VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SIMULACE TEKUTIN

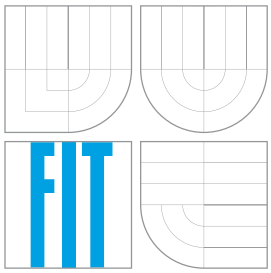
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ ŽIVOTSKÝ

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SIMULACE TEKUTIN

SIMULATION OF FLUIDS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — TOMÁŠ ŽIVOTSKÝ
AUTHOR

VEDOUCÍ PRÁCE — Ing. ZSOLT HORVÁTH
SUPERVISOR

BRNO 2013

Abstrakt

Tato práce se zabývá simulací a následnou vizualizací kapalných látek. Jsou zde popsány různé metody pro výpočet reálného pohybu a chování kapalin. Samotná práce je poté založena na metodách, které dokáží být simulovány v reálném čase.

Abstract

This thesis describes simulation and visualization of fluids by different methods for calculation of real behaviour. The core of this thesis consists of methods working with real-time simulation.

Klíčová slova

simulace, tekutina, CUDA, SPH, částicový systém

Keywords

simulation, fluid, CUDA, SPH, partile system

Citace

Tomáš Životský: Simulace tekutin, bakalářská práce, Brno, FIT VUT v Brně, 2013

Simulace tekutin

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Horvátha

.....
Tomáš Životský
14. května 2013

Obsah

Kapitola 1

Úvod

1.1 Motivace

S rozvojem techniky a odvětví zabývající se výpočetními systémy, získávají naši pozornost úlohy, které nebylo možné dříve řešit. Jednou z takových úloh je simulace reálného světa a jeho přenesení do virtuálního prostoru. I když matematický popis chování tekutých látek je znám již od padesátých let dvacátého století[1], tak samotné simulování přišlo na řadu až mnohem později.

Simulování tekutých látek je ve výpočetní technice vcelku složitý problém. V okolním světě se každodenně setkáváme s kapalinami a jejich chování si dokážeme poměrně přesně představit. Děláme tak však na základě předchozích zkušeností a pozorování. Pokud bychom chtěli předpovědět chování kapaliny v situaci, kterou jsme nemohli nikde pozorovat, bylo by to výrazně obtížnější. Tekutiny jako takové se řídí velkým množstvím pravidel, v závislosti na všech fyzikálních silách, jež na ně působí. Obecně tyto síly působí na každou částici obsaženou v tekutině zvlášť. Z hlediska nároků na výpočetní techniku se tedy jedná o dosti náročné výpočty. Na každou částici tekutiny působí směsice přírodních sil, které ve výsledku dávají silový vektor, po kterém se daná částice bude pohybovat.

Výpočtem pohybu tekutin se zabývá například *CFD (computational fluid dynamics*[2]*)*, což je odvětví oboru mechaniky tekutin, které používá numerické metody a algoritmy pro výpočet fyzikálních vlastností tekutin. Tento přístup využívají různé vědecké či průmyslové organizace, například pro zjištění aerodynamiky automobilů atd. Tyto metody jsou nicméně stejně velice náročné na výpočetní výkon, a proto se jich využívá v této podobě pouze na zjišťování co nejreálnějších výsledků za cenu delší doby výpočtu.

Všechny CFD řešení vychází z Navier – Stokesových rovnic, definující všechny jednofázové pohyby tekutin. Tyto rovnice lze řešit dvěma způsoby. Eulerovými rovnicemi, poté se tekutina chápe jako uskupení buněk v mřížce, nebo Lagrangeovou metodou, čili počítání vektoru síly pro každou částici tekutiny. Různými zjednodušeními můžeme dostat takovou podobu rovnic, které se již dají řešit v reálném čase i na strojích dnes běžně dostupných.

Důvodem, proč se snažit nasimulovat proudění tekutin, mohou být buď již zmíněné výpočty aerodynamiky, hydrodynamiky, využití v herním průmyslu a mnoho dalších.

[1] http://www.navier-stokes.net/
[2] http://www.cfd-online.com/

1.2 Cíl práce

Cílem této práce je simulace tekutin za pomocí CUDA Toolkitu. Hlavním důvodem pro vývoj v tomto prostředí je snadná a efektivní paralelizace a přesun výpočtů na grafickou kartu. Výpočetní výkon je důležitý, neboť simulace by měla probíhat v reálném čase.

1.3 Struktura dokumentu

Technická zpráva se skládá ze dvou částí. V první, teoretické, budou představeny metody pro simulování tekutin a jejich popis. Budou zde představeny jak principy, tak konkrétní rovnice, které popisují fyzikální vlastnosti tekutin. Čtenář se seznámí s pojmy jako *jádro* a *vyhlazovací délka*. Základ práce je založen na Lagrangeově metodě 2.4 a využití metody SPH 2.3.

Praktická část se dá poté rozdělit také na dvě části. První popisuje implementaci aplikace, jména funkcí, proměnných a principy fungování algoritmů a postup výpočtu simulace. Druhá část se zabývá testováním aplikace a návrhy na další pokračování vývoje.

1.4 Současný stav odvětví

Byť je tato problematika relativně čerstvá, současných nástrojů pro simulaci je překvapivě hodně. Mnohé jsou čistě komerční, další slouží pro neziskové účely.

Naprostá většina dnes vyvíjených programů na simulaci kapalin pracuje na bázi SPH. Touto metodou lze totiž velmi rychle a s dostatečnou kvalitou simulovat kapalné, ale i třeba sypké, látky. Diplomová práce Stanislava Kontára [5] popisuje úspěšnou implementaci simulátoru, který dokáže v reálném čase zpracovávat dva tisíce částic.

1.4.1 Blender

Blender je grafický nástroj pro tvorbu animací, obrázků, modelování. Hojně se využívá pro herní a filmový průmysl, nebo právě pro tvorbu simulací. Simulace tekutin jsou, stejně jako obsah této práce, založeny na simulování systému částic.

1.4.2 Glu3D

Jedná se o komerční nástroj, v případě simulace tekutin opět založený na částicích. Tento nástroj však neslouží pro reálné vědecké výpočty. Výsledky práce s tímto softwarem jsou cíleny spíše na tvorbu animací.

1.4.3 LightWave

Tento nástroj je opět určen spíše pro animační a uměleckou tvorbu. Stejně jako předchozí dva nástroje se i tento zabývá modelováním více odvětví než jen kapalin.

1.4.4 RealFlow

Simulátor RealFlow je komerční aplikace, zaměřená především na simulování tekutých materiálů. Výsledky tohoto projektu je možno vidět převážně ve filmovém odvětví.

Kapitola 2

Metody

V této kapitole bude popsáno, jak fungují jednotlivé metody simulace proudění tekutin a jaké rovnice v těchto postupech platí.

2.1 Navier – Stokesovy rovnice

Jak již bylo zmíněno v úvodu, tyto rovnice tvoří základ výpočtu dynamiky tekutin. Lze je aplikovat na různé typy tekutin, my však budeme pracovat pouze s tzv. Newtonovskými kapalinami, čili takovými, které jsou nestlačitelné a dynamická viskozita je konstanta úměrnosti mezi napětím a rychlostí deformace.

Tyto rovnice počítají se třemi základními vlastnostmi tekutin. Jsou jimi rychlost v, hustota ρ a tlak p. Výpočet vektoru zrychlení proběhne na základě rovnice pro vyjádření pohybu v čase t (2.1), kde μ je viskozita tekutiny a f je součet všech externích sil, působících na tekutinu. Základní Navier – Stokosovy rovnice jsou založeny na mřížkové struktuře tekutiny. To znamená, že při výpočtu nezáleží pouze na čase, ale i na pozici v mřížce. Zrychlení je poté vyjádřeno v levé části rovnice, skládající se ze zrychlení závislého na čase ($\frac{\partial}{\partial t}$) a na pozici v mřížce ($v \cdot \nabla v$).

$$\rho(\frac{\partial}{\partial t} + v \cdot \nabla v) = -\nabla p + \mu \nabla \cdot \nabla v + f \tag{2.1}$$

Rovnice (2.2) vyjadřuje zachování hustoty v Newtonovských tekutinách.

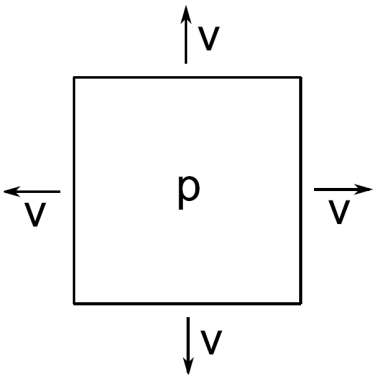
$$\nabla \cdot v = 0 \tag{2.2}$$

2.2 Eulerova metoda

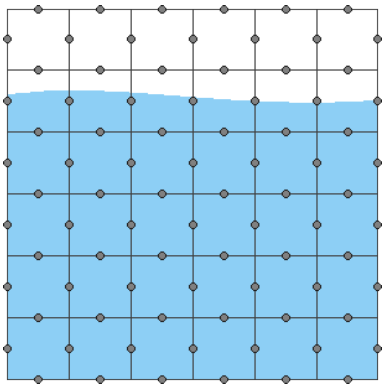
Tato metoda je založená na Navier – Stokesových rovnicích. Tekutina je rozložená do mřížky, kde každá buňka mřížky obsahuje několik částic (např. molekul). V závislosti na prostoru se pracuje buď s 2D, případně 3D mřížkou. Stěny buňky obsahují údaj o rychlosti buňky, obsah buňky pak o tlaku tekutiny v buňce (2.1a).

Pro simulaci tekutiny tímto způsobem, musí být celá tekutina zachycena v mřížce. Metoda nepodporuje situaci, kdy by část tekutiny byla mimo. Proto systémy, které nejsou ve tvaru čtverce pro 2D (2.1b), nebo krychle pro 3D, obsahují prázdné buňky, nicméně stále zabírají místo v paměti. V reálných systémech se vyskytují takovéto ideální situace jen zřídka. Z toho plyne, že tato metoda je velice náročná na paměť. Pro optimalizaci

se používají různé metody, například členění prostoru do oktanového stromu. Takto lze dosáhnout menšího dělení mřížky v zaplněných částech systému a detailnějšího členění na okrajích. I přes tyto optimalizace a fakt, že Eulerova metoda popisuje některé vlastnosti tekutin lépe než jiné metody, například tlak či hustotu, je výpočetní a paměťová náročnost příliš velká, proto se tento přístup nehodí pro simulaci v reálném čase.



(a) Obrázek popisující buňku mřížky



(b) Šedé body představují místa s rychlostním vektorem příslušné buňky

Zatímco malé deformace povrchu kapaliny jdou touto metodou simulovat dobře, simulace rozsáhlých deformací (např. tříštění kapky) selhávají.

2.3 Smoothed particle hydrodynamics

Smoothed particle hydrodynamics (SPH) byla původně zavedena pro simulování osově nesymetrických jevů v astrofyzice [6]. Narozdíl od Eulerovy metody nepracuje s mřížkou, ale s částicemi, tím pádem odpadá paměťová náročnost (v poměru k Eulerové metodě). Místo toho se využívá interpolačních rovnic [3]. Výpočet ovlivňuje takzvané jádro, které funguje jako filtr. Bližší částice mají na právě počítanou částici větší vliv, než ty vzdálené. Rozpětí tohoto jádra definuje tzv. vyhlazovací délka h. V reálných systémech je tato hodnota neomezená, čili zahrnuje všechny částice v jakékoliv vzdálenosti, v simulačních programech je pak zvolena vhodná hodnota.

2.3.1 Rovnice

Interpolační integrál lze zapsat jako

$$A_I(r) = \int A(r')W(r - r', h)dr', \tag{2.3}$$

kde rozsah integrálu je celý prostor ω, r je bod, ve kterém vyhodnocujeme funkci A, h je parametr jádra, který určuje vyhlazovací délku, r' je nezávislá proměnná a W je interpolační jádro, které má následující dvě vlastnosti

$$\int W(r - r', h)dr' = 1, \tag{2.4}$$

$$\lim_{h \to 0} W(r - r', h) = \delta(r - r'). \tag{2.5}$$

Číselný ekvivalent pro funkci, pro kterou známe její diskrétní hodnoty, lze zapsat jako suma

$$A_S(r) = \sum_{j=1}^{N} A_j V_j W_j(r - r_j, h), \tag{2.6}$$

kde V_j je objem spojený s částicí j. Když nahradíme objem V_j podle vzorce

$$V = \frac{m}{\rho}, \tag{2.7}$$

dostaneme rovnici ve tvaru

$$A_S(r) = \sum_{j=1}^{N} A_j \frac{m_j}{\rho_j} W_j(r - r_j, h) \tag{2.8}$$

Po zderivování obou stran rovnic dostaneme tvar

$$\nabla A_S(r) = \sum_{j=1}^{N} A_j \frac{m_j}{\rho_j} \nabla W(r - r_j, h) \tag{2.9}$$

Z této rovnice vyplývá, že jediná veličina, která závisí na derivaci r je jádro W.

2.3.2 Jádra

Z podkapitoly 2.1 víme, že na výpočtu směrového vektoru síly částice se podílí, mimo jiné, tři základní složky, a to jsou hustota ρ, tlak p a koeficient viskozity. Každá z těchto složek se pro stejnou částici počítá jinak. Výpočet je ovlivněn právě takzvaným jádrem.

Obecný tvar jádra vyplývá z grafu funkce 2.2. Tento model odpovídá reálným systémům, kde jedna částice působí na částice v jakékoliv vzdálenosti. Z grafu je ale vidět, jak rychle taková síla klesá a od určité vzdálenosti je téměř zanedbatelná. Grafem jádra je Gaussova křivka s popisem (2.10). V praxi se pak používá omezené jádro s podmínkou (2.11), aby se nepočítaly částice příliš vzdálené, ovlivněné velice málo. Dále musí jádro musí být nezáporné na celém definičním oboru.

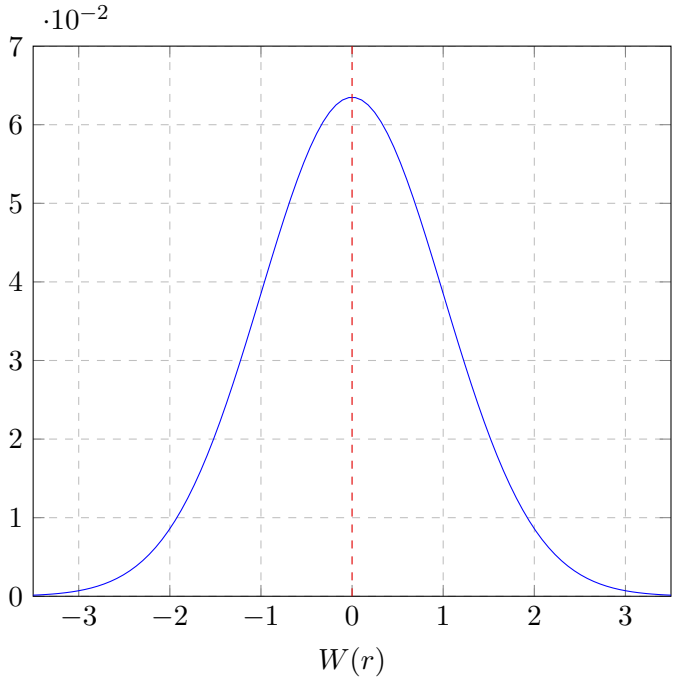
$$W(r, h) = \frac{1}{(2\pi h^2)^{\frac{3}{2}}} \cdot exp^{-\frac{|r|^2}{2h^2}} \tag{2.10}$$

$$W(r, h) = 0, |r| > h. \tag{2.11}$$

Standardní jádro

Standardní jádro 2.3 je vhodné pro výpočet všech veličin částice, pokud nejsou kladeny speciální požadavky. Jedním z takových požadavků je například nenulová hodnota derivace pro výpočet hustoty v bodě $r = 0$. Tento vzorec je oproti Gaussovu rychlejší pro výpočet, protože neobsahuje žádnou časově náročnou operaci (např. odmocninu). Graf tohoto jádra je znázorněn na 2.3, popis jeho funkce pak (2.12) [8].

$$W(r, h) = \begin{cases} \frac{315}{64\pi h^9}(h^2 - r^2) & , \quad 0 \leq r \leq h \\ 0 & , \quad jinak \end{cases} \tag{2.12}$$

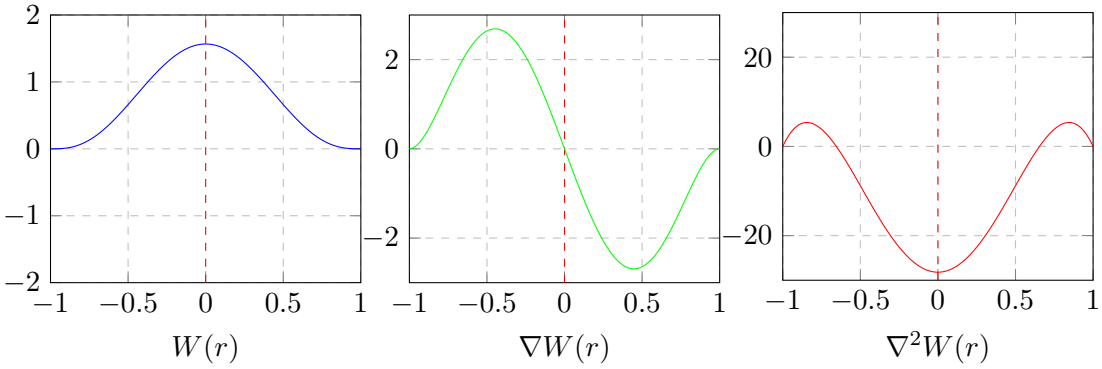


Obrázek 2.2: Obecné Gaussovo jádro pro $h = 1$

Pro použití metody SPH je zapotřebí znát ještě gradient a laplacián jádra, čili první a druhou derivaci (2.13) a (2.14).

$$\nabla W(r,h) = -\frac{945}{32\pi h^9}(h^2 - |r|^2)^2 \cdot r \quad , \quad 0 \leq r \leq h \tag{2.13}$$

$$\nabla^2 W(r,h) = -\frac{315}{64\pi h^9}(h^2 - |r|^2)(3h^2 - 7|r|^2) \quad , \quad 0 \leq r \leq h \tag{2.14}$$



Obrázek 2.3: Omezené (standardní) jádro pro $h = 1$

Z grafu 2.3 je vidět, že standardní jádro má v bodě $r = 0$ nulovou normálu, což způsobuje problémy pro výpočet tlaku a viskozity. Proto se pro ně používají speciální jádra.

Jádro pro tlak

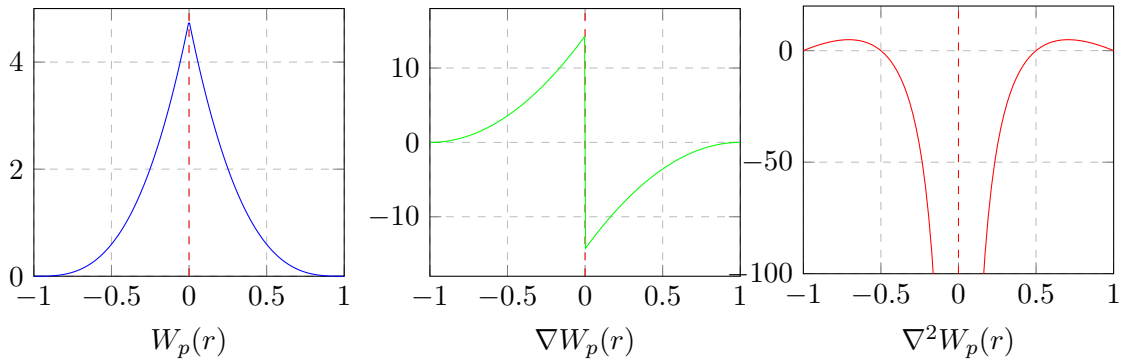
Kvůli výše popsanému problému s nulovou derivací v $r = 0$ bylo potřeba zavést nové jádro pro výpočet tlaku. Jedna z možností jádra je zobrazena na grafu funkce 2.4 s předpisem (2.15) [8].

$$W_p(r,h) = \begin{cases} \frac{15}{\pi h^6}(h - |r|)^3 & , \quad 0 \leq r \leq h \\ 0 & , \quad jinak \end{cases} \tag{2.15}$$

Stejně jako v případě standardního jádra potřebujeme znát gradient a laplacián (2.16), (2.17).

$$\begin{gathered} \nabla W_p(r,h) = -\frac{45r}{\pi h^6 |r|}(h - |r|)^2 \cdot r \quad , \quad 0 \leq r \leq h, \\ \lim_{r \to 0^-} \nabla W_p(r,h) = \frac{45}{\pi h^6}, \qquad \lim_{r \to 0^+} \nabla W_p(r,h) = -\frac{45}{\pi h^6} \end{gathered} \tag{2.16}$$

$$\begin{gathered} \nabla^2 W_p(r,h) = -\frac{90}{\pi h^6 |r|}(h - |r|)(h - 2|r|) \quad , \quad 0 \leq r \leq h, \\ \lim_{r \to 0} \nabla^2 W_p(r,h) = -\infty. \end{gathered} \tag{2.17}$$



Obrázek 2.4: Jádro pro výpočet tlaku s $h = 1$

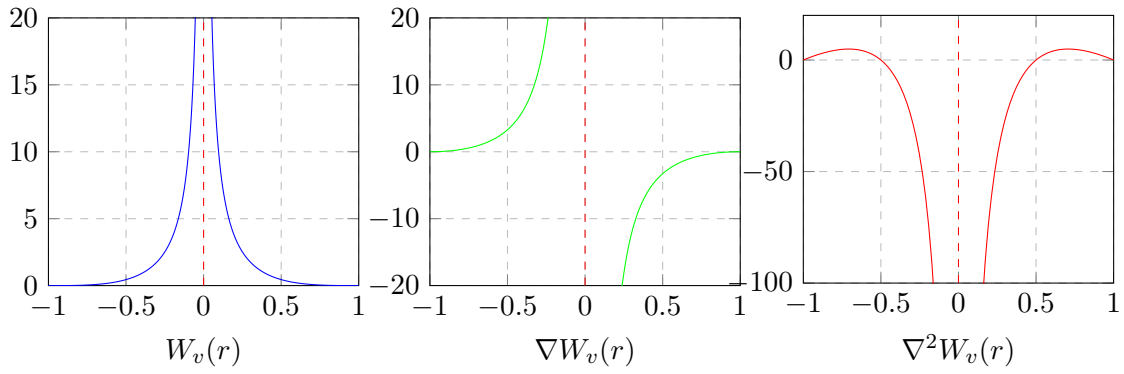
Jádro pro viskozitu

Při výpočtu působení viskozních sil je zapotřebí laplacián jádra $\nabla^2 W$. Protože výše zmíněná jádra mají laplacián záporný, musí se použít další speciální jádro. Jedno z možných jader je předepsáno na (2.18) s gradientem (2.19) a laplaciánem (2.20) [8].

$$W_v(r,h) = \frac{15}{2\pi h^3} \begin{cases} (\frac{-|r|^3}{2h^3} + \frac{|r|^2}{h^2} + \frac{h}{2|r|} - 1) & , \quad 0 \leq r \leq h \\ 0 & , \quad jinak \end{cases} \tag{2.18}$$

$$\begin{gathered} \nabla W_v(r,h) = \frac{15}{2\pi h^3} r(\frac{-3|r|}{2h^3} + \frac{2}{h^2} - \frac{h}{2|r|^3}), \\ \lim_{r \to 0^-} \nabla W_v(r,h) = +\infty, \qquad \lim_{r \to 0^+} \nabla W_v(r,h) = -\infty \end{gathered} \tag{2.19}$$

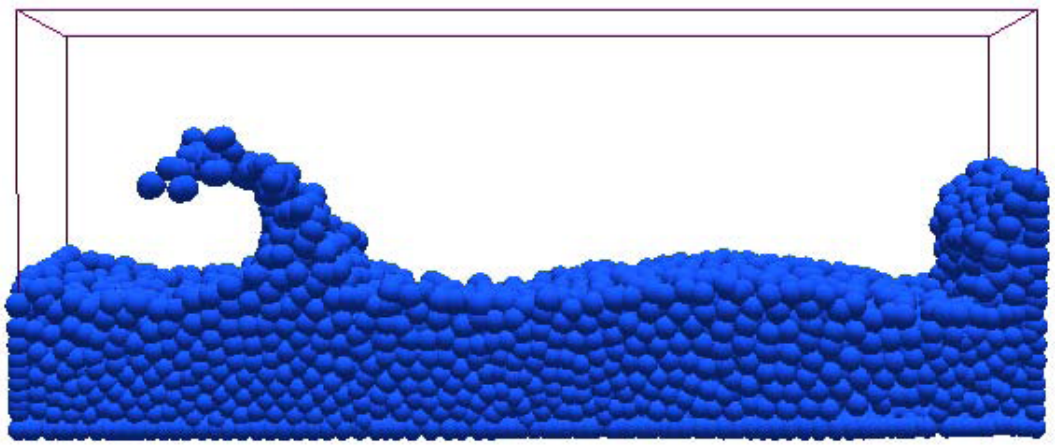
$$\nabla^2 W_v(r,h) = \frac{45}{\pi h^6}(h - |r|) \tag{2.20}$$



Obrázek 2.5: Jádro pro výpočet viskozity s $h = 1$

2.4 Lagrangeova metoda

Jedná se o další z možností simulování tekutin. Na rozdíl od Eulerovy metody není založena na mřížce (i když existuje i varianta s mřížkou [4]), nýbrž na soustavě částic, z nichž každá obsahuje určité množství tekutiny (2.6). Pro každou takovou částici pak platí všechny přírodní zákony jako pro kapalinu, již reprezentují.



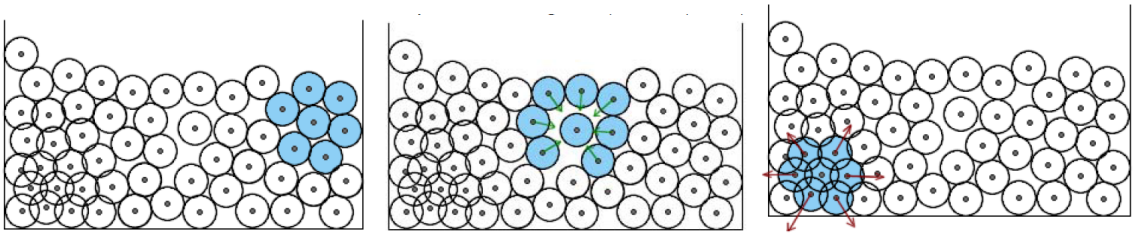
Obrázek 2.6: Rozdělení tekutiny na částice.

Budeme se zabývat pouze situacemi, kdy množství částic a teplota jsou konstantní. Proto nemusíme brát v potaz změnu hustoty tekutiny, viz rovnice (2.2).

2.4.1 Tlak

Jak bylo zmíněno na začátku kapitoly 2.1, jednou složkou pro výpočet výsledné síly na částici je tlak. Tlak na částici se odvíjí od hustoty částic v okolí [1]. Znázornění vlivu hustoty na tlak zobrazeno na obrázcích 2.7a, 2.7b a 2.7c.

K vypočítání směru a velikosti vektoru síly, který je dán tlakem, je zapotřebí znát hodnotu hustoty ve všech částicích v dosahu jádra (2.15). Pro všechny kapaliny platí, že mají svoji základní hustotu značenou ρ_0. V případě, že bychom tuto hustotu nevzali v potaz, kapalina by měla tendenci se neustále rozpínat. Proto se při výpočtu počítá s rozdílem



(a) Zvýrazněné částice s rovnoměrným tlakem

(b) Částice v nerovnoměrném okolí s vyznačenými vektory síly

(c) Částice v nerovnoměrném okolí s vyznačenými vektory síly

aktuální hustoty ρ a ρ_0. Tlak vypočítáme tedy podle vzorce

$$p = k(\rho - \rho_0),$$

kde k je konstanta tuhosti (pro konstantní hmotu a teplotu kapaliny). Výslednou sílu, působící na částici v rámci vlivu tlaku, pak spočítáme podle vzorce (2.21) [7].

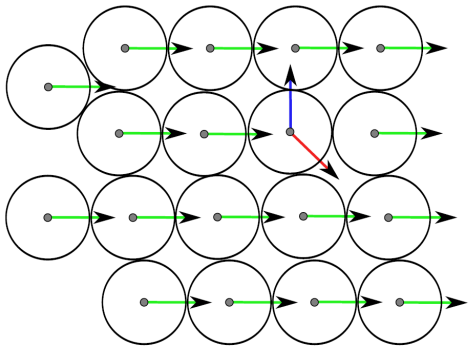
$$f_i^{\text{press}} = \sum_{j=1}^{N} m_j \frac{p_i + p_j}{2\rho_j} \nabla W(r - r_j, h), \tag{2.21}$$

Jak již bylo zmíněno v sekci 2.3.2, není vhodné používat standardní jádro (popsané v 2.3.2). Toto jádro má s přibližováním se nule snižující se gradient, a proto na částice, které se dostanou blízko sebe, působí velmi malá, až žádná, odpudivá síla. S využitím standardního jádra by tedy vznikaly shluky částic o velké hustotě.

2.4.2 Viskozita

Viskozita udává, jak velký vliv mají rychlosti okolních částic na právě počítanou částici. Čím je tedy hodnota větší, tím více se nechává částice ovlivnit okolím. Jako příklad pro kapalinu s velkou viskozitou si uveďme třeba med.

Pro výpočet viskozity musíme opět použít jiné jádro než standardní. Se zkracující se vzdáleností mezi částicemi, roste vzájemný vliv rychlostí velice rychle. Bližší částice působí výrazně více než ty vzdálenější. Pro popis tohoto vztahu se použije jádro popsané v sekci 2.3.2.



Obrázek 2.8: Vliv okolí na částici

Na obrázku 2.8 můžeme vidět, jak se podílí okolí částice na jejím vektoru síly. Zelenými šipkami jsou vyznačeny rychlostní vektory částic, modrým pak demonstrační rychlost částice n. Výsledná síla (červený vektor) je pak výsledkem působení viskozity a gravitace (viz 2.4.4).

V reálném systému se vlivem viskozity vytváří tření, které je transformováno na teplo. V simulačním prostředí tento jev ale zanedbáme, proto můžeme pracovat s kapalinou o konstantní hmotě a teplotě.

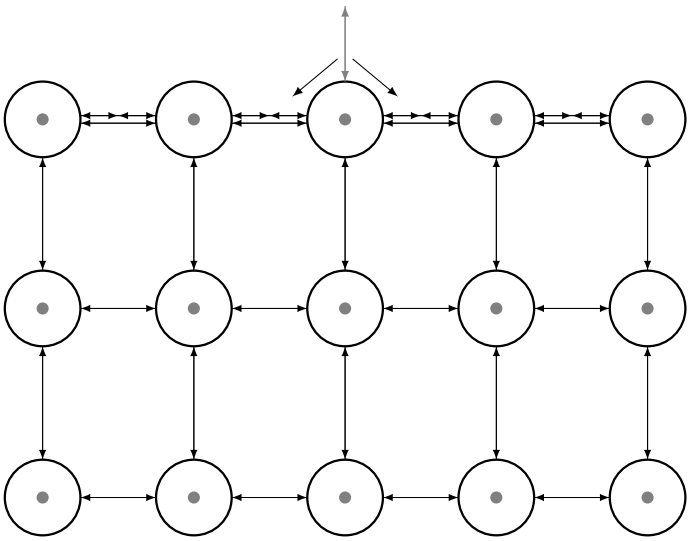
Pro samotný výpočet viskozity použijeme vzorec (2.22) [7], kde μ je koeficient viskozity.

$$f_i^{\text{visc}} = \mu \sum_{j=1}^{N} m_j \frac{v_j - v_i}{\rho_j} \nabla^2 W(r - r_j, h) \tag{2.22}$$

2.4.3 Povrchové napětí

Povrchové napětí je jev, který vzniká díky nevyváženým mezimolekulárním silám [1]. Zatímco uvnitř kapaliny na každou částici působí okolí v celém rozsahu prostoru, na částice na okraji kapaliny je toto okolí nevyvážené. Výslednice sil na povrchu kapaliny je pak kolmice do středu kapaliny. U každé kapaliny je koeficient ovlivňující sílu povrchového napětí jiný, označme jej σ. Výpočet síly povrchového napětí je uveden na (2.25) a znázorněn na 2.9 [2].

Čím větší je zakřivení povrchu, tím větší je síla povrchového napětí, která tlačí kapalinu do středu. Na základě toho, se kapalina, na kterou nepůsobí žádné jiné síly, formuje do tvaru koule.



Obrázek 2.9: Znázornění sil povrchového napětí

Metoda, zabývající se nalezením částic na povrchové vrstvě, se nazývá *obarvení* [7]. Toto obarvení poté tvoří pole, kde na pozici částice nabývá hodnoty 1 a hodnotu 0 jinde. Rovnici pro výpočet obarvení je uveden na (2.23). Podle gradientu obarvení ∇c_S můžeme

pozorovat, že jeho délka roste pouze v případě blízkosti k povrchu kapaliny, takže snadno poznáme částice na okraji, a její směr roste směrem ke středu kapaliny. Zakřivení povrchu kapaliny označme κ.

$$c_S(r) = \sum_{j=1}^{N} 1 \frac{m_j}{\rho_j} W(r - r_j, h) \tag{2.23}$$

$$\kappa = \frac{-\nabla \cdot \nabla c_S}{|\nabla c_S|} \tag{2.24}$$

$$F_{\text{surface tension}} = \sigma \kappa \nabla c_S = -\sigma \nabla^2 c_S \frac{\nabla c_S}{|\nabla c_S|} \tag{2.25}$$

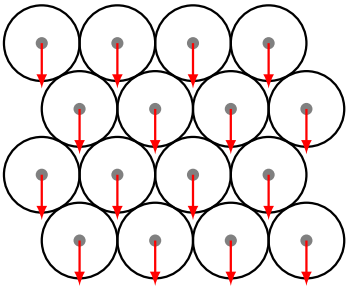
2.4.4 Gravitační síla

V reálném systému existuje nekonečně mnoho gravitačních sil, které působí na každé těleso. To je dáno vzorcem (2.26), kde m_1 a m_2 jsou hmotnosti těles na sebe působících, r je vzdálenost mezi těmito tělesy a κ je gravitační konstanta.

$$F_g = \kappa \frac{m_1 m_2}{r^2} \tag{2.26}$$
$$\kappa = 6.67 \cdot 10^{-11} [m^3 \cdot kg^{-1} \cdot s^{-2}]$$

V rámci simulace si tento vzorec ale patřičně zjednodušíme a zohledníme pouze největší položku z gravitačních sil a to sílu tíhovou. Tato síla působí na každou částici zvlášť, i když hodnota je stejná. Gravitační sílu pro částici tedy vypočteme podle vzorce (2.27), kde m je hmotnost částice a $a = 9.81275$[1] je tíhové zrychlení[2] Působení gravitace na částice je znázorněno na obrázku 2.10.

$$F_g = m \cdot a \tag{2.27}$$



Obrázek 2.10: Působení gravitace na částice

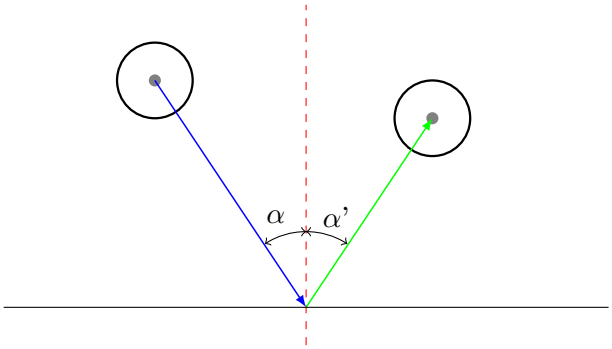
[1] http://earth-info.nga.mil/GandG/wgs84/gravitymod/egm2008/

[2] Tíhové zrychlení se v rámci jedné planety mění, protože do jeho výpočtu je zahrnuta odstředivá síla rotace. Námi použitá konstanta je platná pro oblast Brna.

2.4.5 Odrazy

Působením sil se částice uvádí do pohybu. Pokud ale nastane situace, že se částice dostane do kontaktu s okolním objektem (nepočítaje další částice, protože ty tam fyzicky nejsou a místo odrazů na sebe působí tlakem), tak se částice odrazí 2.11.

I zde platí zákon dopadu a odrazu [9]. Částice dopadne pod úhlem α vůči normále povrchu kolizního tělesa a pod úhlem $\alpha = \alpha'$ se opět od povrchu odrazí. V rámci odrazu se ale část energie přenese do kolizního materiálu a energie částice se zmenší.



Obrázek 2.11: Odražení částice podle zákonu dopadu a odrazu a absorbování části síly.

Kapitola 3

Programová část

3.1 Cíl

Cílem bakalářské práce je vytvořit aplikaci, která v reálném čase simuluje chování tekutin. Pro samotné vytvoření aplikace simulující tekutiny lze použít více přístupů, některé z nich jsou popsány v kapitole 2. Hlavním problémem práce tedy je právě ta část o zpracovávání v reálném čase. Toho lze dosáhnout buď simulováním malého počtu částic, maximální optimalizací kódu (i zde ale opět narážíme na hranici maximálního počtu částic), nebo využitím paralelizace. Následující program je implementován za pomoci technologie *CUDA*, čímž využívá právě výhod paralelizmu.

3.2 Technologie

Projekt je implementován v jazyce C++ s využitím několika následujících technologií. Jako vývojové prostředí bylo použito Microsoft Visual Studio 2012 Ultimate.

3.2.1 CUDA

CUDA je technologie vyvíjená společností NVIDIA. Jedná se o paralelně pracující platformu a programovací model, který výrazně dokáže zvýšit výpočetní výkon počítače za využití GPU.

Princip výpočtu pomocí CUDA je následující:

1. Alokování paměti na grafické kartě
2. Nakopírování dat z operační paměti na GPU (případně jen inicializace hodnot)
3. Spuštění CUDA jádra a provedení operací nad daty
4. Zpracování (např. zobrazení) dat, případně zkopírování zpět do operační paměti
5. Uvolnění paměti (pokud již není dále potřeba)

Paralelismus se využívá v případě, že máme cyklus o předem známém počtu iterací. V takovém případě můžeme zavolat CUDA jádro (kernel), které provede výpočet nad daty. V porovnáním s CPU má GPU výrazně větší počet aritmeticko – logických jednotek (ALU), proto se při zavolání jádra použije pro každou iteraci cyklu zvláštní ALU a výpočty mohou běžet současně.

Obrázek 3.1: Porovnání CPU a GPU

3.2.2 OpenGL

Na vytvoření grafického výstupu programu je využita knihovna OpenGL, která zajišťuje jak vytvoření okna výstupu, tak vykreslení jeho obsahu a překreslování. K práci s maticemi (posun, rotace, perspektiva,...) jsou také využity funkce z této knihovny.

3.3 Návrh

Když se podíváme na aplikaci z co nejsvrchnějšího pohledu, uvidíme renderer. Proto i v aplikaci bude renderer hlavním objektem, který se bude starat o chod simulátoru. Simulátor jako takový, pak bude atributem rendereru.

Lidské oko snímá okolí rychlostí zhruba 24 FPS (frame per second). Podle toho se pak bude odvíjet rychlost překreslování okna. To by mělo vyvolat dojem, že jde o souvislou animaci. K implementaci tohoto problému pak poslouží časovač, který bude po maximálně čtyřiceti milisekundách (25 FPS) dávat povel k překreslení. V každém překreslení se pak bude volat metoda simulátoru, která zajistí simulační výpočet a vypočítání nových souřadnic částic. Následně se data předají OpenGL knihovně, která je vykreslí.

Aplikace bude reagovat na stisky příslušných kláves, například pro ukončení simulace, restartování, přidání objektu a podobně.

Program se ukončí po stisknutí příslušné klávesy nebo zavření okna.

3.4 Renderer

Jak bylo popsáno v 3.3, hlavním - řídícím - objektem je instance třídy `Renderer`, která se stará o rozhraní celé aplikace. Tato instance je tvořena ve funkci `main()`, spolu s instancí třídy `Fluid_Simulator`. Vykreslování se zahájí voláním metody `Renderer->render(int ms)`, kde parametr `ms` značí periodu mezi překreslením okna, viz 3.4.2.

3.4.1 Inicializace

Před samotným spuštěním časovače musíme `Renderer` inicializovat. To znamená vytvořit okno, zadat parametry (šířka, výška, hloubka, počet barev, ...), inicializovat shadery 3.4.5 a zavolat metodu na iniciování hodnot simulátoru 3.6.1.

3.4.2 Překreslování

Ze sekce 3.3 víme, že pro správné zobrazování a navození dojmu, že se jedná o spojitou simulaci, se musí obraz překreslovat minimálně stejnou rychlostí, jako je lidské oko schopno obraz snímat. Hodnota parametru `ms` v metodě `Renderer->render(int ms)` je nastavena na 40 ms a její měření zajišťuje třída `RedrawTimer`. Tato hodnota není pro překreslení okna absolutní. V případě nutnosti (zvětšení okna, otočení scény,...) se vyvolá událost pro vynucení překreslení a okno se aktualizuje, nehledě na hodnotu časovače.

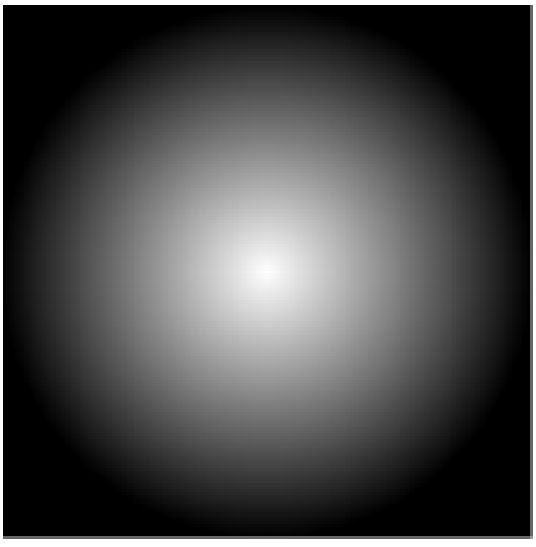
3.4.3 Události

Třída `Renderer` využívá hojně funkcí z knihovny OpenGL a SDL. Časovač z předchozí sekce 3.4.2 využívá pro překreslení okna událost `SDL_VIDEOEXPOSE`. Pro ovládání programu B jsou využity události `SDL_KEYDOWN`, `SDL_MOUSEMOTION`, `SDL_MOUSEBUTTONDOWN` a `SDL_QUIT`. Při vyvolání události se zavolá příslušná obsluhující rutina a po jejím provedení se program vrátí do normálního toku výpočtu.

3.4.4 Textury

I když se to nezdá, textura může za 90 % dojmu z aplikace. Pouze na textuře závisí, zda na obrazovce budou vykresleny jen body, kuličky znázorňující částice nebo reálně vypadající tekutina. Z důvodu zaměření se především na simulující část programu je implementována varianta s kuličkami.

Pro maximální akceleraci a efektivitu programu je jako medium pro vykreslování použita OpenGL entita `POINTS`. Tyto body se však vykreslují jako krychličky, kterým sice lze nastavit velikost, ale tato velikost je nezávisle na vzdálenosti od kamery konstantní. Abychom vytvořili z hranatých bodů kuličky, je nutno na tyto body nanést texturu s alfa kanálem 3.2, která v závislosti na množství alfy propouští podkladovou vrstvu. Výsledkem jsou kulaté body.



Obrázek 3.2: Alfa kanál textury

Po aplikaci α – textury jsou částice sice kulaté, ale mají naprosto totožnou barvu a tak ve větším počtu splývají do souvislé masy. Proto muselo být implementováno stínování,

které dodá částici náznak plastičnosti, byť se ve skutečnosti pořád jedná o 2D texturu.

S nanášením textur se váže sekce o shaderech 3.4.5.

3.4.5 Vykreslení modelu

Na vykreslení samotného obsahu bufferu se podílí metoda `Renderer->_render()`. Součástí této metody je poté i samotný krok simulace 3.6. Co se týče ale příkazů v souvislosti s vykreslením scény, musí mít kód v závislosti na OpenGL určitou strukturu či posloupnost.

Před samotným vykreslením dat se musí nastavit takzvaná *model view (MV)* matice. Jde o matici složenou ze tří jiných matic, které popisují polohu scény vůči kameře, čili to, co vidíme v SDL okně.

Protože matice se skládají maticovým násobením, je důležité, v jakém pořadí je skládáme. Pokud bychom zvolili pořadí rozdílné od (3.1), proběhl by například napřed posun objektu od středu soustavy a poté se - stále s osou otáčení ve středu soustavy - objekt otočil. Proto je důležité na toto pořadí dbát.

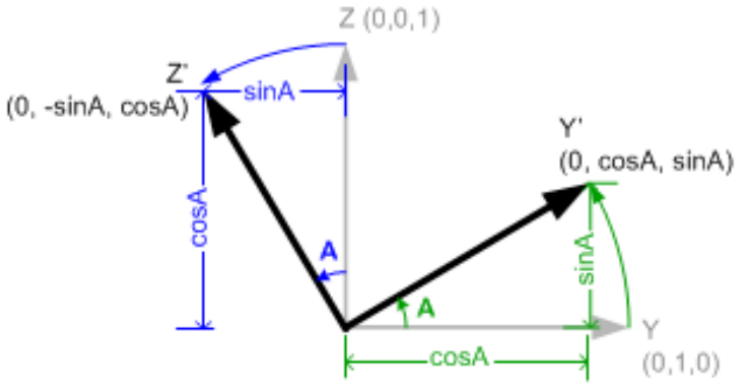
$$\mathrm{MV} = \mathrm{rotate} * \mathrm{scale} * \mathrm{translate} \tag{3.1}$$

Jako poslední úprava této matice je rozšíření na *projection model view* matici. Ta má v sobě zakomponováno to, jakým způsobem se budeme na scénu dívat. Typy projekce jsou dva, nás bude zajímat pouze projekce perspektivní (3D). Úprava proběhne podle vzorce (3.2), opět záleží na pořadí.

$$\mathrm{PMV} = \mathrm{projection} * \mathrm{MV} \tag{3.2}$$

Rotace

Pro otočení objektu se používá rotační matice. Její použití má za následek otočení modelu okolo zvolené osy. V případě potřeby otočení po více osách se sloučí více matic pro rotaci kolem jedné osy.



(a) Graf rotace

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos A & -\sin A \\ 0 & \sin A & \cos A \end{pmatrix}$$

(b) Matice rotace

Obrázek 3.3: Příklad rotace podle osy x

Zvětšení

Objekty jsou uloženy v paměti GPU v určitém měřítku. Pokud ale chceme vykreslit objekt dvakrát větší než máme uložený, není potřeba jeho hodnoty v paměti měnit. Stačí jen využít takzvané *scale* matice, která toto zvětšení provede. Pokud bychom chtěli obraz převrátit, lze této využít také této matice s parametrem zvětšení -1. Rovnice (3.3) popisuje obecnou matici zvětšení o S_x ve směru osy x, S_y ve směru y a S_z ve směru osy z.

$$\begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.3}$$

Posun

V případě potřeby posunu celého objektu opět nebudeme měnit hodnoty bodů uložených v paměti. Místo toho využijeme translační matice. Tato matice se aplikuje jako poslední, až jsou veškeré další úpravy polohy aplikovány. Jak tomu bylo u rotace 3.4.5, i zde se posun ve více směrech skládá ze součtu posunů v jednotlivých směrech. Obecná matice pro posun ve více směrech je pak uvedena (3.4). Posun proběhne o T_x po ose x, o T_y po y a o T_z po ose z.

$$\begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.4}$$

Shadery

Protože v celé simulaci pracujeme s 3D modelem, používáme i 3D souřadnice. My ale potřebujeme vykreslit model ve 2D na obrazovku. O převod 3D souřadnic na 2D se stará *vertex shader*, který přijímá pole vrcholů (body s 3D souřadnicemi) a vrací jejich souřadnice na obrazovce. Zdrojový text tohoto shaderu je uložen v proměnné `vertexShaderSource`.

Všechny barvy (např. kvádr pro průběh simulace) i textury (částice) jsou nanášeny ve *fragment shaderu*, jehož zdrojový text je uložen v proměnné `fragmentShaderSource` pro obyčejnou barvu a `textureShaderSource` pro texturu. To je zvláštní kus kódu, který nanáší na všechny body do něj zaslané definovanou barvu/texturu.

3.5 Fluid_Simulator

V této sekci je popsána implementace samotného simulátoru.

3.5.1 Vytvoření simulátoru

Jak již bylo zmíněno v 3.3, simulátor je součástí rendereru. Simulátor je třeba nejdříve ale vytvořit a to voláním jeho konstruktoru s patřičnými parametry. V implementovaném

řešení se simulátor vytváří funkcí `new Fluid_Simulator(PARTICLES_COUNT)`, kde její jediný parametr je počet částic, s nimiž se bude pracovat[1].

Součástí kontruktoru je i částečná inicializace hodnot a to těch, které nemá smysl při restartu simulátoru 3.6.6 měnit. Příkladem se může jednat o výšku a šířku simulátoru 3.5.2, hmotnost částice, velikost částice a další.

3.5.2 Atributy

Pro simulátor jsou nastaveny specifické konstanty, které ovlivňují hodnoty výpočtu. Všechny konstanty jsou definována jako makra `#define MACRO VALUE` a do výpočtů se tak mohou započítávat globálně. Mezi prvky atributů patří jak veličiny, týkající se konkrétně simulátoru (šířka, výška, velikost částic, ...), tak fyzikální vlastnosti (viskozita kapaliny, tíhová síla, ...).

3.5.3 Časovač

I když frekvenci simulátoru řídí renderer, tak je potřeba zjišťovat, jak dlouhá doba uběhla mezi kroky simulace. To je potřeba jednak pro to, že renderer nevykresluje s naprosto přesnou periodou (rozptyl samotného rendereru je ± 10 ms), ale i proto, že pokud se simulace na nějakém výpočtu zpozdí, musí se tato prodleva zohlednit při příštím kroku simulace.

Pro časovač byla použita struktura `time_t` ze standardní knihovny `time.h`. K práci s časovačem je využita funkce `gettimeofday()`[2].

3.5.4 Částice

Na rozdíl od ostatních simulátorů kapalin, tento pracuje s abstraktním pojmem částice. To znamená, že částici jako takovou, nikde v programu nenajdeme. Simulátor si zajistí jen množinu polí, které popisují fyzikální vlastnosti částic. Pro pozice je to pole `float4 _positionsD[]`, pro rychlost `float4 _velocityD[]` atd.[3] Další výhodou tohoto přístupu je, že jediné co potřebujeme znát, abychom se dostali k vlastnostem částice, je jen její index. To výrazně omezí velikost parametrů předávaných do funkcí.

Může se zdát, že použití `float3` pro veličiny, kdy stejně potřebujeme jen tři souřadnice, by bylo vhodnější. `float4` je použit kvůli zarovnání v paměti. Krom toho, u některých veličin se využívá právě čtvrté souřadnice pro implementaci zvláštního výpočtu 3.5.5.

3.5.5 Pozice

Pro uložení polohy částic slouží pole `Fluid_Simulator->_positionsD`. Poloha částic ve 3D prostoru je určena typem proměnné `float4`. První tři značí po řadě x, y a z souřadnici. Poslední položka struktury `float4` udává index do pole 3D mřížky 3.5.9.

To je velmi důležitý bod pro urychlení výpočtů a vyhledávání v prostoru. Na základě této hodnoty si pak buňky vybírají částice, které si uloží do svého seznamu indexů `TCubes.particles` 3.5.9.

V simulátoru je ještě pole `Fluid_Simulator->_positionsH`, přes které upravujeme pozice částic v poli `Fluid_Simulator->_positionsD` vnějším zásahem. Například uspořádání do koule, náhodný rozptyl atd. Způsob vkládání je popsán v sekci B.

[1] Tato hodnota je fixní a za běhu programu ji nelze měnit.

[2] Funkce byla převzata z http://social.msdn.microsoft.com/

[3] Písmeno `D` v názvu proměnné značí, že je pole alokováno v paměti GPU. Proto se do něj nesmí vstupovat v klasickém režimu kódu, což znesnadňuje zejména ladění programu.

3.5.6 Rychlost

Jedna z vlastností každé částice je vektor rychlosti. Tato vlastnost je implementována opět jako pole vektorů, kde index do něj je shodný s indexy do ostatních polí pro danou částici. Pole je typu `float4` kvůli zarovnání v paměti. Hodnoty jsou uloženy v proměnné `Fluid_Simulator->_velocityD`.

3.5.7 Hustota

Hustota je skalární veličina, proto nám pro uložení stačí použít pole typu `float`. Pole je uloženo v proměnné `Fluid_Simulator->_densityD`. Hustota částice se odvíjí od počtu částic v okolí. Jak bylo popsáno v 2.4.1, od hustoty ρ je potřeba odečíst ρ_0, aby se tekutina nerozpínala jak plyn. Tento rozdíl však implementován nebyl, protože při jeho užití se simulátor choval nestabilně.

3.5.8 Tlak

Další vlastností částic je tlak. Implementace je víceméně shodná s implementací rychlosti 3.5.6, hodnoty jsou uloženy v `Fluid_Simulator->_pressureD`.

3.5.9 3D mřížka

Protože musíme v rámci výpočtů zjišťovat, které částice patří do dosahu jádra a které už ne, byla implementována 3D mřížka přes celý prostor simulátoru. Ta má za účel vyloučit valnou většinu částic.

Mřížka je implementovaná jako pole struktur `TCubes`, kde každý prvek tohoto pole představuje buňku mřížky. V každé této struktuře je pak ukazatel na pole indexů částic, a počet částic v poli `TCubes.particles`.

Při inicializaci mřížky (probíhá ve funkci `Fluid_Simulator->_initParticles()` 3.6.1) se spočítá funkcí `Fluid_Simulator->_cubesCount()` počet buněk mřížky v závislosti na rozměrech simulátoru a velikosti vyhlazovací délky jádra `SMOOTHING_H`. Čím je tato délka větší, tím méně bude v simulátoru buněk mřížky a tím více částic na sebe bude působit.

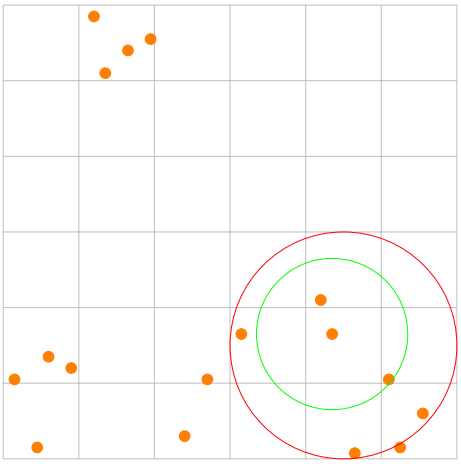
Při hledání částic, které na sebe navzájem působí se pak zavolá funkce `find_neighbors()`, které se jako parametr předá index aktuální buňky, pole buněk a rozměry simulátoru, projde se seznam touto funkcí vrácených sousedních buněk a pro každou buňku se projde seznam indexů částic, uložených v poli `TCubes.particles`.

Tímto způsobem se výrazně sníží obtížnost, která by byla bez využití tohoto mechanismu $O(N^2)$, kde N je počet částic.

Při každém kroku simulace se volá jádro `kernel_fill_cubes`, které naplní buňky 3D mřížky indexy částic. Aby tato metoda mohla fungovat, musí se na spolupráci podílet samotné částice. Více o propojení s částicemi v sekci 3.5.5.

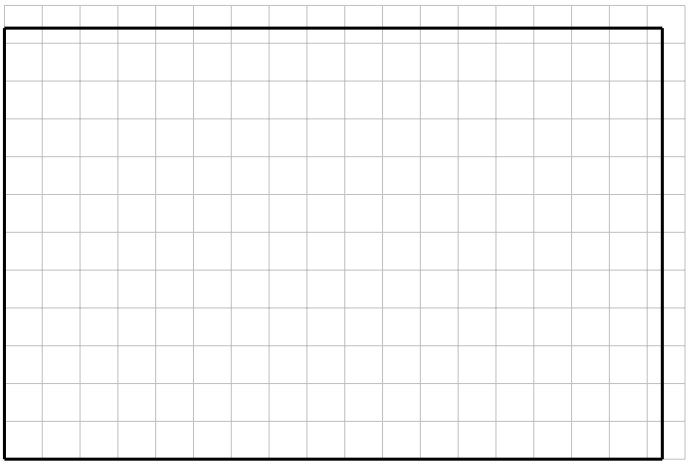
Na obrázku 3.4 můžeme vidět 2D pohled na mřížku. Částice r (ve středu zelené kružnice) má vliv pouze na částice v dosahu vyhlazovací délky. Tím, že je hrana buňky mřížky rovna právě vyhlazovací délce, na částici mohou mít vliv pouze částice z buněk, do kterých zasahuje červená kružnice.

Při inicializaci, která se děje v rámci funkce `Fluid_Simulator->_initParticles()` 3.6.1, se prostor simulátoru rozdělí do mřížky, kde délka hrany jedné buňky je `SMOOTHING_H`. Pokud ale nastane situace, že šířka simulátoru není dělitelná vyhlazovací délkou beze zbytku, musí se obalit mřížkou o něco větší část, než by bylo ideální. Pokud bychom podíl místo



Obrázek 3.4: Znázornění mřížky s částicemi, buňky mají délku hrany `SMOOTHING_H`. Zeleně je znázorněn dosah vyhlazovací délky částice r.

toho zaokrouhlili, některé částice by se mohli dostat mimo dosah mřížky a ve výpočtech by nikdy nebyly zahrnuty. Vzhled mřížky z 2D pohledu při necelém podílu je vidět na 3.5.

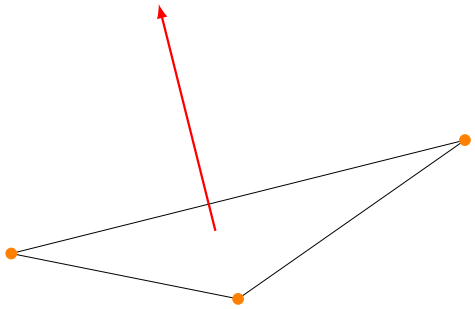


Obrázek 3.5: Vytvoření 3D mřížky při necelém podílu vyhlazovací délkou.

3.5.10 Překážky

Vnější částí simulátoru jsou překážky. Ať už jimi jsou stěny simulátoru, nebo případné objekty vložené do simulace. Překážky jsou uloženy jako kolizní plochy definované třemi vrcholy a normálou směřující z povrchu 3.6. V aplikaci jsou implementovány jako pole struktur `TCollideTri` a v simulátoru jsou v proměnné `Fluid_Simulator->_collisionsD`. Normála ve struktuře musí být normalizovaná.

V aplikaci jsou uloženy jako kolizní plochy jen stěny kvádru kontejneru, ale algoritmus je univerzální a teoreticky by se tímto způsobem dal upravit tvar kontejneru na jakýkoliv možný. Tímto způsobem se by se daly do simulátoru vložit případně další externí předměty 4.1.1.



Obrázek 3.6: Příklad kolizní plochy.

3.5.11 Výpočet sil

Hlavní algoritmus simulátoru je založen na tom, že se v každém kroku spočítá velikost a směr vektoru síly, který v daném okamžiku na částici působí. Na základě tohoto vektoru potom může proběhnout samotný výpočet rychlosti a změny pozice 3.5.6.

Jak bylo popsáno v 2.1, výsledná síla je směsice několika rozdílných sil. Jejich výpočet je popsán v sekci 3.6.

3.6 Průběh simulace

V této sekci bude popsán model výpočtu simulátoru, jak jdou jednotlivé kroky za sebou a v jaké části algoritmu se řeší jaký problém.

3.6.1 Inicializace

Samotná metoda simulátoru `Fluid_Simulator->init()` obsahuje volání dalších inicializačních metod pro konkrétní součásti simulátoru. Krom potřebných, byť rutinních inicializací jako jsou inicializace CUDA a OpenGL kontextu, se jedná o tyto metody:

- _initAquabox() - inicializace vrcholů ohraničení simulátoru (jakési akvárium pro částice)
- _initBox() - inicializace vrcholů ohraničení simulátoru pro vykreslování OpenGL
- _initParticles() - nejdůležitější inicializátor simulátoru
- _initCollisions() - inicializace pole trojúhelníků kolizních ploch a jejich normál
- _initTimer() - spuštění časovače pro simulaci

_initParticles()

Byť kontruktor již alokoval všechnu potřebnou paměť pro simulátor, data v ní jsou náhodná. Metoda `Fluid_Simulator->_initParticles()` tuto paměť buď naplní nulami, jak je tomu v případě pole souřadnic částic 3.5.5 či vektorů rychlostí částic 3.5.6, nebo rozdělí částice do podprostorů, takzvané 3D mřížky 3.5.9. Tímto způsobem se inicializují všechny potřebné datové struktury.

V této funkci si můžeme všimnout prvního použití volání CUDA jader. Většinou jde však o inicializační záležitosti, které kvůli urychlení výpočtu řešíme na GPU, a v průběhu simulace se s nimi již nesetkáme.

_initCollisions()

Kvůli zjišťování a následnému ošetření kolizí (např. kontakt částice se stěnou simulátoru) bylo potřeba vytvořit pole ploch, na kterých k těmto kolizím dochází 3.5.10. V metodě `Fluid_Simulator->_initCollisions()` probíhá pak inicializace jejich souřadnic v prostoru a nastavení normál, směřujících z povrchu plochy.

3.6.2 Krok simulace

Tato událost se děje při každém překreslení okna s parametrem časového kroku. Za zmínění stojí, že ač je v rovnicích popisující výpočet sil počítána vzdálenost aktivních bodů neustále znovu, v implementovaném řešení se spočítá jednou a do funkcí se předává jako parametr. Je to důsledek snahy vyhnout se velkému počtu časově náročných operací.

Hustota

Abychom mohli vůbec spočítat síly tlaku, jež působí na částici díky většímu shluku částic v okolí 2.4.1, musíme znát hustotu částic v daném místě. Tato hodnota musí být pro výpočet tlaku známa ve všech částicích, tudíž se musí vypočítat předem. Proto je voláno jádro `kernel_pressure`, které spočítá hustotu v okolí částice a na základě této hodnoty dosadí tlakový vektor do pole `Fluid_Simulator->_pressureD`.

Jádro musí nejdříve najít částice v dosahu vyhlazovací délky, na což je volána funkce `find_neighbors()`. Vrácené pole sousedních buněk se následně v cyklu projde a sečtou se počty částic uložených v buňkách a součet se uloží do proměnné `in_distance`.

Hodnotu hustoty následně zjistíme voláním funkce `dense()`, která přičte hodnotu hustoty pro každou částici v dosahu jádra 2.3.2. Použitý vzorec je na (3.5).

$$\rho_i = \sum_{j=0}^{N} m_j W(r - r_j, h) \tag{3.5}$$

Díky tomu, že není pro potřeby jádra znát přesnou vzdálenost částic, ale jen zda leží v dosahu vyhlazovací délky, může být pro výpočet vzdálenosti použita funkce `distancePoints2()`, která se vyhne užití odmocniny. Pro maximální urychlení výpočtu je konstantní část rovnice uložena v proměnné `W_dens`.

Gravitace

Výpočet působení tíhové síly na částici je velmi jednoduchý. Gravitace působí na všechny částice stejně. K vektoru síly se pouze připočte součin vektoru tíhového zrychlení a hmotnosti částice. Byť hmotnost částice nemá rychlost pádu vliv, musíme ji zde započítat, protože v pozdější fázi výpočtu sil se s ní pracuje a je její užití zde očekáváno.

Tlak

Když máme vypočítané hodnoty tlaků v částicích, je možné spočítat sílu, jež díky vzájemnému působení tlaků vznikla. Výslednou sílu spočítáme přesně podle vzorce (2.21) uvedeného v teoretické sekci o tlaku.

Výpočet síly provádí funkce `press()`. Konstantní hodnoty rovnice jsou vyhodnoceny v době překladu programu, proto se výpočet nezdržuje jejich výpočtem, a tato hodnota je uložena v proměnné `d_W_press`.

Viskozita

Pro výpočet síly, způsobené viskozitou, potřebujeme dvě veličiny. Jednou je rychlost částic, které na sebe působí, a druhou jsou jejich souřadnice. Výpočet probíhá podle vzorce (2.22), kde konstantní část je uložena v proměnné `dd_W_visc`.

Povrchové napětí

Výpočet povrchového napětí je trochu jiný, než tomu bylo u tlaku a viskozity. Pro částici se nejdříve musí spočítat velikost normály síly povrchového napětí, která závisí na poloze částice. Částice na kraji tekutiny budou mít velikost normály větší, než ty vzdálenější. Podle typu tekutiny se pak ořízne výběr částic, na které bude povrchové napětí působit, právě podle velikosti normály.

Základní vztah pro sílu povrchového napětí je

$$F_s = \sigma \kappa n,$$

kde σ je koeficient povrchového napětí, κ značí zakřivení povrchu a n je normála síly. Jednotlivé veličiny spočítáme podle soustavy rovnic (3.6).

$$\kappa = -\frac{\nabla^2 c_i}{|n|} \tag{3.6}$$

$$\nabla^2 c_i = \sum_{j=1}^{N} m_j \frac{1}{\rho_j} \nabla^2 W(r_i - r_j, h) \tag{3.7}$$

$$n = \sum_{j=1}^{N} m_j \frac{1}{\rho_j} \nabla W(r_i - r_j, h) \tag{3.8}$$

3.6.3 Rychlost

Výpočet rychlosti je dosti triviální. Pro její určení se využije aktuální hodnota rychlosti a připočte se podíl součinu síly působící na částici (3.9) s časovým krokem simulátoru a hmotnosti částice (3.10). Aktuální hodnota částic je poté uložena v poli `Fluid_Simulator->_velocityD`. Tento výpočet, spolu s výpočty sil, se provádí v CUDA jádru `kernel_force`.

$$F = F_g + F_p + F_v + F_s \tag{3.9}$$

$$v = v + \frac{F \cdot \text{timeStep}}{m} \tag{3.10}$$

3.6.4 Změna pozice

Cílem každého kroku simulace je právě změna pozice částice. Na základě rychlosti, která je spočítána v předchozím kroku, lze určit souřadnice budoucího výskytu částice. Samotný výpočet proběhne podle rovnice (3.11) a konkrétně je implementován ve funkci `newPosition()`, která jako parametry bere vektor rychlosti a časový krok simulátoru.

$$\text{Position} = v \cdot \text{timeStepS} \cdot \text{SIZE\_KOEF} \tag{3.11}$$

Člen rovnice (3.11) timeStepS vyjadřuje hodnotu časovače simulátoru v sekundách, SIZE_KOEF pak představuje korelační konstantu pro simulátor. Tato hodnota udává měřítko použité v simulaci.

Výpočet nové pozice je spolu s korelací odrazů počítán v jádru `kernel_position`. Jádra pro výpočet rychlosti a změny pozice nelze sjednotit, protože při výpočtu sil se čtou hodnoty polohy částice a kvůli rozdílnému běhu vláken se mohou přečíst neplatná data.

3.6.5 Odrazy

Každý simulační model musí být nějak ohraničený. Ohraničení `Fluid_Simulator` je dáno jeho rozměry 3.5.2. Proto je potřeba implementovat prostředek, který nám zajistí udržení částic v daném prostoru.

Tyto požadavky zpracovává funkce `collide()`, která pracuje ve dvou fázích. V první zjistí, zda vůbec ke kolizi s překážkou došlo, ve druhé pak vypočítá souřadnice a rychlost částice po kolizi.

Pro otestování, zda částice narazila do kolizního tělesa 3.5.10, se využívá znalostí analytické geometrie. Jelikož se kolize zjišťují až po vypočítání nové pozice částice, musíme si odvodit předchozí pozici částice. Zjištění kolize probíhá podle algoritmu 1.

Algoritmus 1: Detekce kolize a reakce na ni

```
1: before = stepBack(actual)                          // nalezeni předchozí pozice
2: if collisionArea is between(before, actual)        // test, zda proběhla kolize
3: then
4:     actual = intersection(before, actual, collisionArea)   // nastavení pozice
5:     velocity = reflect(velocity) * absorbtion              // nastavení rychlosti
6: end if
```

Test na kolizi proběhne v cyklu pro všechny kolizní plochy v `Fluid_Simulator->_collisionsD`.

3.6.6 Restartování simulátoru

V případě, že je potřeba simulaci opakovat, provede se restartování simulátoru. To spočívá v zavolání funkce `Fluid_Simulator->_init()` 3.6.1. K samotnému restartu tak může dojít například na popud rendereru, viz ovládání aplikace B.

3.7 Testování

3.7.1 Parametry

Aplikace byla testována na cyklickém spouštění s různými parametry. Zejména jsem se zaměřil na délku času potřebného pro krok simulace při počítání jednotlivých sil. Výpočet sil na částici je právě tou časově nejnáročnější složkou simulace. Testy probíhaly na kapalině s parametry podle tabulky 3.1. Stanice, na které byla aplikace vyvíjena, má parametry podle tabulky 3.2.

3.7.2 Časy

Jak bylo zmíněno na začátku sekce, hlavní směr testování byl na změření vlivu výpočtů jednotlivých fyzikálních vlastností kapaliny. Naměřené výsledky popisuje následující graf 3.7.

Vyhlazovací délka jádra h	8.0
Poloměr částice r	3.0
Konstanta tuhosti k	3.0
Hustota ρ_0	999.9720
Koeficient viskozity μ	0.01002
Hranice povrchového napětí `THRESHOLD`	7.065
Koeficient povrchového napětí κ	0.0728
Koeficient absorbování energie při odrazu	0.9

Tabulka 3.1: Tabulka fyzikálních hodnot simulované tekutiny

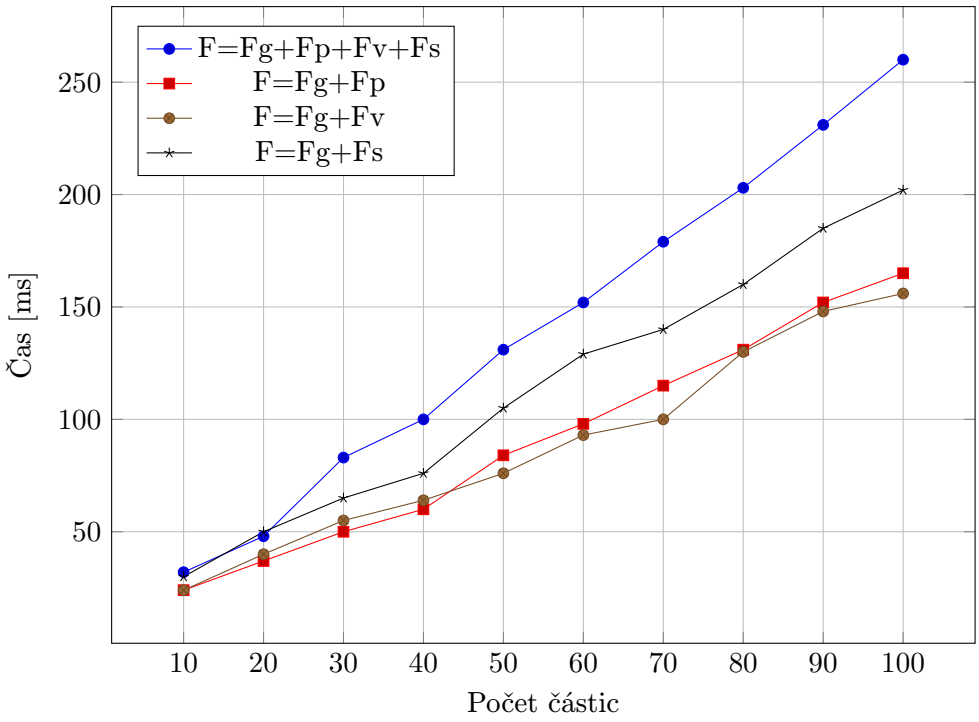
Operační systém	Windows 8 64bit
Vývojové prostředí	Microsoft Visual Studio 2012 Ultimate
Procesor	AMD Phenom(tm) II X4 840, 3.20GHz
RAM	4.00 GB
Grafická karta	NVIDIA GeForce GT 220

Tabulka 3.2: Hardwarové a softwarové parametry stanice.

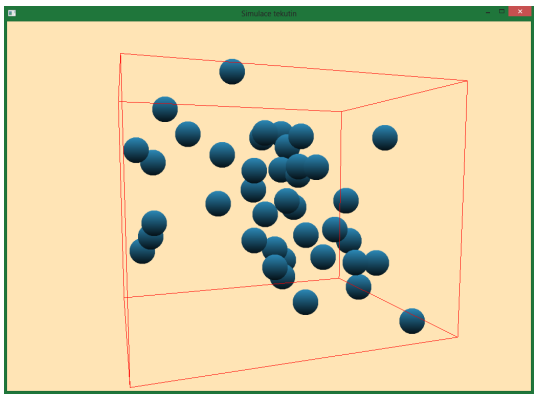
3.7.3 Odchylky

Při testování aplikace bylo zjištěno, že chování částic v některých případech neodpovídá chování reálného systému. Odchylky nejspíše nastávají při výpočtu tlakové síly. Částice se shlukují do menších samostatných útvarů. Možné vysvětlení je to, že částice zabírají malou část prostoru a snaží se vytvořit hustotu simulované kapaliny.

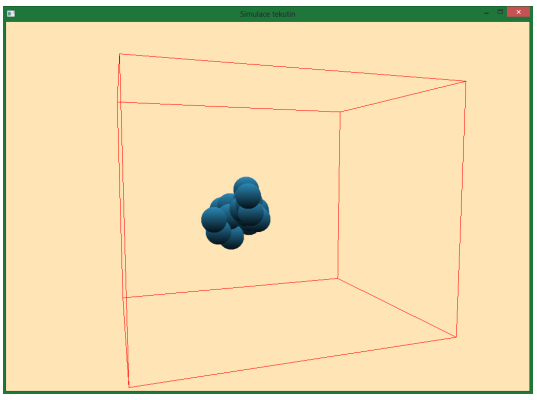
Vůbec nejreálnějšího chování kapaliny ovšem simulace nabývá při vypnuté tíhové síle. Částice tvoří shluky v prostoru, což evokuje chování kapalin ve stavu beztíže 3.8b.



Obrázek 3.7: Doba trvání jednoho kroku simulace při výpočtu jednotlivých sil.



(a) Spuštění simulátoru bez gravitace.



(b) Výsledek simulace bez gravitace.

Kapitola 4

Závěr

Cílem práce bylo prostudovat metody pro simulování tekutin, stav oboru a pokusit se implementovat řešení založené na těchto poznatcích. V průběhu tvorby této práce jsem nastudoval rozdíly mezi Eulerovou a Langrangeovou metodou, výhody SPH a seznámil se se základy programování na GPU s využitím CUDA Toolkitu.

Na základě testování ale tvrdím, že výsledná aplikace podává přibližně reálné výsledky pouze za určitých podmínek, například absence tíhové síly. V porovnání s jinými simulátory založenými na metodě SPH se ani nejedná o nijak výkonný simulátor. Pro to, aby simulace probíhala v reálném čase a uživatel mohl interagovat s částicemi, je nutná simulaci spustit s méně než dvaceti částicemi.

Čeho práce dosáhla je ale porovnání náročnosti výpočtu sil. V dalším vývoji, ať už aplikace nebo oboru obecně, se tak dá zaměřit právě na optimalizaci výpočtů časově nejnáročnějších prvků.

4.1 Možnosti pokračování

Problematika simulátorů tekutin má velký potenciál v budoucnosti. Ať už kvůli vědeckým či vojenským projektům, nebo kvůli hernímu a filmovému průmyslu. Výsledky této práce se však jen těžko dají uplatnit ve výše zmíněných odvětvích. Nicméně je zde užito několik algoritmů a mechanismů, které by se daly při podobných experimentech využít.

I přes to si ale uvědomuji několik nedostatků, které práce má.

4.1.1 Interaktivita simulátoru

Vyvinutá verze má velmi málo ovládacích prvků, které umožňují zasahovat do simulace. Nelze například vkládat pevné objekty do kontejneru, rušit stěny kontejneru za běhu aplikace a podobně.

V případě vkládání objektů do simulátoru by se řešení nejspíš nabízelo vytvořením dalšího simulátoru, na pevné objekty, který zajistí působení fyzikálních sil na tělesa. Může jít například o vložení kamenného kvádru, kde by bylo potřeba reagovat na silnější nárazy tekutiny posunem kvádru.

4.1.2 Obarvování částic

Simulátor v současnosti pracuje s jednotnou texturou 3.4.4 pro všechny částice. Tato textura by se však mohla měnit podle fyzikálních vlastností částice, například podle velikosti sil na

nic působících, nebo podle její rychlosti.

4.1.3 Přidávání částic

Snad největším omezením současné verze je konstantní počet částic, se kterými se pracuje. To je zapříčiněno zjednodušením práce s pamětí grafické karty. Aniž by byl měněn tento koncept, mohou být částice umístěné v jiném kontejneru, mimo scénu, a v případě příkazu pro přidání částic by se pouze přemístily z jednoho kontejneru do druhého. Opět ale narážíme na maximální počet částic v rámci jednoho spuštění aplikace.

4.1.4 Náročné operace

Byť jsem se tomu snažil vyhnout, nepodařilo se mi v určitých případech vyloučit užití časově náročných operací. Příkladem může být funkce `sqrt()` pro odmocninu. Tato funkce je užita pro počítání vzdáleností mezi částicemi nebo pro normalizaci vektorů.

Řešením by mohla být aproximace výpočtu. Otázkou ale zůstává, jak moc by taková aproximace urychlila běh programu.

4.1.5 Vykreslení tekutiny

V dnešní době existuje velká spousta programů, které vykreslují tekutiny metodou *Ray – Tracing*. Tato metoda dokáže velmi realisticky vykreslit povrch objektu, podle nastavených fyzikálních vlastností. Kvůli zaměření se na fyzikální stránku simulátoru, byla pro vykreslování implementována jen verze kulovitých částic. Dalším možným mezi krokem je vykreslení pomocí metody *margin cubes*.

Literatura

[1] A Unified Lagrangian Approach to Solid-Fluid Animation. http://graphics.ethz.ch/Downloads/Publications/Papers/2005/Kei05/Kei05.pdf, 2005.

[2] Auer, S.: *Realtime particle – based fluid simulation.*

[3] Bockmann, A.; Shipilova, O.; Skeie, G.: Incompressible SPH for free surface flows. *COMPUTERS & FLUIDS*, ročník 67, AUG 30 2012: s. 138–151, ISSN 0045-7930, doi:–10.1016/j.compfluid.2012.07.007″.

[4] Dursi, L. J.: Lagrangian methods and Smoothed Particle Hydrodynamics (SPH). 2006.

[5] Kontár, S.: *Real – Time Fluid Simulation.* Vysoké Učení Technické v Brně, Fakulta Informačních Technologií.

[6] Monoghan, J. J.: Smoothed particle hydrodynamics. *In: Annual review of astronomy and astrophysics.*, ročník 30 (A93-25826 09-90), 1992: s. 543–574.

[7] Müller, M.; Charypar, D.; Gross, M.: Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '03, Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2003, ISBN 1-58113-659-5, s. 154–159.
URL http://dl.acm.org/citation.cfm?id=846276.846298

[8] Novák, O.: *Simulace viskozních kapalin.* České vysoké učení technické v Praze, 2007.

[9] Weisstein, E. W.: Law of Reflection.
http://scienceworld.wolfram.com/physics/LawofReflection.html, 1996-2007.

Příloha A

Obsah CD

Přiložené CD obsahuje následující soubory:

`fluid_flow/include`	složka s funkcemi OpenGL a SDL
`fluid_flow/lib`	složka s knihovnami OpenGL a SDL
`fluid_flow/x64/Debug`	cíl pro přeloženou aplikaci, obsahuje dynamické knihovny
`fluid_flow/errors.h`	hlavičkový soubor pro chybové stavy
`fluid_flow/fluid_simulator.cuh`	hlavičkový soubor simulátoru
`fluid_flow/fluid_simulator.cu`	zdrojový soubor simulátoru
`fluid_flow/main.cpp`	hlavní soubor s funkcí `main()`
`fluid_flow/my_types.h`	definice vlastních typů
`fluid_flow/renderer.h`	hlavičkový soubor rendereru
`fluid_flow/renderer.cpp`	zdrojový soubor rendereru
`fluid_flow/shaders.h`	hlavičkový soubor shaderů
`fluid_flow/shaders.cpp`	zdrojový soubor shaderů

Příloha B

Ovládání

Aplikaci lze ovládat následujícími příkazy:

Tah `levým` tlačítkem myši	Rotace kamery
Tah `pravým` tlačítkem myši	Přiblížení, oddálení kamery
Tah `prostředním` tlačítkem myši	Posun simulátoru
Klávesa `'s'`	Upravení pozic částic do podoby koule
Klávesa `'d'`	Upravení pozic částic do náhodného rozmístění
Klávesa `'mezerník'`	Pozastavení, spuštění simulace
Klávesa `'r'`	Restartování simulátoru
Klávesa `'c'`	Změna barvy částic
Klávesa `'n'`	Navrácení kontejneru na střed okna