gamma - 1), \tag{2.25}$$

najlepší výsledok bol dosiahnutý pre $\gamma = 7$. Konštanta B riadi fluktuáciu hustoty v kvapaline. Aby sme vedeli stanoviť hodnotu B, predpokladá sa, že efekt stlačiteľnosti škáluje s $O(M^2)$, kde M je Machovo číslo pre tok. Na základe tohto predpokladu môže byť zapísaný nasledujúci vzťah:

$$\frac{\Delta\rho}{\rho_0} \sim \frac{|\mathbf{v}_f|^2}{c_s^2}, \tag{2.26}$$

kde $\mathbf{v}_f$ je rýchlosť toku a c_s je rýchlosť zvuku v kvapaline. Môžeme zaručiť, že $\frac{|\Delta\rho|}{\rho_0} \sim \eta$, ak je rýchlosť zvuku v kvapaline dostatočne vysoká, že platí $\frac{|\mathbf{v}_f|^2}{c_s^2} < \eta$. Konštanta η je zvolená dostatočne malá, aby sa zabránilo fluktuácii hustoty (napríklad $\eta = 0,01$ dovoľujúca kolísanie menšie ako 1%). Na vynútenie tejto podmienky je B zvolené ako:

$$k = \frac{\rho_0 c_s^2}{\gamma}. \tag{2.27}$$

Nevýhoda tohto prístupu je, že c_s nie je fyzikálna konštanta, ale je ovplyvnená podľa zvoleného η a rýchlosti toku. Preto je k ťažké zvoliť ešte pred spustením simulácie. Animátor sa nemôže zaobísť bez rozsiahleho testovania a ladenia parametra. Ďalšia nevýhoda je, že keď chceme dosiahnuť malú stlačiteľnosť, B musí byť zvolené veľké, preto musia byť zvolené malé časové kroky, aby bola zachovaná stabilita ale výpočtový čas sa takto predlžuje.

2.2.8 Prediktívno-korektívne nestlačiteľné SPH

Prístup známy ako PCISPH (Predictive-corrective incompressible SPH alebo Prediktívno-korektívne nestlačiteľné SPH) bol použitý v [SP09]. Tlakové hodnoty sú iteratívne počítané vzhľadom na predpovedané pozície častíc. Iteráciami je možné dosiahnuť ľubovoľnú úroveň nestlačiteľnosti. Podrobnejší popis a vysvetlenie vyrátania tlaku je daný na nasledujúcich riadkoch.

Hustota i-tej častice v čase $t+1$ za predpokladu rovnakej hmoty pre všetky častice a vyhladzovaciu dĺžku h je vyrátaná pomocou sumácie SPH hustoty:

$$\begin{aligned}\rho_i(t+1) &= m\sum_{j=0}^{N} W(\mathbf{x}_i(t+1) - \mathbf{x}_j(t+1)) \\ &= m\sum_{j=0}^{N} W(\mathbf{x}_i(t) + \Delta\mathbf{x}_i(t) - \mathbf{x}_j(t) - \Delta\mathbf{x}_j(t)) \\ &= m\sum_{j=0}^{N} W(\mathbf{d}_{ij}(t) + \Delta\mathbf{d}_{ij}(t)),\end{aligned} \tag{2.28}$$

kde $\mathbf{d}_{ij} = \mathbf{x}_i(t) - \mathbf{x}_j(t)$, $\Delta\mathbf{d}_{ij} = \Delta\mathbf{x}_i(t) - \Delta\mathbf{x}_j(t)$ a m je hmota častice. Za predpokladu, že Δd_{ij} je relatívne malé, môžeme aplikovať Taylorovu aproximáciu prvého rádu, čoho výsledkom je:

$$\begin{aligned}\rho_i(t+1) &= m\sum_{j=0}^{N} W(\mathbf{d}_{ij}(t)) + \nabla W(\mathbf{d}_{ij}(t)) \cdot \Delta\mathbf{d}_{ij}(t) \\ &= m\sum_{j=0}^{N} W(\mathbf{x}_i(t) - \mathbf{x}_j(t)) + m\sum_{j=0}^{N} \nabla W(\mathbf{x}_i(t) - \mathbf{x}_j(t)) \cdot (\Delta\mathbf{x}_i(t) - \Delta\mathbf{x}_j(t)) \\ &= \rho_i(t) + \Delta\rho_i(t).\end{aligned} \tag{2.29}$$

Keď použijeme $W_{ij} = W(\mathbf{x}_i(t) - \mathbf{x}_j(t))$, výraz $\Delta\rho_i(t)$ môžeme zapísať ako:

$$\begin{aligned}\Delta\rho_i(t) &= m\sum_{j=0}^{N}\nabla W_{ij}\cdot(\Delta\mathbf{x}_i(t) - \Delta\mathbf{x}_j(t)) \\ &= m\left(\sum_{j=0}^{N}\nabla W_{ij}\Delta\mathbf{x}_i(t) - \sum_{j=0}^{N}\nabla W_{ij}\Delta\mathbf{x}_j(t)\right) \\ &= m\left(\Delta\mathbf{x}_i(t)\sum_{j=0}^{N}\nabla W_{ij} - \sum_{j=0}^{N}\nabla W_{ij}\Delta\mathbf{x}_j(t)\right).\end{aligned} \tag{2.30}$$

Hodnota $\Delta\mathbf{x}$ je odvodená z integračnej schémy zanedbávajúcej všetky sily okrem tlakovej:

$$\Delta\mathbf{x}_i = \Delta t^2\frac{\mathbf{f}_i^{press}}{m}. \tag{2.31}$$

Uvažujúc nestlačiteľný tok, všetky častice majú hustotu rovnú pokojovej hustote a všetky hodnoty tlakov sú rovnaké. Toto nám dáva nasledujúce vyjadrenie tlakovej sily:

$$\begin{aligned}\mathbf{f}_i^{press} &= -m^2\sum_{j=0}^{N}\left(\frac{\tilde{p}_i}{\rho_0^2} + \frac{\tilde{p}_i}{\rho_0^2}\right)\nabla W_{ij} \\ &= -m^2\frac{2\tilde{p}_i}{\rho_0^2}\sum_{j=0}^{N}\nabla W_{ij}.\end{aligned} \tag{2.32}$$

Vložením tejto tlakovej sily do rovnice 2.31 dostaneme:

$$\Delta\mathbf{x}_i = -\Delta t^2 m\frac{2\tilde{p}_i}{\rho_0^2}\sum_{j=0}^{N}\nabla W_{ij}. \tag{2.33}$$

Keď máme všetky tieto odhady, môžeme vložiť rovnicu 2.33 do rovnice 2.30 a dostaneme:

$$\begin{aligned}\Delta\rho_i(t) &= m\left(-\Delta t^2 m\frac{2\tilde{p}_i}{\rho_0^2}\sum_{j=0}^{N}\nabla W_{ij}\cdot\sum_{j=0}^{N}\nabla W_{ij} - \sum_{j=0}^{N}\left(\nabla W_{ij}\cdot\Delta t^2 m\frac{2\tilde{p}_i}{\rho_0^2}\nabla W_{ij}\right)\right) \\ &= \Delta t^2 m^2\frac{2\tilde{p}_i}{\rho_0^2}\left(-\sum_{j=0}^{N}\nabla W_{ij}\cdot\sum_{j=0}^{N}\nabla W_{ij} - \sum_{j=0}^{N}(\nabla W_{ij}\cdot\nabla W_{ij})\right).\end{aligned} \tag{2.34}$$

Vyriešením rovnice 2.34 pre $\tilde{p}_i$ dostaneme:

$$\tilde{p}_i = \frac{\Delta\rho_i(t)}{\Delta t^2 m^2 \frac{2}{\rho_0^2}\left(-\sum_{j=0}^{N}\nabla W_{ij}\cdot\sum_{j=0}^{N}\nabla W_{ij} - \sum_{j=0}^{N}(\nabla W_{ij}\cdot\nabla W_{ij})\right)}. \quad (2.35)$$

Aby sme urýchlili výpočet a zabránili problému s nedostatkom častíc v doméne nosiča, je hodnota δ predpočítaná na prototype častice s jej susedmi:

$$\delta = \frac{-1}{\Delta t^2 m^2 \frac{2}{\rho_0^2}\left(-\sum_{j=0}^{N}\nabla W_{ij}\cdot\sum_{j=0}^{N}\nabla W_{ij} - \sum_{j=0}^{N}(\nabla W_{ij}\cdot\nabla W_{ij})\right)}. \quad (2.36)$$

V každej iterácii sú pre častice predikované nové pozície. Podľa predikovaných pozícií sú aktualizované tlakové hodnoty pomocou predpočítanej δ:

$$\begin{aligned} \rho_i^{err} &= \rho_i^{pred} - \rho_0 \\ \tilde{p}_i &= \delta\rho_i^{err} \\ p_i &= p_i + \tilde{p}_i, \end{aligned} \quad (2.37)$$

kde ρ_i^{pred} je hustota i-tej častice na predikovanej pozícii. Predikcia začína vždy na rovnakej pozícii a pozičné hodnoty nie sú akumulované. Tento proces je iterovaný pokiaľ nie je dosiahnutá požadovaná nízka fluktuácia hustoty.

2.2.9 Implicitne nestlačiteľné SPH

Prístup IISPH (Implicit incompressible SPH - implicitne nestlačiteľné SPH) bol použitý v [ICS+14]. IISPH je založené na čiastočne implicitnej forme predikcie hustoty. Formuláciu získame z priamej diskretizácie spojitej rovnice $\frac{D\rho}{Dt} = -\rho\nabla\cdot\mathbf{v}$ v čase t pomocou rozdielu pre časové odvodenie hustoty $\frac{\rho_i(t+\Delta t)-\rho_i(t)}{\Delta t}$ a SPH konceptu pre divergenciu rýchlosti

$$\nabla\cdot\mathbf{v}_i = -\frac{1}{\rho_i}\sum_{j=0}^{N} m_j\mathbf{v}_{ij}\nabla W_{ij}, \quad (2.38)$$

ktorý tvorí:

$$\frac{\rho_i(t+\Delta t)-\rho_i(t)}{\Delta t}=\sum_{j=0}^{N} m_j \mathbf{v}_{ij}(t+\Delta t)\nabla W_{ij}(t). \tag{2.39}$$

Táto konkrétna diskretizácia zavádza neznáme rýchlosti $\mathbf{v}_{ij}(t+\Delta t)=\mathbf{v}_i(t+\Delta t)-\mathbf{v}_j(t+\Delta t)$, ktoré závisia od neznámych tlakových síl v čase t, ktoré sú lineárne k neznámym tlakovým hodnotám v čase t. Použitím čiastočne implicitnej Eulerovej schémy pre aktualizáciu pozície a rýchlosti, môže byť rýchlosť z rovnice 2.39 prepísaná na:

$$\mathbf{v}_i(t+\Delta t)=\mathbf{v}_i(t)+\Delta t\frac{\mathbf{F}_i^{adv}(t)+\mathbf{F}_i^{p}(t)}{m_i}, \tag{2.40}$$

s neznámymi tlakovými silami $\mathbf{F}_i^p(t)$ a známymi silami $\mathbf{F}_i^{adv}(t)$ ako gravitácia, povrchové napätie a viskozita. Sledujúc koncepciu odhadu, uvažujeme predikované rýchlosti $\mathbf{v}_i^{adv}=\mathbf{v}_i(t)+\Delta t\frac{\mathbf{F}_i^{adv}(t)}{m_i}$ čo vedie k hustote:

$$\rho_i^{adv}=\rho_i(t)+\Delta t\sum_{j=0}^{N} m_j \mathbf{v}_{ij}^{adv}\nabla W_{ij}(t). \tag{2.41}$$

Ďalej budeme riešiť tlakové sily, aby sme tým vyriešili stlačenie - odchýlku od pokojovej hustoty:

$$\Delta t^2\sum_{j=0}^{N} m_j\left(\frac{\mathbf{F}_i^p(t)}{m_i}-\frac{\mathbf{F}_j^p(t)}{m_j}\right)\nabla W_{ij}(t)=\rho_0-\rho_i^{adv}. \tag{2.42}$$

Poznamenajme, že 2.42 zodpovedá 2.39 pre $\rho_i(t+\Delta t)=\rho_0$. Dosadením rovnice, ktorej tlakové sily zachovávajú hybnosť:

$$\mathbf{F}_i^p(t)=-m_i\sum_{j=0}^{N} m_j\left(\frac{p_i(t)}{\rho_i^2(t)}+\frac{p_i(t)}{\rho_i^2(t)}\right)\nabla W_{ij}(t). \tag{2.43}$$

do rovnice 2.42, dostaneme lineárny systém $\mathbf{A}(t)\mathbf{p}(t)=\mathbf{b}(t)$ s jednou rovnicou a jednou neznámou tlakovou hodnotou $p_i(t)$ pre každú časticu. $\mathbf{b}(t)$

zodpovedá pravej strane rovnice 2.42, čiže $b_i(t) = \rho_0 - \rho_i^{adv}$. Pre každú časticu nakoniec dostaneme rovnicu v tvare:

$$\sum_{j=0}^{N} a_{ij} p_j = b_i = \rho_0 - \rho_i^{adv}, \tag{2.44}$$

pre zlepšenie čitateľnosti sme vynechali čas t.

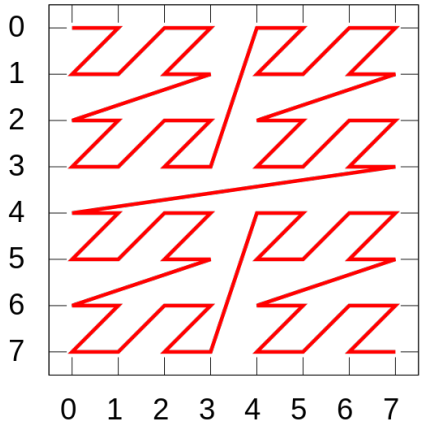
2.3 Hľadanie k-najbližších susedov

Jednou z najdôležitejších častí v SPH metóde je hľadanie k-najbližších susedov. Pre každú časticu sa v jej okolí hľadajú všetky častice, ktoré sú od danej častice vzdialené maximálne na vzdialenosť h. V tejto kapitole si rozoberieme tri metódy. Jednu, ktorú sme použili v našej implementácii v OpenCL, druhú ktorú sme s ňou porovnávali a pre porovnanie uvádzame ešte aj tvz. brute-force metódu, ktorá je veľmi neefektívna.

2.3.1 Z-Indexing a triedenie

Túto metódu z [Gre10] na hľadanie k-najbližších susedov sme použili v našej implementácii SPH. Hlavnou časťou metódy je tzv. Mortonova Z-krivka, ktorá mapuje viacrozmerný priestor do jednorozmerného. V roku 1966 ju prestavil Guy M. Morton. Svoje využitie má napríklad pri indexovaní viacrozmerných dát, čo je presne náš prípad. Na Obr. 2.3 je ukážka Mortonovho rozkladu.

Majme na začiatku zoznam všetkých častíc v systéme a nazvime ho A. Metóda pre každú časticu vypočíta jej Z-Index z jej pozície pomocou Mortonovej Z-krivky. Túto hodnotu uloží spolu aj s indexom častice v zozname všetkých častíc do jednorozmerného zoznamu dvojíc, kde prvé číslo dvojice (kľúč) je Z-Index častice a druhé číslo (hodnota) je jej index v zozname častíc



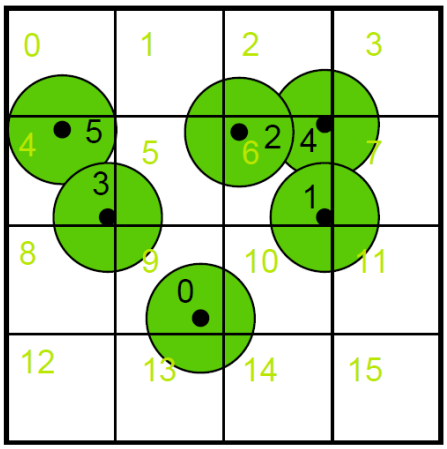
Obr. 2.3: Mortonov rozklad dvojrozmernej matice 8x8.

A. Nazvime tento zoznam B. Výhoda Mortonovej Z-krivky spočíva v tom, že častice, ktoré majú Z-Index blízko pri sebe sa aj v priestore nachádzajú blízko pri sebe. Týmto spôsobom môžeme dosiahnuť vyšší výkon pri hľadaní susedov, pretože keď sme už spracovávali časticu, ktorá sa nachádza blízko pri častici, ktorú práve spracovávame, je možné že hodnoty tejto častice sú už v pamäti procesora načítané a netreba ich z pamäte RAM znova načítavať do pamäte procesora. Keď procesor načítava nejakú hodnotu z pamäte, tak nenačíta iba tú samotnú hodnotu z pamäte, ale aj hodnoty niekoľkých okolitých prvkov a keď potom chceme získať hodnotu nejakého tohto okolitého prvku, tak je už táto hodnota načítaná v pamäti procesora a netreba ju už načítavať z pamäte RAM. Každú časticu teda virtuálne zaradíme do bunky v mriežke priestoru, ktorej veľkosť v osiach x, y a z je vzdialenosť h, v ktorej hľadáme všetkých susedov častice.

Keď máme takto vytvorený zoznam Z-Indexov a indexov do poľa častíc, utriedime tento zoznam podľa Z-Indexov. V našej implementácii sme použili

paralelné Triedenie zlučovaním (angl. Merge sort). Existujú ešte aj iné paralalné triediace algoritmy ako Radix sort a Bitonic sort, ktorých paralelná implementácia je tiež veľmi efektívna. Týmto dostaneme zoznam indexov častíc v jednotlivých bunkách.

V ďalšom kroku vytvoríme zoznam indexov C do utriedeného zoznamu B, ktorý sme vytvorili v predchádzajúcom kroku. Tento zoznam sa dá vytvoriť paralelne pre každú bunku, kde zoberieme Z-Index bunky a nájdeme v zozname B prvú časticu s týmto Z-Indexom. Na Obr. 2.4 je ukážka tejto metódy pre dvojrozmernú mriežku, kde je ale použité jednoduché lineárne indexovanie buniek.



Index	Neutriedený zoznam B (Z-Index, index častice)	Utriedený zoznam B podľa Z-Indexov	Zoznam indexov C
0	(9, 0)	(4, 3)	
1	(6, 1)	(4, 5)	
2	(6, 2)	(6, 1)	
3	(4, 3)	(6, 2)	
4	(6, 4)	(6, 4)	0
5	(4, 5)	(9, 0)	
6			2
7			
8			
9			5
10			
11			
12			
13			
14			
15			

Obr. 2.4: Dvojrozmerná mriežka pomocou lineárneho indexovania [Gre10].

Keď máme takto vytvorené zoznamy, pri hľadaní susedov konkrétnej častice si najprv vyrátame jej Z-Index a zistíme tým, v ktorej bunke sa častica nachádza. Potom zoberieme okolitých 26 buniek a bunku, v ktorej sa častica nachádza a do zoznamu susedov dávame všetky častice z týchto buniek, ktoré majú vzdialenosť menšiu alebo rovnú h. Treba ešte raz podotknúť, že veľ-

kosť každej bunky vo všetkých osiach je práve h, čiže všetci susedia častice na vzdialenosť h sa budú nachádzať iba v týchto bunkách.

2.3.2 Indexovanie pomocou smerníkov

Táto metóda bola použitá v implementácii SPH, ktorú uvádzame ako porovnanie k našej implementácii SPH. Táto metóda využíva možnosti smerníkovej aritmetiky, ktorú v OpenCL kerneloch nie je možné využiť. Metóda používa tiež rozdelenie celej domény na mriežku buniek. Každá bunka obsahuje spájaný zoznam častíc, ktoré sa v bunke nachádzajú. Bunka má v sebe smerník na prvú časticu a každá častica má v sebe smerník na ďalšiu časticu, ktorá sa nachádza v tejto bunke. Celá mriežka je uložená v pamäti a pre každú časticu sa pri hľadaní susedov spočíta z jej pozície index na bunku v tejto mriežke a zaradí sa do spájaného zoznamu častíc v príslušnej bunke. Na vypočítanie indexu sa dá tiež použiť Z-Indexing, no v implementácii, ktorú používame pre porovnanie sa index počíta spôsobom $1 * x + dimX * y + dimX * dimY * z$ kde x, y a z je pozícia bunky, v ktorej sa častica nachádza a $dimX$, $dimY$ je veľkosť celej mriežky v osiach x a y.

Pri tomto prístupe netreba triediť mriežku podľa indexov, pretože každá bunka mriežky obsahuje spájaný zoznam častíc. Toto sa v predchádzajúcom prístupe (Z-Indexing) nedá robiť, pretože ten prístup nevyužíva smerníky a spájaný zoznam častíc, ale pre každú časticu si spočíta jej Z-Index a uloží ho do zoznamu. Potom je potrebné tento zoznam ešte utriediť, aby sme mali bunky a častice, ktoré sa v nich nachádzajú uložené v zozaname za sebou a mohli tak hneď získať všetky častice, ktoré sa v danej bunke nachádzajú. Táto metóda teda šetrí čas potrebný na triedenie.

Po vytvorení tejto mriežky, v ktorej sa nachádzajú všetky častice v nej môžeme hľadať susedov podobným spôsobom ako pri Z-Indexingu a triedení.

Vyrátame si z pozície častice bunku, v ktorej sa nachádza a potom prejdeme spájané zoznamy 26 susediacich buniek a tejto bunky a susediace častice sú všetky tie, ktoré sa v týchto bunkách nachádzajú do vzdialenosti h. Aj pri tejto metóde platí, že všetci susedia častice sa nachádzajú iba v týchto 27 bunkách pokiaľ majú bunky veľkosť h vo všetkých osiach.

2.3.3 Brute-force metóda

Táto metóda je najjednoduchšia na implementáciu, ale zároveň aj najmenej efektívna, pričom jej dolný aj horný odhad časovej zložitosti je $O(n^2)$. Uvádzame ju však pre porovnanie k predchádzajúcim dvom metódam.

Táto metóda pri hľadaní k-najbližších susedov častice postupne prechádza celý zoznam častíc a ak má daná častica vzdialenosť k našej častici menšiu alebo rovnú h, tak sa berie ako jej susediaca častica. Pre každú časticu v zozname teda prejde celý zoznam častíc a hľadá v ňom častice, ktoré sa nachádzajú do danej vzdialenosti, z čoho vyplýva časová zložitosť $O(n^2)$.

Kapitola 3

Návrh softvérového diela

V tejto kapitole rozoberáme rozdelenie našej implementácie na jednotlivé časti. Zaoberáme sa v nej aj architektúrou a technológiami, ktoré sme použili v našej implementácii.

3.1 Cieľ práce

Hlavným cieľom našej práce je paralelizácia SPH metódy a jej implementácia v knižnici OpenCL, vďaka čomu vieme SPH simulovať na zariadeniach ako GPU. GPU zariadenia sú dizajnované na paralelné výpočty, pričom paralelné algoritmy sú oproti ich sekvenčným verziám spracovávané na GPU zariadeniach oveľa rýchlejšie. Paralelizáciou SPH a vykonávaním výpočtov na GPU vieme dosiahnuť zrýchlenie celej simulácie. Metóda SPH je veľmi dobre paralelizovateľná, pretože sily pre jednotlivé častice môžu byť vypočítané nezávisle od síl pre ostatné častice. Ak by bol počet procesorov väčší alebo rovný ako je počet častíc, vieme v jednom kroku vypočítať sily pre všetky častice naraz.

3.2 Technológie

Pre paralelizáciu SPH sme použili knižnicu OpenCL. Ďalšou variantou bola knižnica CUDA. Knižnicu CUDA je však možné používať iba na grafických kartách firmy NVIDIA, ale knižnicu OpenCL je možné použiť aj na GPU od firmy AMD a aj NVIDIA, čo bol hlavný dôvod pre výber OpenCL na paralelizáciu SPH. Na vizualizáciu častíc sme použili knižnicu OpenGL a implementáciu sme realizovali v programovacom jazyku C++ v prostredí Microsoft Visual Studio 2013.

3.3 Architektúra

Architektúru našej implementácie môžeme rozdeliť na dva hlavné logické celky:

- Vizualizačná časť
- Simulačná časť

Vizualizačná časť má na starosti zobrazovanie výstupu simulácie v podobe jednotlivých simulovaných častíc kvapaliny, objektov a domény, v ktorej sa častice nachádzajú. Vizualizačná časť využíva knižnicu OpenGL na vykresľovanie scény. Touto časťou sa v návrhu už viac nebudeme zaoberať pretože jej architektúra je jednoduchá a implementácia priamočiara a nie je pre nás taká zaujímavá ako práve simulačná časť, ktorá je zároveň aj stredobodom tejto práce.

Simulačná časť má za úlohu riešiť simuláciu metódy SPH. V nasledujúcich častiach sa podrobnejšie zoznámime s návrhom simulačnej časti.

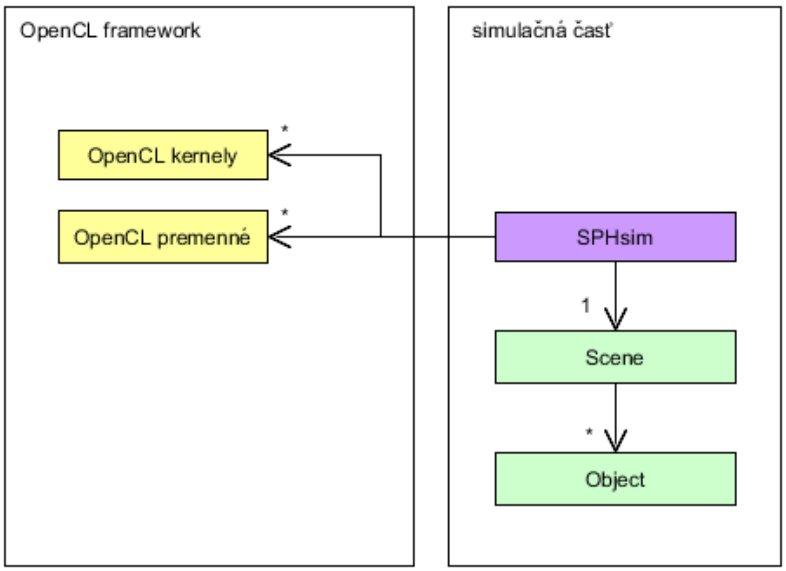
3.4 Simulačná časť

Simulačná časť využíva na simuláciu SPH knižnicu OpenCL a všetky výpočty a kroky simulácie vykonáva v rámci OpenCL zariadení. Preto je simulačná časť rozdelená ešte na dve časti, pričom prvá je tvorená kódom, tvz. OpenCL kernelmi, ktorý predstavuje všetky výpočty SPH simulácie a je vykonávaný na OpenCL zariadeniach. Táto časť zahŕňa aj všetky dátové štruktúry pre simuláciu SPH, ktoré sú uložené v pamäti OpenCL zariadenia. Celú SPH simuláciu, všetky jej výpočty a dátové štruktúry sme teda presunuli na OpenCL zariadenie, na ktorom celá simulácia prebieha.

Druhú časť tvorí kód, ktorý spúšťa všetky výpočty na OpenCL zariadeniach v správnom poradí a má na starosti, aby mohla byť simulácia SPH vôbec vykonávaná. Jedná sa o kód v C++, ktorý využíva API funkcie OpenCL, ktoré umožňujú vytváranie, správu a spúšťanie OpenCL kernelov na OpenCL zariadeniach. Vytvorenie, spustenie a správa paralelných výpočtov na OpenCL zariadeniach je proces v programovacom jazyku, ktorý zahŕňa množstvo volaní OpenCL funkcií. Vyplýva to aj z toho, že OpenCL sám o sebe je nízkoúrovňový framework. Kód na správu programov uskutočňujúcich paralelné výpočty sa v podstate vždy zhoduje a preto sa prirodzene vyskytla požiadavka vytvoriť znovu použiteľný kód vo forme tried, ktoré budú volať funkcie OpenCL API. Pre účely implementácie paralelnej SPH metódy sme vytvorili nad samotným OpenCL ešte vyšší stupeň abstrakcie vo forme frameworku. Tento framework si popíšme v časti 3.5 OpenCL framework.

3.4.1 Štruktúra tried

Štruktúru tried a architektúru simulačnej časti znázorňuje Obr. 3.1. Postupne si rozoberieme jednotlivé triedy z tejto časti.



Obr. 3.1: Diagram tried simulačnej časti.

Trieda SPHsim

Najdôležitejším prvkom celej simulačnej časti je trieda SPHsim, ktorá obsahuje celú logiku simulácie a má za úlohu spúšťanie OpenCL kernelov. Trieda obsahuje premenné tried OpenCL frameworku, ktoré zahŕňajú OpenCL kernely na vykonávanie SPH simulácie a dátové štruktúry potrebné pre simuláciu častíc. Trieda taktiež obsahuje aj logiku na načítavanie scén a objektov v nich.

Trieda Scene

Táto trieda v sebe udržiava všetky objekty nachádzajúce sa v scéne. Je to dôležitý prostredník medzi vizualizačnou a simulačnou časťou, pretože triedy vo vizualizačnej časti a aj trieda SPHsim má k tejto triede prístup. Simulačná časť vykonáva všetky výpočty na OpenCL zariadení a vždy po vykonaní jed-

ného kroku simulácie, stiahne trieda SPHsim novo vypočítané pozície všetkých častíc z pamäte OpenCL zariadenia. Trieda Scene má k týmto pozíciám častíc prístup, vďaka čomu ich môže vizualizačná časť vykresliť spolu aj s inými objektami v scéne.

Trieda Object

Účelom tejto triedy je správa geometrie objektov v scéne. Geometriu objektov je možné načítavať z .obj súborov. Trieda poskytuje metódy na transláciu, škálovanie a vykreslenie objektu pomocou OpenGL. Najdôležitejšou funkciou triedy je ale metóda na navzorkovanie povrchu objektu časticami. Vďaka tejto metóde môže mať kvapalina kolíziu s objektom. Túto triedu využíva aj vizualizačná časť implementácie na vykreslenie objektov, ktoré sa nachádzajú v scéne.

OpenCL kernely

Všetky OpenCL kernely sú uložené v samostatných súboroch. V týchto kerneloch sú uložené výpočty pre celú SPH simuláciu. Rozdelili sme ich do šiestich kategórií podľa ich účelu:

- *Hľadanie susedov* - kernely, ktoré slúžia na hľadanie k-najbližších susedov jednotlivých častíc.
- *Správa stavov* - tieto kernely majú za úlohu ukladanie, načítavanie a vymazávanie vlastností a síl častíc.
- *Výpočet síl* - úlohou týchto kernelov je výpočet síl pôsobiacich na častice jednotlivými metódami SPH.

- *Integrácia* - v tejto kategórii sa nachádzajú kernely, ktoré vypočítajú aktuálne pozície častíc vzhľadom na sily, ktoré na ne pôsobia.
- *Pomocné funkcie* - tu sú pomocné kernely pre správne fungovanie simulácie.
- *Doplnkové funkcie* - funkcie, ktoré sú pripojené do kernelov, ktoré ich potrebujú pre správne fungovanie. Je tu napríklad funkcia na výpočet štvorca vzdialenosti medzi dvoma časticami.

OpenCL premenné

Trieda SPHsim obsahuje premenné, ktoré sú nevyhnutné na simuláciu SPH, a s ktorými pracujú OpenCL kernely. Tieto premenné sú prevažne buffre, ktoré tvoria zoznamy častíc. Každá častica má nielen svoju pozíciu, ale aj vlastnosti a sily, ktoré na ňu pôsobia a zoznam susediacich častíc. Sú tu aj konštanty pre SPH simuláciu ako napríklad časový krok, gravitačné zrýchlenie atď.

3.5 OpenCL framework

OpenCL framework spája logické rozdelenie a správne poradie volania jednotlivých funkcií OpenCL API do tried. Náš framework umožňuje rýchlo začať programovať OpenCL aplikácie aj bez detailnej znalosti samotného OpenCL, nakoľko framework je jednoduchý na pochopenie. Programátor teda nemusí úplne vedieť čo sa deje v pozadí frameworku na to, aby mohol programovať OpenCL aplikácie. Náš framework bol z veľkej časti inšpirovaný OpenCL frameworkom použitým v OpenCL implementácii SPH metódy [Isp]. Framework je naprogramovaný tak, aby sa dal ľahko prevziať a integrovať do iných projektov a aby bolo umožnené hneď začať písať OpenCL programy a pou-

žívať ich pre paralelizáciu algoritmov. Na ďalších stranách sa zameriame na logickú štruktúru nášho frameworku s popisom tried.

3.5.1 Štruktúra tried

Framework sa skladá z 13 tried, ktoré sa dajú logicky rozdeliť na 4 časti podľa ich spoločnej funkcionality. Rozdelenie je v súlade s rozdelením OpenCL na časti spomínaným v časti 2.1.1 Architektúra OpenCL. V tejto sekcii uvádzame zoznam jednotlivých častí so zoznamom a popisom funkcionality tried, ktoré sa v týchto častiach nachádzajú.

Platformová časť

Triedy v platformovej časti frameworku majú za úlohu spravovanie OpenCL zariadení a host zariadenia. Umožňujú získať hierarchiu zariadení a informácie o nich. Platformovú časť tvoria tieto triedy:

Trieda CLSystem

Trieda CLSystem je implementovaná návrhovým vzorom singleton. Je to základná trieda z platformovej časti. Slúži na spravovanie OpenCL systému v rámci host zariadenia. Trieda je hneď pri statickom prístupe k nej inicializovaná. Inicializácia prebieha tak, že sa nájdu a inicializujú všetky OpenCL platformy v rámci daného host zariadenia. Nájdené OpenCL platformy sú reprezentované triedou CLPlatform. Trieda obsahuje aj statickú metódu na získanie veľkosti OpenCL dátového typu a metódu na získanie popisu chyby, ktorá sa vyskytla pri volaní jednej z OpenCL funkcií. Pomocou triedy CLSystem sa príslušnou metódou dá pristupovať ku všetkým OpenCL platformám. Trieda umožňuje aj profiling, ktorý keď je nastavený, tak sa po vykonaní

niektorých OpenCL funkcií, ktoré môžu byť časovo náročné vypisuje dĺžka trvania danej funkcie do konzoly.

Trieda CLPlatform

Inštancie tejto triedy vytvára trieda CLSystem. Trieda slúži na vytvorenie abstrakcie pre správu OpenCL platforiem, v rámci ktorých sa nachádzajú OpenCL zariadenia. Trieda obsahuje inštanciu triedy CLDevice pre každé zariadenie, ktoré sa v danej platforme nachádza. Hneď po vytvorení inštancie triedy CLPlatform sa pomocou funkcií OpenCL API nájdu a inicializujú OpenCL zariadenia a vytvoria sa inštancie triedy CLDevice. Pomocou príslušnej metódy triedy CLPlatform sa dá pristupovať ku všetkým OpenCL zariadeniam nachádzajúcim sa na danej platforme.

Trieda CLDevice

Triedu CLDevice spravuje a jej inštancie vytvára trieda CLPlatform. Trieda predstavuje OpenCL zariadenie nachádzajúce sa na danej OpenCL platforme. Hneď po vytvorení inštancie triedou CLPlatform je inštancia inicializovaná pričom sa získavajú všetky podrobné informácie o danom OpenCL zariadení. Na OpenCL zariadeniach sa vykonávajú kernel funkcie programov, CLDevice je preto základnou triedou na vykonávanie programov.

Výpočtová časť

Výpočtová časť zahŕňa triedy, ktoré sa starajú o načítanie kernel funkcií a funkcií napísaných v jazyku C99, ich kompilovanie a vykonávanie na OpenCL zariadeniach. Jednotlivé triedy v rámci výpočtovej časti sú:

Trieda CLLink

Trieda CLLink slúži ako prostredník medzi triedami CLDevice a CLProgram. Pri vytváraní inštancie triedy CLLink sa jej predajú zariadenia alebo celá platforma, na ktorej majú byť spúšťané programy a vytvorí sa asociácia s týmito zariadeniami. Trieda CLLink vlastne predstavuje pripojenie na zariadenia, na ktorých budú programy kompilované a spúšťané, umožňuje aj vytváranie bufferov na daných zariadeniach. Je to hlavná trieda, ktorá umožňuje komunikáciu medzi host zariadením a OpenCL zariadeniami. Obsahuje aj objekt command queue, v ktorom sa nachádzajú všetky príkazy, ktoré majú byť na OpenCL zariadeniach vykonané.

Trieda CLProgram

Táto trieda predstavuje spustiteľný program. Pri vytvorení inštancie triedy sa jej predá cesta k súboru zo zdrojovým kódom v jazyku C99, ktorý má byť vykonávaný na OpenCL zariadeniach pričom sa hneď aj načíta jeho obsah. Trieda obsahuje zoznam všetkých argumentov pre kernel funkciu v danom programe. Tieto argumenty môžu byť inštancii triedy nastavené pomocou príslušnej metódy. Ďalej je potrebné nastaviť aj odkaz na príslušný CLLink, aby mohol byť program skompilovaný a spúšťaný na OpenCL zariadeniach, na ktoré je CLLink pripojený. Trieda umožňuje komplexné spravovanie OpenCL programu od jeho vytvorenia, skompilovania, spúšťania a správneho uvoľnenia z pamäte. Aby bolo toto všetko umožnené trieda volá funkcie OpenCL API v správnom poradí a so správnymi parametrami, no je však potrebné, aby boli parametre správne nastavené a samotné metódy triedy tiež volané v správnom poradí pre jej korektné fungovanie. Napríklad ešte pred spustením programu je potrebné ho skompilovať a zaradiť do objektu command queue v rámci OpenCL zariadenia.

Trieda CLFunction

Trieda CLFunction slúži na načítanie ľubovoľnej OpenCL funkcie zo súboru a jej následné vloženie do OpenCL programu, ktorý môže vo svojom zdrojovom kóde túto funkciu potom zavolať.

Pamäťová časť

Pamäťovú časť tvoria triedy na správu a alokáciu pamäte na OpenCL zariadeniach. Umožňujú vyhradenie pamäťových blokov, ich inicializáciu a nastavenie, ich odosielanie na OpenCL zariadenia a ich získavanie z OpenCL zariadení späť. Pamäťová časť frameworku sa skladá z týchto tried:

Trieda CLVariable

CLVariable je abstraktná bázová trieda pre premenné, ktoré sa predajú ako argumenty kernel funkciám v inštanciách triedy CLProgram. Z tejto triedy nie je možné vytvárať inštancie, je ale možné vytvárať inštancie tried, ktoré od nej dedia. Trieda definuje typ premennej, rozhranie pre jej alokáciu a uvoľnenie a dátovú štruktúru umožňujúcu pristupovať k dátam premennej. Trieda má aj priradený CLLink, ktorý je asociovaný so zariadeniami, na ktorých bude premenná alokovaná a na ktorých sa bude používať.

Trieda CLKernelArgument

CLKernelArgument je odvodená trieda od triedy CLVariable. Predstavuje premennú, ktorá slúži ako argument pre kernel funkciu v CLProgram a má alokovanú pamäť v súkromnej časti pamäti OpenCL zariadenia.

Trieda CLLocalBuffer

CLLocalBuffer je odvodená trieda od triedy CLVariable. Zastupuje OpenCL buffer, ktorý je alokovaný na OpenCL zariadení v lokálnej pamäti.

Trieda CLGlobalBuffer

CLGlobalBuffer je odvodená trieda od triedy CLVariable. Je to OpenCL buffer, ktorý je alokovaný globálne v rámci OpenCL zariadenia. Do buffra je po vytvorení inštancie možné nastaviť pole hodnôt, ktoré budú potom pomocou príslušnej funkcie odoslané na zariadenie do jeho globálnej pamäti. Dá sa použiť aj ako výstupné pole, do ktorého budú po vykonaní kernel funkcie uložené vypočítané hodnoty a je možné ich stiahnuť pomocou príslušnej metódy z pamäti OpenCL zariadenia do pamäti host zariadenia.

Trieda CLConstant

Pomocou triedy CLConstant je možné zadefinovať si do OpenCL programov konštanty skalárnych alebo vektorových typov. Konštanta je pridaná do OpenCL programu ešte pred jeho skompilovaním a program je potom skompilovaný s konkrétnou hodnotou konštanty. Pri zmene hodnoty konštanty je potrebné celý program prekompilovať. Trieda je odvodená od triedy CLVariable a obsahuje iba názov konštanty a jej hodnotu.

Pomocná časť

Pomocná časť frameworku zahŕňa pomocné triedy, ktoré frameworku poskytujú dodatočnú funkcionalitu. Sú to tieto triedy:

Trieda Log

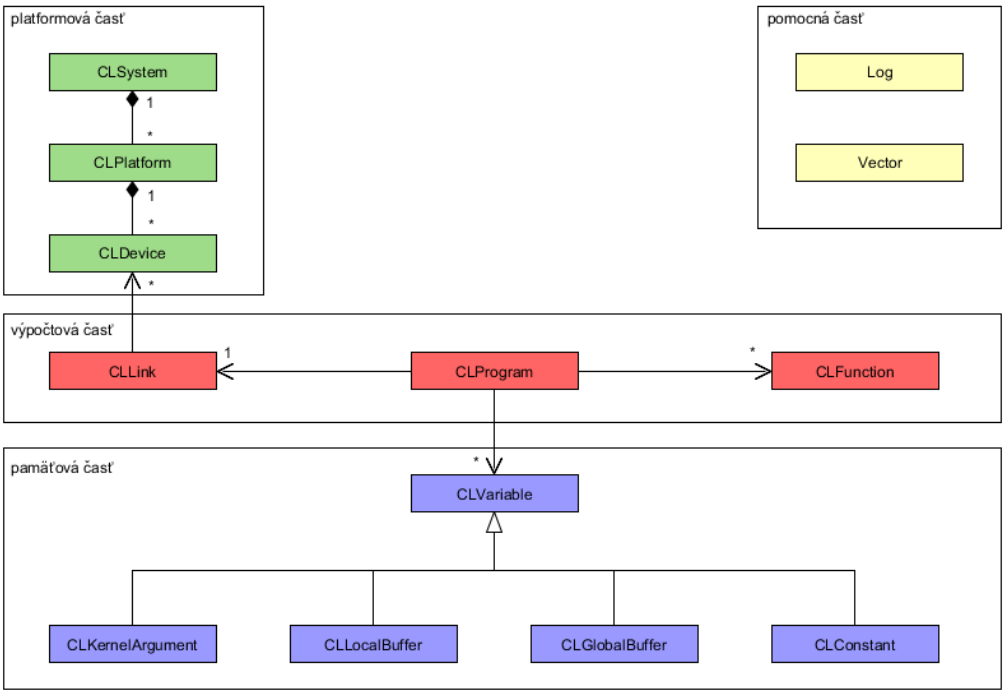
Trieda Log umožňuje vypisovanie kontrolných výpisov frameworku, výsledkov volania jednotlivých OpenCL funkcií a aj vypisovanie chýb, ktoré môžu nastať pri volaní týchto funkcií. Výpisy sa zapisujú do konzoly a aj do textového súboru. Výpisy je možné zrušiť pomocou príslušnej metódy triedy. Trieda je implementovaná ako singleton, aby k nej mala každá trieda v rámci frameworku prístup a mohla volať jej metódy.

Trieda Vector

Trieda Vector slúži ako pomocná trieda pre pamäťovú časť frameworku. Je to šablónová trieda predstavujúca vektor dát a umožňuje správu dát pre triedu CLVariable ako aj ich nastavovanie a získavanie.

3.5.2 Diagram tried

Celkovú štruktúru a architektúru OpenCL frameworku najlepšie vystihuje Obr. 3.2, ktorý znázorňuje diagram tried vo frameworku a to ako môžu medzi sebou tieto triedy komunikovať.



Obr. 3.2: Diagram tried OpenCL frameworku.

Kapitola 4

Implementácia

V tejto kapitole pojednávame o niektorých technických detailoch našej implementácie, zaoberáme sa priebehom SPH simulácie v programovom kóde, štruktúrou projektu a uvádzame aj pseudokódy z implementácie.

4.1 Štruktúra projektu

Zdrojové súbory projektu sme rozdelili do šiestich kategórií podľa funkcionality (zdrojové súbory jednotlivých kategórií sú umiestnené v samostatných adresároch):

- *OpenCL framework* - zdrojové súbory a triedy OpenCL frameworku, ktorý nám umožňuje spúšťať výpočty na OpenCL zariadeniach.
- *OpenGL system* - v tejto kategórii sú zdrojové súbory pre vizualizačnú časť, ktorá spravuje kamery, svetlá, materiály a GLSL shadre a vykresľuje scénu do okna pomocou OpenGL.
- *Window system* - sú zdrojové súbory pre vytváranie okna, nastavovanie OpenGL kontextu, odchytávanie užívateľských vstupov z klávesnice a

myši pomocou rozhrania WinAPI.

- *SPH* - zdrojové súbory pre nastavovanie parametrov simulácie, načítavanie scén, načítanie kernelov pre simuláciu a ich spúšťanie na OpenCL zariadeniach.

- *Súbory aplikácie* - sú zdrojové súbory na načítavanie nastavení aplikácie zo súboru, vytváranie programových výpisov a súbor s funkciou main.

- *OpenCL kernely* - zdrojové súbory pre kernel funkcie, ktoré vykonávajú výpočty pre SPH simuláciu na OpenCL zariadeniach.

4.2 Použitie OpenCL frameworku

Celá simulačná časť našej implementácie využíva OpenCL framework na vytváranie a spúšťanie SPH simulácie. V tejto podkapitole vysvetlíme ako používame framework na realizovanie simulácie. Základnými bunkami sú OpenCL kernely, ktoré spúšťame na OpenCL zariadeniach a premenné, ktoré slúžia ako vstupné argumenty pre tieto kernely a do ktorých sú ukladané výsledky výpočtov.

Ešte pred tým než budeme môcť spúšťať kernely na OpenCL zariadení, musíme si najprv inicializovať triedu CLSystem. Táto trieda je implementovaná ako singleton a pri volaní jej metódy *getInstance()* sa OpenCL framework nainicializuje - nájdu sa v počítači všetky dostupné OpenCL platformy a všetky dostupné OpenCL zariadenia v rámci týchto platforiem. Keď máme framework inicializovaný, vyberieme niektorú z dostupných platforiem a niektoré konkrétne zariadenie (napríklad GPU) a vytvoríme tvz. pripojenie na toto zariadenie. Pripojenie na zariadenie sa realizuje pomocou triedy CLLink, ktorá do konštruktora dostane inštanciu triedy CLDevice, ktorá predstavuje

zariadenie, na ktoré sa chceme pripojiť.

Keď máme prichystané pripojenie na zariadenie, môžeme začať vytvárať OpenCL kernely, na čo slúži trieda CLProgram. Trieda CLProgram dostane ako parametre do konštruktora cestu k súboru, z ktorého sa má kernel načítať, názov kernelu a inštanciu triedy CLLink na zariadenie, na ktorom má byť kernel skompilovaný a potom aj vykonávaný. Ešte je možné zo súboru načítať aj funkciu, ktorá bude pridaná do kernelu a kernel ju môže pri vykonávaní svojho kódu volať. Na toto slúži trieda CLProgram.

Ďalej potrebujeme vytvoriť premenné, ktoré slúžia ako vstupné argumenty pre kernely, ktoré sme vytvorili v predchádzajúcom kroku. Najčastejšie sme používali triedu CLGlobalBuffer, cez ktorú sme vytvárali polia pre častice, napr. pre ich pozície v priestore. Používali sme aj CLKernelArgument, cez ktorý sme ovládali funkcionalitu samotného kernelu - či má alebo nemá niektorú časť svojho kódu vykonať alebo nie. Používali sme aj triedu CLLocalBuffer, ale iba v prípadoch, kedy sme výsledky výpočtov nepotrebovali vrátiť, pretože tento buffer sa nachádza iba v lokálnej pamäti OpenCL zariadenia a obsah tejto pamäti sa nedá na host zariadení získať. Pomocou triedy CLConstant sme nastavili niektoré parametre pre simuláciu ako konštanty, napríklad časový krok simulácie, polomer nosiča vyhladzovacej funkcie, gravitačné zrýchlenie atď. Tieto konštanty sú pridané priamo do zdrojového kódu kernelu a po skompilovaní kernelu už tieto konštanty nie je možné meniť - po zmene konštanty je potrebné znovu skompilovať celý kernel.

Po vytvorení kernelov, funkcií, premenných a konštánt ich môžeme pridať do samotných kernelov. Trieda CLProgram má metódy, ktoré to umožňujú. Keď sme všetko pre kernel úspešne nastavili môžeme tento kernel skompilovať. Po úspešnom skompilovaní kernelu môžeme dať vykonať jeho kód na OpenCL zariadení a stiahnuť výsledky výpočtov z pamäte OpenCL zaria-

denia do pamäte host zariadenia pomocou premenných, ktoré sme nastavili inštanciami triedy CLGlobalBuffer. Keď máme hodnoty už uložené v pamäti host zariadenia, môžeme ich napríklad vypísať na štandardný výstup aplikácie alebo ďalej spracovať iným spôsobom.

Použitie frameworku je jednoduché a je možné vďaka nemu rýchlo vytvárať kernely a spúšťať ich na OpenCL zariadeniach. Preto bolo pre nás dôležité tento framework vytvoriť a použiť, aby sme tak mohli rýchlo vyvíjať kód pre simuláciu SPH. Po ukončení práce s frameworkom treba na všetky inštancie tried, ktoré sme alokovali pomocou operátora *new* zavolať operátor *delete*, ktorý zavolá deštruktor triedy a správne uvoľní pamäť a všetky zdroje, ktoré daná inštancia využívala, či už na OpenCL zariadení, alebo na host zariadení. Všetky kroky pre vytvorenie a spustenie programu sú zhrnuté v pseudokóde 4.1.

Algoritmus 4.1 Pseudokód - použitie OpenCL frameworku

inicializuj CLSystem

získaj platformu z CLSystem

získaj OpenCL zariadenie z platformy

načítaj kernel a nastav pripojenie na zariadenie

vytvor vstupné parametre pre kernel

nastav kernelu vstupné parametre

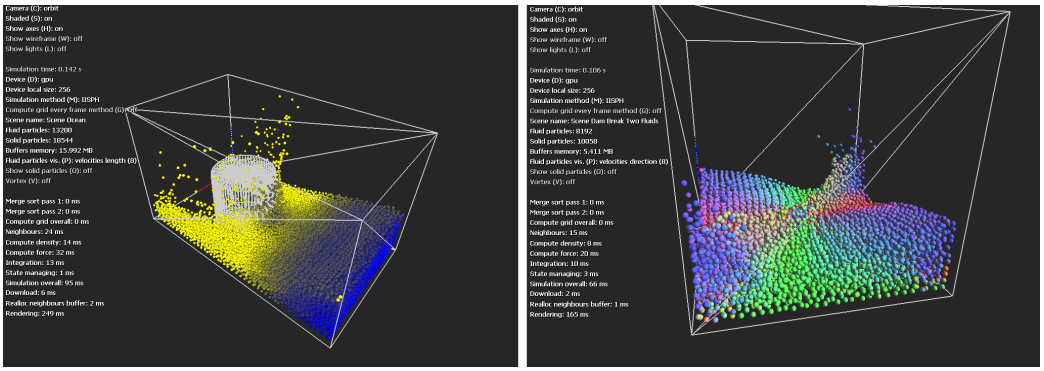
skompiluj kernel

spusti kernel

získaj vypočítané dáta

spracuj dáta

uvoľni kernel a vstupné parametre



Obr. 4.1: Vľavo scéna s 13200 časticami kvapaliny a tuhým objektom v strede (čím má častica farbu viac do žlta tým má vyššiu rýchlosť), vpravo scéna s dvoma zrážajúcimi sa kvapalinami (ofarbenie častíc predstavuje rýchlosť v jednotlivých osiach).

4.3 Simulačná časť

Simulačná časť je najdôležitejšou časťou našej implementácie. V tejto podkapitole podrobnejšie rozoberáme jej implementáciu. Uvádzame ako sme implementovali tri rôzne SPH metódy na simuláciu kvapalín. Prikladáme aj pseudokódy, ktoré sú vykonávané na host zariadení a aj pseudokódy vykonávané na OpenCL zariadeniach.

4.3.1 SPH metódy

V našom projekte sme implementovali a paralelizovali tri metódy SPH. Metódy sú plne simulované na OpenCL zariadení, čiže všetky výpočty simulácie sa vykonávajú na iba tomto zariadení. Z host zariadenia sa tieto výpočty iba spúšťajú - host zariadenie nastavuje premenné a spúšťa kernely. Metódy, ktoré sme implementovali:

- *Slabo stlačiteľné SPH*, angl. WCSPH (Weakly compressible SPH).

- *Prediktívno-korektívne nestlačiteľné SPH*, angl. PCISPH (Predictive-corrective incompressible SPH).
- *Implicitne nestlačiteľné SPH*, angl. IISPH (Implicit incompressible SPH)

Na ďalších riadkoch si popíšeme hlavný cyklus vykonávania výpočtov jednotlivých metód formou pseudokódu. Pre zjednodušenie každý riadok predstavuje volanie jedného kernelu na OpenCL zariadení a výpočet sa presunie na ďalší riadok až keď sa dokončí výpočet na predchádzajúcom riadku pre všetky častice. Paralelizáciu výpočtov sme uskutočnili tak, že jednotlivé kernely sa pre každú časticu vykonávajú paralelne. Keď má napríklad OpenCL zariadenie 256 samostatných výpočtových jednotiek, tak sa paralelne v každom kroku naraz vykonáva 256 kernelov pre 256 častíc. Niektoré výpočty sa ale takýmto spôsobom paralelizovať nedali, a preto sme sa pri nich snažili dosiahnuť čo najväčšiu mieru paralelizácie - spúšťať čo najväčšie množstvo kernelov paralelne.

V našej implementácii môže byť kvapalina ovplyvňovaná tuhými telesami, pričom tuhé telesá kvapalinou ovplyvňované nie sú. Tuhé telesá je možné načítať zo súborov .obj a pridať ich do scény. Počas pridávania do scény sú tuhé telesá navzorkované časticami, ktoré sa potom využívajú pri výpočte síl častíc kvapaliny, ktorej tok je potom týmito časticami ovplyvňovaný. Ešte pred spustením samotnej simulácie je potrebné vytvoriť mriežku buniek priestoru, v ktorom sa častice nachádzajú a zaradiť jednotlivé častice do týchto buniek, potom vypočítať koľko susedov má každá častica a nájsť časticu, ktorá má najväčší počet susedov. Podľa tohto počtu častíc je potom alokovaná pamäť pre zoznam susedných častíc. Keď je napríklad najväčší počet susedov 30, tak sa toto pole alokuje na veľkosť *30 * 1.2 * počet častíc v systéme*. Toto sa nastaví na začiatku simulácie, no aj počas výpočtu sa každý desiaty krok hľadá častica s najväčším počtom susedov a zoznam susedných častíc sa

znova realokuje podľa daného vzorca, aby sme mali vždy dostatočný priestor pre všetkých susedov. S tým, že realokácia sa deje iba vtedy, keď je najväčší počet susedných častíc o 10% väčší ako pri poslednej kontrole. Pri našom testovaní bol priemerný počet susedov pre častice kvapaliny a kvapaliny 30 a počas simulácie sa tento počet nemenil, čiže realokácia sa vykonala iba na začiatku a počas simulácie sa už nevykonávala. V našej implementácii sa hľadajú susedia pre častice kvapaliny a kvapaliny, častice kvapaliny a tuhých telies, tuhých telies a tuhých telies. Máme teda tri zoznamy susedov pre jednotlivé interagujúce typy častíc.

Susedia pre častice tuhých telies sa hľadajú iba na začiatku simulácie, aby mohli byť vypočítané sily pre jednotlivé častice tuhých telies. Keďže kvapalina tuhé telesá neovplyvňuje, ich pozície sa nemenia a sily pôsobiace na častice tuhých telies medzi sebou sa tiež nemenia, preto ich netreba prepočítavať počas simulácie. Implementáciu samotného hľadania susedov si popíšeme v ďalšej kapitole. Pseudokód pre inicializáciu SPH simulácie, ktorý je vykonávaný ešte pred samotnou simuláciou (tento kód sa vykonáva pred spustením každej z troch SPH metód):

Algoritmus 4.2 Pseudokód - inicializácia

inicializuj častice
vytvor mriežku pre častice kvapaliny
vytvor mriežku pre častice tuhých telies
realokuj zoznamy susedných častíc
nájdi susedov pre tuhé častice
vypočítaj interakčné sily medzi tuhými časticami a tuhými časticami

Teraz postupne uvádzame pseudokódy pre vykonávanie výpočtov jednotlivých SPH metód:

Algoritmus 4.3 Pseudokód - WCSPH

vytvor mriežku pre častice kvapaliny

nájdi susedov pre častice kvapaliny

vynuluj sily pôsobiace medzi časticami

vyrátaj hustotu pre každú časticu

normalizuj hustotu každej častice

vyrátaj sily pôsobiace na častice

integruj rýchlosti častíc

uprav rýchlosti podľa hustoty častíc

integruj pozície častíc

Algoritmus 4.4 Pseudokód - PCISPH

vytvor mriežku pre častice kvapaliny

nájdi susedov pre častice kvapaliny

vynuluj sily pôsobiace medzi časticami

vyrátaj hustotu pre každú časticu

normalizuj hustotu každej častice

zisti maximálnu fluktuáciu hustoty

vyrátaj sily pôsobiace na častice

while kým nie je dosiahnutá požadovaná maximálna fluktuácia hustoty **do**

integruj pozície častíc

vyrátaj hustotu pre každú časticu

normalizuj hustotu každej častice

zisti maximálnu fluktuáciu hustoty

vyrátaj sily pôsobiace na častice

end while

integruj rýchlosti častíc

uprav rýchlosti podľa hustoty častíc

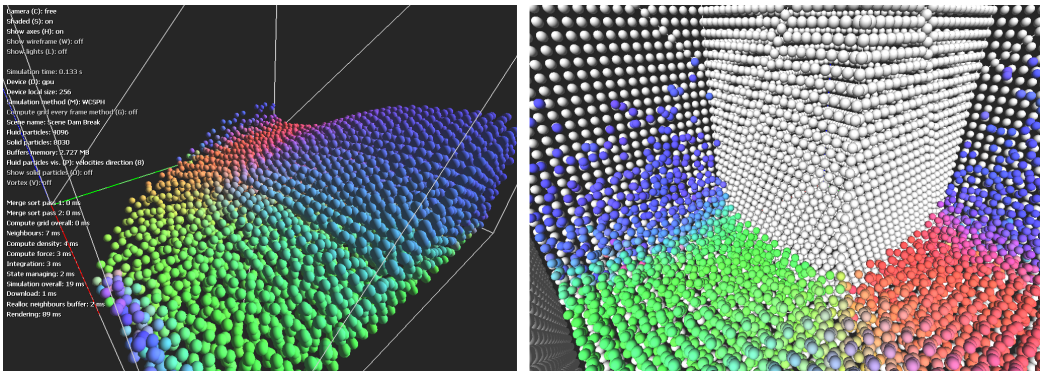
integruj pozície častíc

Algoritmus 4.5 Pseudokód - IISPH

vytvor mriežku pre častice kvapaliny
nájdi susedov pre častice kvapaliny
vynuluj sily pôsobiace medzi časticami
vyrátaj hustotu pre každú časticu
normalizuj hustotu každej častice
zisti priemernú fluktuáciu hustoty
vyrátaj sily pôsobiace na častice
predikuj rýchlosti jednotlivých častíc
predikuj hustotu jednotlivých častíc
while kým nie je dosiahnutá požadovaná priemerná fluktuácia hustoty **do**
    vyrátaj hustotu a sily pôsobiace na častice
    zisti priemernú fluktuáciu hustoty
end while
vyrátaj sily pôsobiace na častice
integruj rýchlosti častíc
uprav rýchlosti podľa hustoty častíc
integruj pozície častíc

4.3.2 Hľadanie k-najbližších susedov

Hľadanie k-najbližších susedov sme implementovali metódou popísanou v Kapitole 2 v sekcii 2.3.1 Z-Indexing a triedenie. Pre porovnanie rýchlosti sme implementovali aj neefektívne hľadanie susedov, ktoré má zložitosť $O(n^2)$. Táto metóda využíva viacero polí na OpenCL zariadení. Metóda najprv paralelne pre každú časticu vypočíta jej Z-Index z jej pozície a uloží tento index spolu aj s indexom do poľa častíc do poľa, ktoré bude predstavovať mriežku buniek, v ktorých sa častice nachádzajú. Ďalej je toto pole paralelne utrie-



Obr. 4.2: Vľavo scéna so 4096 časticami kvapaliny, bez vykreslených častíc tuhých objektov, vpravo scéna tiež so 4096 časticami kvapaliny spolu aj s vykreslenými časticami tuhých objektov.

dené na OpenCL zariadení pomocou algoritmu Triedenie zlučovaním podľa Z-Indexu. Potom sa paralelne vytvorí pole, pomocou ktorého môžeme indexovať jednotlivé bunky cez Z-Index. Toto pole musíme realokovať na takú veľkosť, aby sme mohli priamo indexovať každú bunku. Keď napríklad máme bunku s najväčším Z-Indexom 13842, tak musí mať toto pole práve takúto veľkosť plus jedna. Keď ideme potom hľadať susedov pre nejakú časticu i a z jej pozície vyrátame práve Z-Index 13842, pozrieme sa do tohto poľa na tento index a číslo na tomto indexe predstavuje index do mriežky buniek na prvú časticu nachádzajúcu sa v tejto bunke. Potom nájdeme všetkých susedov danej častice do vzdialenosti maximálne h a skontrolujeme ešte všetkých okolitých susediacich 26 buniek.

Ďalší z prístupov, ktorý sme ešte vyskúšali bolo prepočítavať túto mriežku, teda prepočítavať, do ktorých buniek častice patria každý 10 krok simulácie. Prepočítavanie mriežky je totiž časovo veľmi náročné kvôli triedeniu poľa so Z-Indexami. Týmto krokom sa simulácia zrýchlila s tým, že každých 10 krokov, na chvíľu "trhla". Keď však prepočítavame mriežku iba každý 10

krok simulácie, nemáme aktuálne údaje o tom, v ktorých bunkách sa častice nachádzajú. Môže teda nastať situácia, že pre časticu nenájdeme všetkých jej susedov. Nemôže však ale nastať situácia, že by sme pre časticu našli susedov, ktorí v skutočnosti nie sú susedia tejto častice. Vyplýva to z toho, že počas hľadania susedov sa kontroluje vzdialenosť medzi obidvoma časticami.

Pripájame ešte aj krátky pseudokód na hľadanie susedov:

Algoritmus 4.6 Pseudokód - Hľadanie k-najbližších susedov

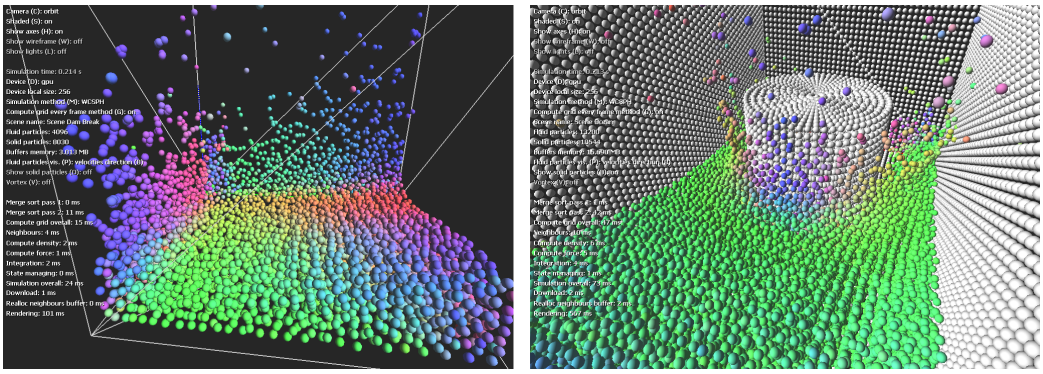
vytvor bunky v mriežke pre častice

utrieď bunky podľa Z-Indexu

vytvor indexy buniek

nájdi susedov pre častice

Implementácia neefektívneho hľadania susedov bola priamočiara, kedy sa pre každú časticu prechádza celý zoznam častíc a hľadajú sa všetky častice do vzdialenosti maximálne h.



Obr. 4.3: Vľavo obrázok zo scény s 4096 časticami kvapaliny. Vpravo obrázok zo scény s 13200 časticami kvapaliny spolu aj s vykreslenými časticami tuhých telies.

Kapitola 5

Výsledky

V našej práci sme úspešne paralelizovali pomocou OpenCL tri SPH metódy: Slabo stlačiteľné SPH, Prediktívno-korektívne nestlačiteľné SPH a Implicitne nestlačiteľné SPH. V tejto kapitole porovnávame rýchlosť našej implementácie pre tieto tri SPH metódy na troch rôznych scénach a s troma prístupmi k hľadaniu k-najbližších susedov.

Na testovanie rýchlosti simulácie sme použili notebook HP Pavilion g6: Intel Core i3 380M @ 2,53Ghz, 8 GB DDR3 RAM a AMD Radeon HD 6470M s 512MB RAM. Grafická karta má výkon 224 GFLOPS. Testovali sme tak, že simuláciu sme spustili do 95. kroku a v ňom sme zaznamenali všetky údaje o rýchlosti okrem FPS, ktoré sme merali a zobrali priemerné.

Robili sme tri testy pre rôzne metódy hľadania susedov. Najprv sme testovali simuláciu s tým, že sa mriežka buniek, v ktorých sa nachádzali častice prerátavala každých desať krokov, z čoho sme zaznamenali priemerné FPS simulácie. Dĺžka vytvárania mriežky je uvedená v stĺpci *Mriežka* a priemerné FPS v stĺpci *FPS 1*. Ďalej sme testovali simuláciu tak, že sa mriežka buniek prerátavala v každom kroku, z čoho sme zaznamenali tiež priemerné FPS uvedené v stĺpci *FPS 2*. Stĺpec *Susedia* predstavuje dĺžku trvania hľa-

dania k-najbližších susedov pre každú časticu, čo sa robilo v každom kroku simulácie. Nakoniec sme simuláciu testovali tak, že sme na hľadanie susedov použili neefektívny algoritmus so zložitosťou $O(n^2)$ popísaný v Kapitole 2 v sekcii 2.3.3 Brute-force metóda, pri použití tohto algoritmu nebolo potrebné vytvárať mriežku buniek. Priemerné FPS simulácie s týmto spôsobom hľadania susedov uvádzame v stĺpci *FPS 3*.

Simuláciu sme testovali na troch rôznych scénach s rôznymi počtami častíc kvapaliny a tuhých telies a pre každú scénu sme simuláciu testovali na GPU aj na CPU na vyššie spomenutom počítači.

Postupne uvádzame výsledky jednotlivých testov:

Tabuľka 5.1: Scéna 01, 4096 častíc kvapaliny, 8030 častíc tuhých telies, GPU

	Mriežka	Susedia	Susedia $O(n^2)$	Simulácia	FPS 1	FPS 2	FPS 3
WCSPH	17ms	5ms	44ms	11ms	46	29	18
PCISPH	15ms	5ms	43ms	68ms	15	13	10
IISPH	18ms	5ms	42ms	22ms	28	20	15

Tabuľka 5.2: Scéna 01, 4096 častíc kvapaliny, 8030 častíc tuhých telies, CPU

	Mriežka	Susedia	Susedia $O(n^2)$	Simulácia	FPS 1	FPS 2	FPS 3
WCSPH	13ms	7ms	131ms	12ms	64	33	8
PCISPH	12ms	3ms	111ms	55ms	15	13	7
IISPH	14ms	4ms	140ms	24ms	36	24	7

Tabuľka 5.3: Scéna 02, 13200 častíc kvapaliny, 18544 častíc tuhých telies, GPU

	Mriežka	Susedia	Susedia $O(n^2)$	Simulácia	FPS 1	FPS 2	FPS 3
WCSPH	50ms	17ms	363ms	26ms	19	11	3
PCISPH	50ms	17ms	364ms	192ms	5	4	2
IISPH	51ms	18ms	364ms	65ms	11	7	3

Tabuľka 5.4: Scéna 02, 13200 častíc kvapaliny, 18544 častíc tuhých telies, CPU

	Mriežka	Susedia	Susedia $O(n^2)$	Simulácia	FPS 1	FPS 2	FPS 3
WCSPH	50ms	16ms	1062ms	31ms	24	11	1
PCISPH	48ms	15ms	988ms	158ms	6	5	1
IISPH	54ms	16ms	986ms	84ms	7	6	1

Tabuľka 5.5: Scéna 03, 167500 častíc kvapaliny, 90130 častíc tuhých telies, GPU

	Mriežka	Susedia	Susedia $O(n^2)$	Simulácia	FPS 1	FPS 2	FPS 3
WCSPH	680ms	207ms	-	302ms	2	1	-
PCISPH	679ms	200ms	-	1892ms	$\frac{1}{2}$	$\frac{1}{3}$	-
IISPH	678ms	203ms	-	549ms	1	$\frac{1}{2}$	-

Tabuľka 5.6: Scéna 03, 167500 častíc kvapaliny, 90130 častíc tuhých telies, CPU

	Mriežka	Susedia	Susedia $O(n^2)$	Simulácia	FPS 1	FPS 2	FPS 3
WCSPH	593ms	207ms	-	277ms	2	1	-
PCISPH	539ms	200ms	-	1633ms	$\frac{1}{2}$	$\frac{1}{3}$	-
IISPH	590ms	203ms	-	563ms	1	$\frac{1}{2}$	-

Z testov je vidno, že veľký rozdiel vo výkone medzi GPU a CPU nie je, rozdiel je skoro zanedbateľný aj pri vysokých počtoch častíc aj pri nízkych. Keď sme pozerali ako simulácia vyťažuje grafický hardvér, tak to bolo na úrovni 90%, čo znamená, že simulácia je maximálne paralelizovaná. Čiže problém v zlej implementácii paralelizmu nie je, problém bude vo výkone GPU.

Prepočítavanie mriežky buniek trvá v niektorých prípadoch $\frac{1}{2}$ až $\frac{1}{3}$ celého simulačného času. V tomto vidíme ešte veľký priestor na zlepšenie, pretože 90% času z dĺžky trvania prepočítavania mriežky je triedenie Z-Indexov. V

našej implementácii sme použili paralelné triedenie zlučovaním, no existujú aj efektívnejšie triedenia ako Radix sort alebo Bitonic sort, ich efektívnou paralelnou implementáciou by sa celá simulácia mohla ešte zrýchliť.

Pretože GPU nebolo dostatočne výkonné testovali sme našu implementáciu na oveľa výkonnejšej GPU - AMD Radeon R9 280X s 3GB RAM a výkonom 3482 GFLOPS. Test sme uskutočnili tak ako predchádzajúce testy, s tým, že sme testovali iba na tretej scéne a vynechali sme testovanie neefektívneho hľadania susedov. Výsledky sme zhrnuli do nasledujúcej tabuľky:

Tabuľka 5.7: Scéna 03, 167500 častíc kvapaliny, 90130 častíc tuhých telies, GPU

	Mriežka	Susedia	Susedia $O(n^2)$	Simulácia	FPS 1	FPS 2	FPS 3
WCSPH	262ms	7ms	-	14ms	21	4	-
PCISPH	263ms	7ms	-	107ms	9	3	-
IISPH	262ms	8ms	-	55ms	11	3	-

Z tabuľky sa dá vyčítať, že najpomalšie na celom výpočte bolo vytváranie mriežky a v ňom ako sme vyššie spomínali tvorí väčšinu času triedenie. Efektívnejším triedením by sme mohli simulovať v reálnom čase aj milióny častíc. Pri metóde WCSPH tvorba mriežky tvorila až 92% času, pri metóde PCISPH to bolo 70% času. Najväčší spomaľovač našej implementácie je teda práve triedenie kvôli tvorbe mriežky a triedeniu v každom kroku simulácie, preto je vidno veľký skok pri údajoch v stĺpcoch *FPS 1* a *FPS 2*.

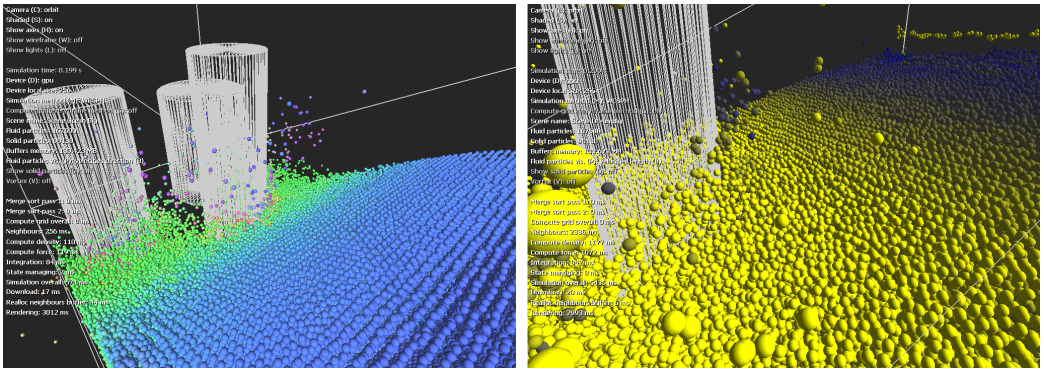
Počas testovania sme zistili, že v niektorých prípadoch nebola simulácia stabilná, keď sa mriežka prepočítavala každých desať krokov simulácie.

Pre porovnanie uvádzame ešte test implementácie SPH na CPU. Testovali sme na scéne, ktorá má rovnaké parametre ako scéna 01 z našej implementácie na počítači uvedenom vyššie s CPU Intel Core i3 380M @ 2,53Ghz a 8 GB DDR3 RAM.

Tabuľka 5.8: CPU implementácia, 4096 častíc kvapaliny, 8030 častíc tuhých telies

	Susedia	Simulácia	FPS
WCSPH	12ms	7ms	42
PCISPH	11ms	34ms	23
IISPH	11ms	21ms	26

Táto implementácia používa metódu na hľadanie k-najbližších susedov popísanú v Kapitole 2 v sekcii 2.3.2 Indexovanie pomocou smerníkov. Ako z tabuľky vidno toto hľadanie susedov je efektívnejšie ako hľadanie susedov použité v našej implementácii a priemerné FPS je vyššie. Naša implementácia sa tejto v rýchlosti vyrovná, keď sa mriežka buniek prepočítava každý desiaty krok simulácie. Táto implementácia je však sekvenčná, nevyužíva paralelizmus, tak je potom otázne ako dobre táto implementácia škáluje s vyšším počtom častíc. Bohužiaľ testy na vyššie počty častíc nemáme, ale pravdepodobne naša implementácia by mala byť efektívnejšia s vyšším počtom častíc na danom hardvéri.



Obr. 5.1: Scéna s 167500 časticami kvapaliny a 90130 časticami tuhých telies.

Kapitola 6

Záver

Cieľom tejto práce bolo implementovať a paralelizovať metódu simulácie toku kvapalín SPH. Implementovali a paralelizovali sme úspešne tri metódy SPH - Slabo stlačiteľné SPH, Prediktívno-korektívne nestlačiteľné SPH a Implicitne nestlačiteľné SPH. Všetky tri metódy sme implementovali pomocou knižnice OpenCL a všetky výpočty simulácie sme presunuli na OpenCL zariadenie, či už GPU, alebo CPU. Na OpenCL zariadenie sme takisto presunuli aj všetky dátové štruktúry potrebné pre simuláciu. Na host zariadení sa vykonáva iba vykresľovanie scény a volanie jednotlivých kernelov simulácie, ktoré sú potom vykonávané na OpenCL zariadení. Implementovali sme aj efektívne paralelné hľadanie k-najbližších susedov pomocou Z-Indexingu a triedenia. Vo výsledkoch práce sme uviedli testovanie rýchlosti našej implementácie na konkrétnom hardvéri. Počas testovania sme zistili, že algoritmus, ktorý sme použili na triedenie pri prehľadávaní susedov najviac spomaľuje celú simuláciu. Ďalším možným vylepšením a zefektívnením simulácie by bolo implementovať efektívnejší algoritmus triedenia, napríklad paralelný Radix sort alebo Bitonic sort. Do implementácie by sa mohla ešte pridať aj možnosť pridávať častice kvapaliny do simulácie počas jej behu, čo teraz nie je možné.

Dodatok A

Slovník

API – angl. Application programming interface, rozhranie pre programovanie aplikácií. Je to zbierka procedúr, funkcií a tried knižnice, ktoré môže programátor používať vo svojom zdrojovom kóde.

Brute-force – angl. hrubou silou, takto sa označujú neefektívne algoritmy, ktoré prehľadávajú celý stavový priestor a nepoužívajú žiadne heuristiky.

Buffer – pamäť, ktorá dočasne uchováva vstupné alebo výstupné dáta počas ich prenosu.

CPU – angl. Central processing unit, procesor v počítači.

Data-based paralelizmus – typ paralelizmu, ktorý sa zameriava na distribúciu dát na všetky výpočtové paralelné jednotky.

DSP – angl. Digital signal procesor, špeciálny typ procesora slúžiaci na spracovávanie digitálneho signálu.

FPGA – angl. Field-programmable gate array, konfigurovateľný integrovaný obvod.

Framework – angl. je to softvérový rámec, ktorý poskytuje určitú všeobecnú funkcionalitu a slúži ako podpora pri vývoji softvéru. Môže obsahovať podporné programy, knižnice API.

GPU – angl. Graphics processing unit, procesor grafickej karty.

Kernel – angl. jadro. V našej práci používame jeho dva významy. Prvý v SPH simulácii pre vyhladzovacie kernel funkcie a druhý v OpenCL, kde kernel predstavuje vstupnú funkciu výpočtu na OpenCL zariadení.

Profiling – angl. profilovanie, je forma dynamickej programovej analýzy, ktorá napríklad meria množstvo využitej pamäti, časovú zložitosť programu, využívanie určitých inštrukcií alebo frekvenciu a dĺžku trvania volania funkcií.

RAM – angl. Random-access memory, je to pamäť umožňujúca ľubovoľný (náhodný) prístup. Čas zápisu do pamäte je rovnaký bez ohľadu na umiestnenie údaja v pamäti.

Run-time – je doba počas ktorej je počítačový program spustený a vykonávaný.

SIMD – angl. Single instruction multiple data, je to trieda paralelných počítačov, ktoré majú viacero výpočtových jednotiek, ktoré paralelne vykonávajú rovnakú operáciu na rôznych dátach.

Singleton – angl. jedináčik. Je to návrhový vzor, pri ktorom môže naraz existovať iba jedna statická inštancia danej triedy.

SPMD – angl. Single program multiple data, je to trieda paralelných počítačov, ktoré rozdeľujú úlohy a paralelne ich spúšťajú na viacerých procesoroch s rozdielnymi vstupmi, aby rýchlejšie získali výsledky.

Task-based paralelizmus – typ paralelizmu, ktorý sa zameriava na distribúciu výpočtových procesov na všetky paralelné výpočtové jednotky.

Vložený systém – po angl. embedded system, je elektronický systém, ktorý je súčasťou väčšieho a komplexnejšieho systému, pre ktorý zabezpečuje riadenie definovanej sady funkcií.

Dodatok B

Manuál k programu

Program do súboru *oclsphlog.txt* ukladá výpisy priebehu niektorých funkcií programu. Do súboru *config.txt* ukladajú nastavenia programu a po spustení programu sa vždy načítajú. Tieto nastavenia je možné meniť ručne otvorením tohto súboru a prepísaním hodnôt.

Ovládanie

Program má tri kamery. Medzi kamerami je možné prepínať klávesom *C*. Ovládanie kamery *Orbit* a *Free orbit* je rovnaké - ľavým tlačidlom myši sa kamera posúva v rovine okna, pravým tlačidlom myši sa kamera rotuje a skrolovaním so stredným tlačidlom sa kamera približuje alebo vzďaľuje. Treťou kamerou je *Free camera*, ktorá sa rotuje pohybom myši v okne, šípkami na klávesnici sa kamera pohybuje v scéne, klávesom *Shift* sa kamera hýbe smerom hore v rovine okna a klávesom *Ctrl* sa hýbe smerom dole v rovine okna.

- *ESC* - ukončenie aplikácie
- *F11* - prepínanie medzi fullscreen a oknovým módom

- *Medzerník* - spúšťanie/zastavovanie simulácie
- *kláves R* - resetovanie scény
- *kláves F* - zobrazenie FPS
- *kláves I* - zobrazenie informačných výpisov
- *kláves S* - zobrazenie častíc ako guličiek alebo kruhov
- *kláves H* - zobrazenie ôs x, y a z
- *kláves W* - zobrazenie mriežkovanej siete pre objekty - platí iba keď sú častice zobrazované ako guličky
- *kláves L* - zobrazenie svetiel v scéne
- *kláves D* - prepínanie medzi OpenCL zariadeniami - CPU a GPU
- *kláves M* - prepínanie medzi simulačnými metódami WCSPH, PCISPH a IISPH
- *kláves G* - prepínanie, či sa má mriežka buniek prepočítavať každý krok simulácie, alebo každých 10 krokov simulácie
- *kláves P* - vizualizácia častíc, buď pevná modrá farba, alebo vizualizácia rýchlosti, alebo vizualizácia rýchlosti v jednotlivých osiach
- *kláves O* - zobrazenie častíc tuhých telies
- *kláves V* - zapnutie/vypnutie víru v strede scény
- *klávesy / ** - zmenšenie/zväčšenie častíc pri zobrazovaní ako kruhov
- *klávesy Page down a Page up* - načítanie predchádzajúcej, alebo ďalšej scény

Dodatok C

Popis obsahu CD

Zdrojové kódy v C++ sú pridané do projektu pre Microsoft Visual Studio 2013. Pre skompilovanie je potrebné mať nainštalovanú túto verziu Visual Studia (môže byť aj Expres, Profesional, Premium alebo Ultimate edícia). Ďalej je potrebné mať nainštalované správne OpenCL SDK pre grafickú kartu (AMD, NVIDIA) alebo procesor (AMD, Intel), podľa toho na akom zariadení chcete simuláciu spúšťať. Keď máte nainštalované OpenCL SDK je potrebné v projekte vo Visual Studiu 2013 nastaviť správne cesty k .lib a .h súborom OpenCL, tieto cesty ale závisia od konkrétneho zariadenia a OpenCL SDK. Na spustenie .exe súboru programu by mali stačiť nainštalované najnovšie ovládače na grafickú kartu alebo procesor, ktoré do systému pridajú potrebné .dll knižnice pre OpenCL. Ďalej na spustenie .exe súboru pokiaľ na počítači nie je nainštalované Microsoft Visual Studio 2013 je potrené nainštalovať redistributable package pre Visual Studio 2013, konkrétne x86 verziu. Tento redistributable package sa nachádza v adresári Redist. Ovládanie rozhrania programu je popísané v súbore readme.txt.

CD obsahuje tieto adresáre a súbory:

- **bin** - obsahuje spustiteľný .exe súbor aplikácie, potrebné .dll knižnice

a súbory pre aplikáciu.

- **redist** - obsahuje redistributable package pre Visual Studio 2013, ktorý je potrebný pokiaľ na počítači chcete iba spustiť .exe súbor aplikácie a nie je nainštalované Visual Studio 2013.
- **src** - obsahuje zdrojové kódy a projekt pre Microsoft Visual Studio 2013.
- **dp_final.pdf** - text diplomovej práce v pdf verzii.
- **readme.txt** - popis ovládania rozhrania programu.

Literatúra

[Bat67] G.K. Batchelor. *An Introduction to Fluid Mechanics*. Cambridge University Press, 1967.

[BT07] Markus Becker and Matthias Teschner. Weakly compressible sph for free surface flows. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 209–217, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[DG96] Mathieu Desbrun and Marie-Paule Gascuel. Smoothed particles: a new paradigm for animating highly deformable bodies. In *Proceedings of the Eurographics workshop on Computer animation and simulation '96*, pages 61–76, New York, NY, USA, 1996. Springer-Verlag New York, Inc.

[FAW10] Roland Fraedrich, Stefan Auer, and Rudiger Westermann. Efficient high-quality volume rendering of sph data. *IEEE Transactions on Visualization and Computer Graphics*, 16:1533–1540, November 2010.

[Gre10] Simon Green. Particle simulation using cuda. *NVIDIA Whitepaper, December 2010*, 2010.

[Gri77] J.J. Gringold, R.A. nad Monaghan. Smoothed particle hydrodynamics - theory and application to non-spherical stars. *Royal Astronomical Society*, 181:375–389, November 1977.

[ICS+14] Markus Ihmsen, Jens Cornelis, Barbara Solenthaler, Christopher Horvath, and Matthias Teschner. Implicit incompressible sph. *IEEE Transactions on Visualization and Computer Graphics*, 20(3):426–435, 2014.

[Isp] Isph - incompressible sph. `http://isph.sourceforge.net`. Accessed: 2014-04-30.

[LL03] G.R. Liu and M.B. Liu. *Smoothed Partice Hydrodynamics a meshfree particle method*. World Scientific Publishing Co. Pte. Ltd., 2003.

[Luc77] L.B. Lucy. A numerical approach to the testing of the fission hypothesis. *Astronomical Journal*, 82:1013–1024, December 1977.

[MCG03] Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 154–159, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[Mon92] J.J. Monaghan. Smoothed particle hydrodynamics. *Annual review of astronomy and astrophysics*, 30:543–574, 1992.

[Ond11] Juraj Onderik. Animation of particle based miscible and immiscible fluids with multiple interfaces, 2011.

[SP09] B. Solenthaler and R. Pajarola. Predictive-corrective incompressible sph. In *SIGGRAPH '09: ACM SIGGRAPH 2009 papers*, pages 1–6, New York, NY, USA, 2009. ACM.