ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Očenášek** Jméno: **Michael** Osobní číslo: **398230**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Studijní obor: **Počítačová grafika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Simulace sopečných erupcí

Název diplomové práce anglicky:

Simulation of volcanic eruptions

Pokyny pro vypracování:

Prostudujte fyzikální procesy probíhající během sopečných erupcí a následného výlevu lávy. Proveďte rešerši existujících metod simulace sopek používaných v počítačové grafice [1-5] a porovnejte je z hlediska schopnosti simulovat jednotlivé fyzikální procesy doprovázející erupci sopek (proudění lávy a její chladnutí, dynamika erupce sopky, fázové přeměny, kouř, atd.).
Detailně nastudujte metodu [1], která je založena na částicovém systému a naimplementujte ji. Funkčnost a možnosti metody demonstrujte alespoň na třech různých scénách.
Implementaci proveďte v C/C++ s využitím OpenGL.

Seznam doporučené literatury:

[1] S. Zhang, F. Kong, C. Li, C. Wang, H. Qin: Hybrid modeling of multiphysical processes for particle□based volcano animation. Computer Animation and Virtual Worlds, 28(3-4), 2017.
[2] D. Stora, P.O. Agliati, M.P. Cani, F. Neyret, J.D. Gascuel: Animating lava flows. In Graphics Interface (GI'99) Proceedings, pp. 203-210, 1999.
[3] A. Hérault, G. Bilotta, A. Vicari, E. Rustico, C. Del Negro: Numerical simulation of lava flow using a GPU SPH model. Annals of Geophysics, 54(5), 2011.
[4] R. Mizuno, Y. Dobashi, T.Nishita: Volcanic smoke animation using cml. Proceedings of International Computer Symposium 2002, Vol. 2, pp. 1375-1382.
[5] R.Mizuno, Y. Dobashi, B.Y. Chen, T. Nishita: Physics motivated modeling of volcanic clouds as a two fluids model. Proceedings of the 11th Pacific Conference on Computer Graphics and Applications, pp. 440, 2003, IEEE Computer Society.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Jaroslav Sloup, Katedra počítačové grafiky a interakce

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **08.10.2019** Termín odevzdání diplomové práce: **22.05.2020**

Platnost zadání diplomové práce: **30.09.2021**

Ing. Jaroslav Sloup podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačové grafiky a interakce

Diplomová práce

Simulace sopečných erupcí

Bc. Michael Očenášek

Vedoucí práce: Ing. Jaroslav Sloup

22. května 2020

Poděkování

Rád bych poděkoval vedoucímu práce panu Ing. Jaroslavu Sloupovi za cenné rady a notnou dávku trpělivosti. Déle bych chtěl poděkovat své rodině za neochvějnou podporu při studiu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 22. května 2020 .....

Odkaz na tuto práci

Očenášek, Michael. *Simulace sopečných erupcí.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta elektrotechnická, 2020.

Abstrakt

Tato diplomová práce se věnuje návrhu a implementaci aplikace simulující chování tekoucí lávy v terénu. Návrh aplikace vychází z kombinace poznatků z existujících řešení na přidružená témata. Simulace probíhá za využití metody SPH pro simulaci kapalin a je implementovaná v jazyce C++. Grafická část aplikace je postavena na knihovně OpenGL. Simulace probíhá paralelně na CPU.

Klíčová slova SPH, simulace, vulkán, kapaliny, fyzikální procesy, OpenGL

Abstract

This master's thesis is devoted to designing and implementation of application which can simulate lava flow in terrain. Aplication design is based on combination of previously presented solutions on similar topics. The simulation uses SPH method to simulate lava flow and is implemented in C++. Graphical part of the aplication is build on the OpenGL library. The simulation utilizes parallel processing on CPU.

Keywords SPH, simulation, volcano, fluids, multiphysical processes, OpenGL

Obsah

Kapitola 1

Úvod

Roztavená hornina se chová jako kapalina, i když o ní pravděpodobně tak nepřemýšlíme. Láva sice není látka, se kterou bychom přišli běžně do styku, nicméně voda, kouř, či třeba sníh mezi ně rozhodně patří. Vše spojuje s ohledem k této práci fakt, že jejich chování lze simulovat pomocí tzv. *computational fluid dynamics* – algoritmů, či rovnic popisujících a zabývajících se chováním kapalných a plynných látek.

Většina těchto metod využívá Navier-Stokesovy rovnice, které popisují právě proudění Newtonovských kapalin. Simulaci kapalin můžeme z praktického hlediska rozdělit do dvou kategorií dle jejich využití. Jde o simulace, jejichž cílem je zcela přesné popsání a vypočtení pohybu kapaliny či plynu - jde o využití pro vědecké a inženýrské úkoly - např. při předpovědi počasí, simulaci povodní, či testování aerodynamiky kokpitu letadla. Do druhé kategorie se řadí využití, kde je hlavním cílem rychlost a uvěřitelnost - za cenu toho, že simulace není fyzikálně zcela přesná. Tyto metody se používají především v oblasti počítačové grafiky, jejich výsledky jsou dostatečné na to, aby byly pro pozorovatele uvěřitelné, a zároveň nejsou tak výpočetně náročné jako metody z první kategorie.

Tato práce se bude zabývat druhou z těchto kategorií.

1.1 Sopky

Pro začátek je vhodné definovat několik základních pojmů spojených se sopkami a v dalších částech práce určit, které jevy se budou v rámci této práce simulovat.

- **Sopka** – je narušený povrch planety, z kterého může na její povrch pronikat roztavená hornina z magmatického rezervoáru pod povrchem.
- **Magma** – tento termín označuje směs roztavených hornin a plynů (nejčastěji vodní páry, oxidy síry, uhlíku a další), které se vyskytují pod po-

vrchem planety. Vzniká v zemském plášti, nebo tavením hornin spodní zemské kůry.

- **Láva** – jakmile při sopečné erupci dojde k úniku magmatu na povrch planety, mluvíme o lávě.
- **Pyroklastiky** – jde o nesoudržné sopečné vyvrženiny. K jejich uvolňování dochází typicky ještě před výtokem lávy, společně s prvním uvolňováním sopečných plynů. Jde o sopečné balvany - dolétávají až stovky metrů daleko, sopečné pumy - "kameny"decimetrových rozměrů, sopečný písek – o velikosti milimetrů až centimetrů a sopečný popel, který dolétne až stovky kilometrů od sopky.
- **Sopečné plyny** – při úniku magmatu na povrch planety dochází k snížení tlaku působícího na magma a plyny, které jsou jeho součástí, mají možnost se uvolnit do okolního prostoru. Plyny uniklé z lávy označujeme jako sopečné plyny.

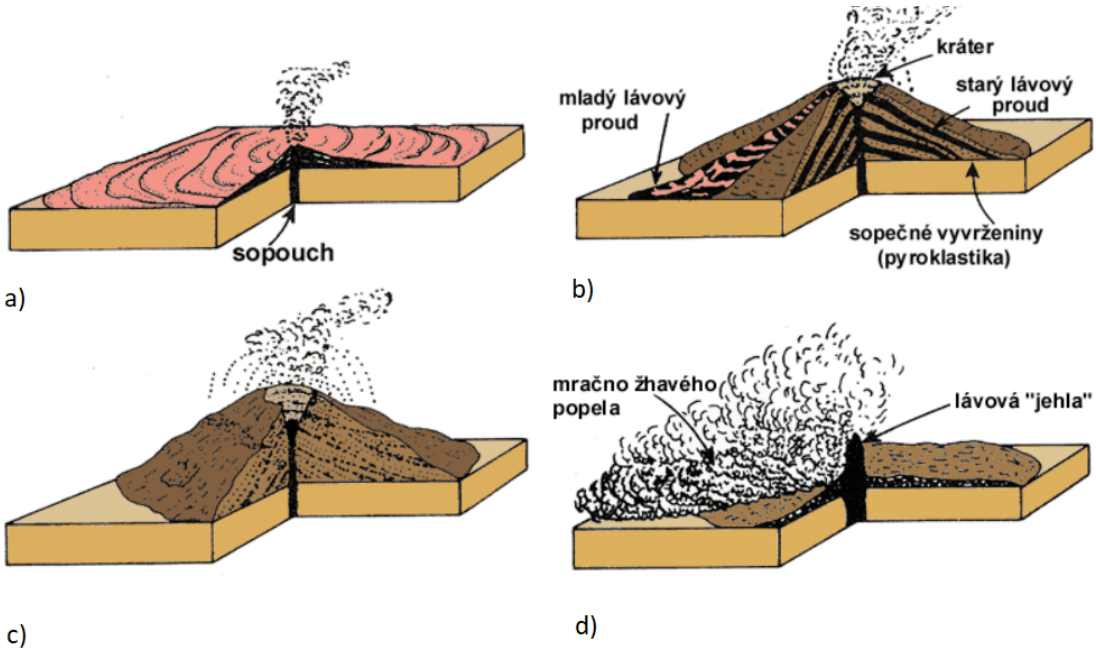
Mezi nejčastěji se vyskytující typy sopek dle [1] patří :

- **Havajský typ (štítová sopka)** – jedná se o více méně ploché sopky o velkém průměru budované vysoce tekutými bazaltovými lávami. Sopečné erupce nejsou hojné a také vzniká málo pyroklastik.
- **Strombolský typ (stratovulkán)** – sopka se sopečným kuželem, který je tvořen střídáním lávových proudů a vrstev nahromaděného pyroklastického materiálu.
- **Vulkánský typ** – produkovány jsou méně tekuté lávy, které jsou neustále rozrušovány výbuchy plynů, a jejich kužely se tedy skládají více méně z pyroklastik.
- **Peléský (katmajský) typ** – z kráteru je vytlačována velmi tuhá láva v podobě žhavé jehly a většinou také vznikají žhavá mračna sopečného popela, která se valí dolů po svahu sopky.

Tyto typy sopek můžeme vidět na obrázku 1.1. Sopky se dále dělí dle způsobu, jakým dochází k jejich erupci a obsahu magmatu, který při erupci uniká.

1.2 Cíl práce

Cílem této práce je za použití stávajících metod pro simulaci tekutin navrhnout a implementovat algoritmus, s jehož použitím by bylo možné simulovat proudění lávy v uzavřené scéně. Tato simulace by měla probíhat v reálném čase a krom proudění lávy také simulovat přesuny tepla mezi lávou, terénem a vzduchem, tuhnutí lávy při jejím toku a kouř, který erupci provází (uvolňování sopečných plynů). Při návrhu budeme vycházet z práce S. Zhanga, F. Konga



Obrázek 1.1: a) Havajský typ, b) Strombolský typ, c) Vulkánský typ, d) Pelélský typ [2]

a kolektivu, kteří se tímto problémem zabývali [3]. Výsledná aplikace by měla splňovat tyto body:

- Simulovat lávu pomocí aproximační metody.
- Umožnit uživateli simulaci sledovat a pohybovat se ve scéně.
- Simulace v reálném čase (a s tím spojená paralelizace výpočtů).
- Simulovat přesuny tepla a změny skupenství (pevné, kapalné, plynné).
- Simulovat tepelnou vodivost a viskozitu magmatu závislou na teplotě.
- Umožnit simulaci na rozsáhlé scéně (sopka a její okolí).

Seznam vlastností, které naopak nebudou simulovány, nebo budou nějakým způsobem zjednodušeny:

- Láva bude simulována pomocí částic jako jeden typ látky - budeme tedy zanedbávat, že by se uvnitř lávy mohly objevovat pevné částice/krystaly s jiným složením a jiným skupenstvím.
- Jednotlivé vlastnosti lávy budou definovány určením několika konstant, místo aby byly odvozené na základě procentuálního obsahu různých hornin a plynů.

- Při uvolňování sopečných plynů z lávy dochází ve skutečnosti k jakýmsi explozím, které roztrhnou tekutou lávu do okolí, čímž se láva prudce ochladí a může se změnit na pevnou látku. Tato simulace bude kouř generovat na základě teploty lávy, pouhým vygenerováním nových částic kouře, výsledný efekt nejspíš bude značně odlišný od skutečnosti.
- Tato práce se nebude zabývat pyroklastiky.

Kapitola 2

Způsoby simulace kapalin

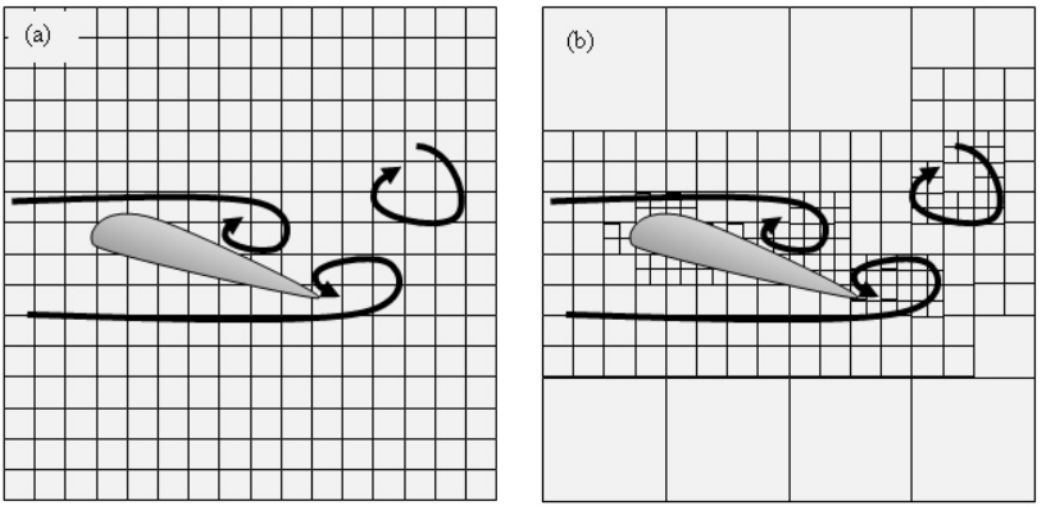
Existují dva hlavní přístupy k simulování kapalin používané v počítačové grafice. První z nich využívá mřížky – jde o tzv. Eulerovský přístup, druhý využívá částicových systémů – jde o tzv. Lagrangeův přístup. Jejich popis bude následovat.

2.0.1 Eulerovský přístup

Eulerovský přístup při specifikaci toku kapaliny bere v potaz, či sleduje, místa, kterými prochází tok kapaliny v čase. Laicky si lze představit, že sedíme na okraji řeky a sledujeme konkrétní místo a to, jak jím prochází proud vody. Při využití Eulerovského přístupu se nejčastěji setkáváme se simulací využívající uniformní mřížku. Pro každou buňku v prostoru si uchováváme rychlost (a směr) procházející kapaliny, hustotu, viskozitu, tlak, případně další vlastnosti. Je definován časový krok, o který se simulace při přepočtu posunuje. K posunu kapaliny typicky dochází tak, že se vypočte změna vlastností na základě hodnot sousedních buněk, a tento výpočet se provede opakovaně pro jeden časový krok – tím dojde k tomu, že jedna buňka je schopna svými vlastnosmi ovlivnit i buňky, které nejsou přímo sousedící. Některá řešení využívající pro urychlení také neuniformní (adaptivní) mřížky – hustota mřížky se lokálně mění, podle toho, jestli se v dané buňce nějaká kapalina vyskytuje. Náhled těchto mřížek je znázorněn na obrázku 2.1.

Pro uniformní mřížky (které jsou jednodušší na implementaci a tedy častěji používané) platí:

- Jsou velice paměťově náročné, náročnost roste s hustotou mřížky. Kvůli tomu nejsou obecně vhodné pro simulaci větších scén.
- Jsou stabilní i pro větší časové kroky (v porovnání s Lagrangeovou metodou)
- Hlavním problémem je, že se provádí výpočty i s buňkami, ve kterých se žádná kapalina nenachází.



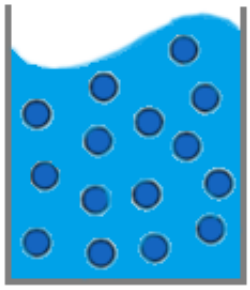
Obrázek 2.1: a) Uniformní mřížka, b) Adaptivní mřížka

Mřížkový systém využil například Jos Stam ve své práci zabývající se simulacemi kouře a ohně pro počítačové hry [4].

2.0.2 Lagrangeův přístup

Lagrangeův přístup či pohled na kapalinu se místo sledování místa zaměřuje na sledování toku. Tento přístup odpovídá tomu, jako kdybychom sledovali trasu, kterou urazí jednotlivá molekula např. vody v daném čase. Případně, že sedíme v lodi a necháváme se unášet proudem – loď zde odpovídá sledované molekule vody. Při simulaci využívající Lagrangeův přístup se využívá systému volně se pohybujících částic. Kapalina je modelována pomocí množiny částic – u každé z ní si pamatujeme, stejně jako u buněk mřížky, její rychlost, hustotu, viskozitu, tlak atd. Hlavním rozdílem je, že přibude informace o pozici částice.

Tento způsob umožňuje simulaci na rozsáhlých scénách, protože náročnost simulace je závislá na počtu částic, nikoliv na velikosti scény (zde se pouze určí okraje, abychom nějakým způsobem uzavřeli prostor simulace). Působení částic mezi sebou, a tím jejich posun, se pro jednotlivé částice vypočítává tak, že se naleznou všechny částice v okolí do nějaké hranice a jejich vliv na danou částici je úměrný vzdálenosti od ní. Tím se dostáváme k důležitému rozdílu mezi Erulerovým a Lagrangeovým přístupem. U druhého z nich se složitost výpočtů může v čase měnit, podle toho, jak blízko či daleko jsou částice od sebe, zatímco u Eulerova přístupu je složitost při použití uniformní mřížky stále stejná. Znázornění částicového systému je na obrázku 2.2.



Obrázek 2.2: Nádoba s vodou jako částicový systém.

2.1 Navier-Stokesovy rovnice

Navier-Stokesovy rovnice jsou základním kamenem pro téměř veškeré výpočty zabývající se chováním kapalin. Kapaliny definuje, a v těchto rovnicích se objevuje, několik veličin:

- *Rychlost* - Jakou rychlostí a jakým směrem se kapalina v daném bodě pohybuje.
- *Tlak* - Jak velký tlak působí na kapaliny v daném bodě a jakým tlakem kapalina působí na svoje okolí.
- *Hustota* - Podíl hmotnosti a objemu tělesa. Hustota se při stlačování a roztahování kapaliny mění a může být různá v různých bodech.
- *Viskozita* - určuje vnitřní tření v kapalině, a její schopnost a rychlost s kterou reaguje na změnu působících sil.
- *Externí síly* - Vnější síly působící na kapalinu - zejména jde o gravitační sílu a síly, které na kapalinu působí v případě kontaktu s překážkou.

Samotné rovnice popisují změnu rychlosti kapaliny v bodě v závislosti na čase. Jde o rovnice pohybu 2.1 a zachování hmoty 2.2.

$$\frac{du}{dt} = -(u \cdot \nabla)u - \frac{\nabla p}{\rho} + \frac{\mu}{\rho}\nabla^2 u + \frac{F_{ext}}{\rho} \tag{2.1}$$

Rovnice pohybu 2.1 – **u** označuje rychlost kapaliny v určitém bodě (obsahuje i směr jejího působení, jde o d složkový vektor, kde d je dimenze simulovaného prostoru), **t** je čas v sekundách. ρ je hustota kapaliny v bodě, **p** je tlak v bodě, μ určuje viskozitu kapaliny. F_{ext} určuje externí síly které na kapalinu v daném bodě působí. První část levé strany rovnice $(u \cdot \nabla)u$ je takzvané konvektivní zrychlení, které definuje změnu rychlosti toku v závislosti na pozici v toku (rychlost se může lokálně měnit, například z důvodu, že v daném místě je řeka zúžená).

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

Rovnice zachování hmoty 2.2 – tato rovnice je jakousi podmínkou, kterou musí rovnice 2.1 splňovat. Zaručuje stálost celého systému – zachování objemu a hmoty kapaliny.

2.2 Dostupná řešení pro simulaci lávy

Erupcím vulkánů se věnuje mnoho prací z rozdílných vědních oborů. Pokud se jedná o snahu tento proces simulovat, rozdíly jsou zejména v tom, které aspekty spojené se sopečnou činností jsou jednotlivé numerické modely schopny simulovat. Například Keszthelyi [5] ve své práci demonstroval, jak je rychlost a objem toku závislý na reologických (obor věnující se deformačním vlastnostem látek) vlastnostech, ochlazování lávy, její krystalizací [6] atd. Většina předešlých prací je závislá na empiricky získaných datech a rovnicích z nich odvozených a nejsou využitelné pro obecné případy. Některé práce se místo simulování toku zaměřují pouze na předpověď trasy toku lávy a odhad rizik s nimi spojenými [6]. Pokud jde o práce, které se snaží o simulaci na základě fyzikálních vlastností, většina z nich využívá SPH (*Smoothed particle hydrodynamics*), což je Lagrangeovská metoda. Například v [7] můžeme nalézt metodu založenou na SPH, která se zabývá simulací lávy, ale zcela ignoruje generování kouře. Stomakhin a kolektiv potom představili práci využívající *augmented material point (MPM)* metodu. Jejich práce přináší kvalitní výsledky z pohledu simulace změny skupenství lávy z kapalného na pevné. Na tuto práci dále navázal Jiang a kol [8]. Tyto metody ovšem nesimulují ostatní jevy spojené s vulkanickou aktivitou, ať už jde o samotnou erupci, simulaci kouře, změny skupenství (ne jen tuhnutí) atd. Tato práce se inspiruje prací Zhanga a kol. [3], jejichž metoda všechny tyto aspekty simuluje. Na obrázku 2.3 můžeme vidět porovnání simulačních možností jednotlivých prací (dle práce [3]).

Recent works	Model	State transition	Smoke	External interaction	Rendering results
Stora et al.	SPH	No	No	Single	Good
Negro et al.	CNNs	No	No	Single	Only numerical results
Stomakhin et al. and Jiang et al.	MPM	Yes	No	Single	Better
Our method	SPH	Yes	Yes	Multiphysics	Better

Note. CNN = cellular nonlinear network; MPM = material point method; SPH = smoothed particle hydrodynamics.

Obrázek 2.3: Porovnání simulačních metod dle [3].

Kapitola 3

SPH

3.0.1 Obecný úvod

Smoother particle hydrodynamics, zkráceně SPH, je metoda která má počátek v roce 1977 v práci R. A. Gingolda a J. J. Monaghan [9] a v práci L.B. Lucy [10] a původně byla vyvinuta pro použití v astrofyzice, konkrétně pro simulaci toku mezihvězdných plynů. Základem této Lagrangeovské metody je snaha převést spojitý fyzikální jev (např. pohyb kapaliny) na diskrétní pomocí systému konečného počtu částic. Je však použitelná pro simulaci mnoha fyzikálních fenoménů; v této práci se budeme ale zmiňovat zejména o jejím možném použití při simulaci kapalin. Každá částice v tomto systému simuluje chování a vlastnosti kapaliny v blízkém okolí, které má reprezentovat (např. jedna částice simuluje jeden centimetr krychlový reálné kapaliny). Každá částice nese informace o vlastnostech kapaliny v blízkém okolí jako je hustota, tlak, teplota, působící síly a další. Samotná simulace probíhá pomocí výpočtů založených na Navier-Stokesových rovnicích, dochází ale k diskretizaci a simulace probíhá v časových krocích, při každém přírůstku času jsou znovu přepočítány vlastnosti jednotlivých částic a na základě těch je vypočtena nová pozice pro každou částici.

Základem SPH metody je tzv. *Integrální interpolace* [11]. Integrovaná hodnota **A** je pro každý bod prostoru **r** definována jako:

$$A(r) = \int A(r')W(r - r', h)dr', \tag{3.1}$$

kde **W** je jádrová funkce (anglicky kernel function), která má vlastnosti, za prvé normovanost:

$$\int W(r - r', h)dr' = 1, \tag{3.2}$$

a za druhé sudost:

$$W(r,h) = W(-r,h). \tag{3.3}$$

Při provádění výpočtů a simulací na počítači je však nutno aproximovat interpolaci pomocí sumy:

$$\mathbf{A}(\mathbf{r_i}) = \sum_j \frac{m_j}{\rho_j} \mathbf{A}(\mathbf{r_j}) W(|\mathbf{r_i} - \mathbf{r_j}|, h), \tag{3.4}$$

kde $\mathbf{A}(\mathbf{r_i})$ určuje hodnotu vlastnosti částice, $\mathbf{r_i}$ určuje pozici částice $\mathbf{i}$, $\mathbf{m_i}$ určuje hmotnost, $\boldsymbol{\rho_i}$ určuje hustotu a $\mathbf{W}(|\mathbf{r_i} - \mathbf{r_j}|, \mathbf{h})$ určuje velikost vlivu částice $\mathbf{j}$ na částici $\mathbf{i}$. Síla vlivu je vypočtena pomocí jádrové funkce $\mathbf{W}$.

Výhodou tohoto přístupu je, že pokud potřebujeme získat gradient $\nabla\mathbf{A}$ či laplaceův operátor $\nabla^\mathbf{2}\mathbf{A}$, stačí znát $\nabla\mathbf{W}$, resp. $\nabla^\mathbf{2}\mathbf{W}$.

3.0.2 Jádrová funkce

Jádrová funkce má klíčovou úlohu ve výpočtu síly interakce mezi částicemi v závislosti na jejich vzdálenosti. Výběr jádrové funkce zásadně ovlivňuje průběh simulace a její kvalitu. Jádrová funkce je ve tvaru:

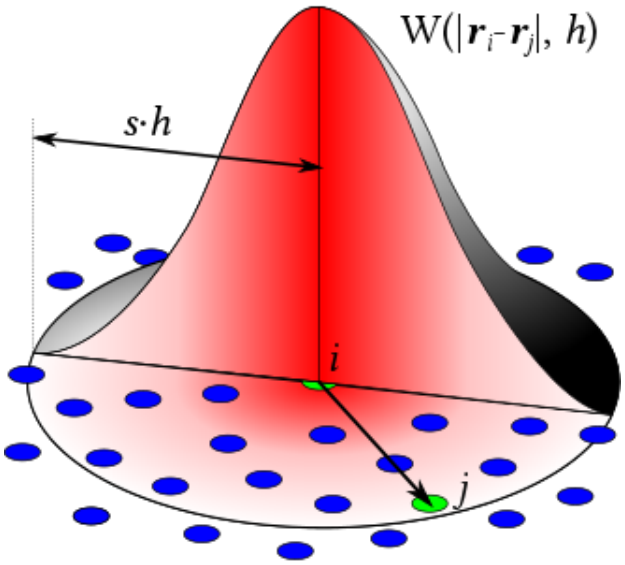
$$\mathbf{W}(\mathbf{r_{ij}}, \mathbf{h}), \tag{3.5}$$

kde $\mathbf{r_{ij}}$ označuje vzdálenost dvou částic a $\mathbf{h}$ označuje vyhlazovací vzdálenosti (anglicky smoothing length) viz rovnice 3.1. Jádrová funkce by měla být navržena v závislosti na $\mathbf{h}$ a čím blíže je hodnota $\mathbf{r_{ij}}$ k $\mathbf{h}$, tím by se měla snižovat hodnota funkce, s tím, že při $\mathbf{r_{ij}} \geq \mathbf{h}$ by měla být hodnota rovna nule. Klesající tendence funkce se vzrůstajícím $\mathbf{r_{ij}}$ zajišťuje, že vzdálenější částice se vzájemně méně ovlivňují v porovnání s částicemi, které jsou k sobě blíže. Nulová hodnota při $\mathbf{r_{ij}} \geq \mathbf{h}$ umožňuje vyřadit z výpočtů pro danou částici všechny ostatní částice, které jsou dále než $\mathbf{h}$. Pro zrychlení výpočtů pomocí rozdělení prostoru do mřížky můžeme také rozdělit prostor v závislosti na velikosti $\mathbf{h}$ a k hledání možných sousedů částice projít pouze několik okolních buněk mřížky (víme, že všechny ostatní buňky již jsou moc daleko, aby mohly obsahovat částice, které budou blíže než $\mathbf{h}$). V této práci jsou využity jádrové funkce vycházející z práce Müllera [13]. Pro výpočet hustoty je použita funkce:

$$W_{poly6}(r_{ij}, h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - r_{ij}^2)^3 & 0 \leq r_{ij} \leq h \\ 0 & \text{v ostatních případech} \end{cases}. \tag{3.6}$$

Pro výpočet viskozity je použita funkce:

$$\nabla^2 W_{spikyV}(r_{ij}, h) = \frac{45}{\pi h^6} \begin{cases} (h - r_{ij}) & 0 \leq r_{ij} \leq h \\ 0 & \text{v ostatních případech} \end{cases}. \tag{3.7}$$



Obrázek 3.1: Ilustrace souvislosti mezi jádrovou funkcí, vyhlazovací délkou a pozicemi částic [12]

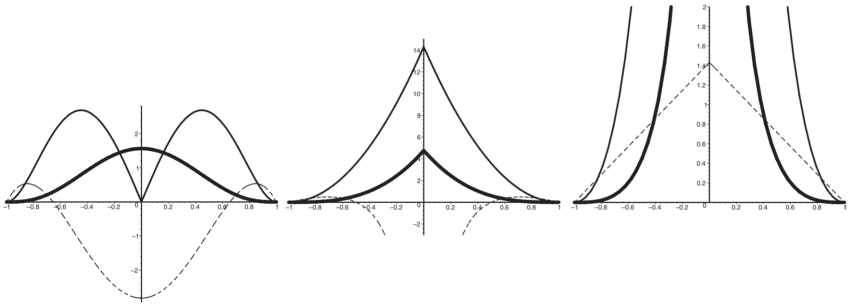
Pro výpočet tlaku je použita funkce:

$$\nabla^2 W_{spikyP}(r_{ij}, h) = \frac{-45}{\pi h^6} \begin{cases} (h - r_{ij}) & 0 \leq r_{ij} \leq h \\ 0 & \text{v ostatních případech} \end{cases}. \tag{3.8}$$

V posledních dvou případech je uveden rovnou laplacián funkce, protože pouze ten je skutečně využit při výpočtech potřebných při simulaci. Velkou výhodou funkce pro výpočet hustoty je, že stačí znát druhou mocninu vzdálenosti mezi částicemi – výpočet odmocniny je relativně drahá operace z pohledu procesorového času. Jádrovou funkci použitou pro výpočet hustoty nelze použít pro výpočet tlaku. Při simulaci tlaku požadujeme, aby tlak narůstal se snižující se vzdáleností mezi částicemi (chceme, aby se velmi blízké částice odpuzovaly), ale derivace funkce (3.6) se pro $\mathbf{r_{ij}}$ blízké nule blíží k nule, což tomuto požadavku neodpovídá. Na obrázku 3.2 můžeme vidět jádrové funkce z práce Müllera [13]. Z obrázků je zřejmé, že jednotlivé funkce se od sebe značně liší. Tyto funkce jsou vytvářeny pro specifické použití. Většina prací sice často přebírá již vyzkoušené a osvědčené jádrové funkce, neznamená to ale, že by tyto funkce s těmito hodnotami byly jediné použitelné. Jádrové funkce se mohou značně lišit, a v některých případech je nutno vytvořit si vlastní.

3.1 Průběh algoritmu a jeho typy

Existuje mnoho různých typů algoritmů pro simulování kapalin či plynů založených na SPH, jež se liší svým průběhem. Rozdíly mezi jedněmi ze známějších



Obrázek 3.2: Ukázka jádrových funkcí z [13]. Jde o funkce (zleva) pro hustotu, tlak a viskozitu. Plná tlustá čára znázorňuje jádrovou funkci, plná tenká čára znázorňuje gradient a přerušovaná čára laplacián. Pozn.: Grafy mají rozdílná měřítka a jde o průmět funkcí do jedné dimenze.

z nich,

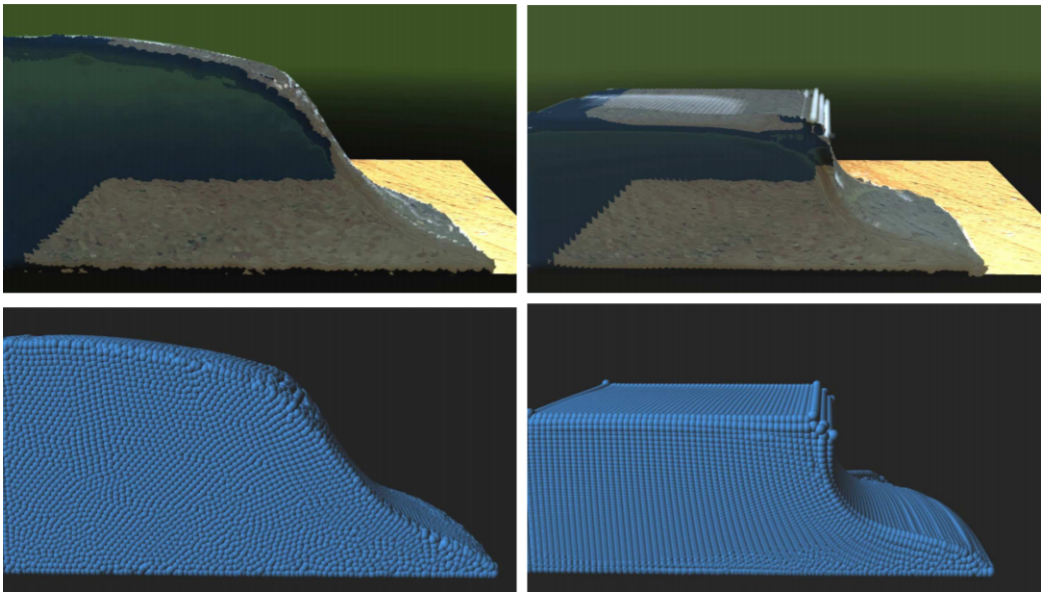
- Müllerovo SPH [13]
- WCSPH [14]
- PCISPH [15],

můžeme najít zejména mezi způsobem, jakým se vypořádávají se stlačitelností simulované látky.

Klasické SPH podle Müllera využívá pro výpočet tlaku zákonu ideálního plynu (více v další kapitole). Výpočet na základě toho zákona je velice jednoduchý, ale dovoluje simulované látce, aby byla poměrně znatelně stlačena, což není vlastnost, která odpovídá chování skutečných kapalin (ty jsou na rozdíl od plynů téměř nestlačitelné – při fyzikálních výpočtech se často můžeme setkat se zanedbáním stlačitelnosti kapalin, kdy se k nim přistupuje jako ke zcela nestlačitelným).

Müllerův přístup při výpočtu hustoty a tlaku sčítá přírůstky z okolních částic. Tento přístup má ale za následek, že částice na povrchu kapaliny nemají zcela správnou hustotu a tlak, protože nemají dostatek sousedních částic k jejich výpočtu. Tento problém společně s použitím zákona ideálního plynu má za následek nepřesnost výpočtu, která se odrazí ve vizuální kvalitě simulované látky.

Weakly compressible SPH (WCSPH), neboli lehce stlačitelné SPH, je metoda poprvé popsána v práci Beckera [14]. Cílem této metody je právě omezení stlačitelnosti simulované látky. Jedním ze způsobů jak toho dosáhnout, je za pomoci Taitovy rovnice pro výpočet tlaku, která byla poprvé využita



Obrázek 3.3: Porovnání mezi WCSPH (vlevo) a Müllerým SPH. Můžeme vidět, že v prvním případě nedojde k takovému stlačení kapaliny a jednotlivé částice se rozprostřou více do okolí [14].

v SPH v práci Monaghana [16]. Rozdíly ve výsledcích těchto dvou metod jsou na obrázku 3.3.

Na obrázku 3.4 můžeme vidět pseudokód pro simulaci SPH a WCSPH (**i** značí jednotlivé částice). Nejdříve dojde k nalezení sousedů pro každou částici na 3. řádce (případně mohou být hledány opakovaně v jednotlivých krocích algoritmu), následně dojde k výpočtu hustoty a tlaku (řádky 4, 5, 6), na 8. řádce dochází k výpočtu působících sil (tlaková síla, viskozita, gravitační síla a další externí síly, např. tření, povrchové napětí atd.). Poslední tři řádky představují tzv. **integrační krok**, při němž se na základě působících sil upraví rychlost částic a následně se vypočte jejich nová pozice.

Predictive-corrective incompressible SPH (PCISPH), neboli prediktivně–korektivní nestlačitelné SPH, je algoritmus představený Solenthalerem v práci [15]. Pseudokód algoritmu můžete vidět na obrázku 3.5. V tomto algoritmu se relativní nestlačitelnost kapaliny (maximální stlačitelnost kapaliny závisí na proměnné) vynucuje pomocí opakovaných cyklů výpočtu (řádky 8 až 17), které pracují s odhadem, kde budou částice v příštím kroku simulace (řádky 9 až 11). Pro tyto částice (posunuté o krok simulace vpřed) se potom vypočte odhad budoucí hustoty (řádek 13). Následuje výpočet odchylky hustoty od referenční hodnoty (řádek 14) a úprava velikosti tlaku na základě odchylky hustoty (řádek 15). Po vypočtení tlakové síly se vyhodnotí, zda je maximální odchylka hustoty menší než stanovený limit (nebo jestli ještě neproběhl minimální počet iterací), pokud ne, probíhá výpočet znovu (řádky 8 až

Algorithm 1 SPH / WCSPH

```
1  while animating do
2     for all i do
3        find neighborhoods N_i(t)
4     for all i do
5        compute density ρ_i(t)
6        compute pressure p_i(t)
7     for all i do
8        compute forces F^{p,v,g,ext}(t)
9     for all i do
10       compute new velocity v_i(t + 1)
11       compute new position x_i(t + 1)
```

Obrázek 3.4: Pseudokód SPH a WCSPH algoritmu [15].

17). Algoritmus de facto v opakované části posouvá částice takovým směrem, aby se jejich hustota vždy blížila referenční hodnotě. Aby se předešlo problémům s částicemi s nedostatkem sousedů, využívá se při výpočtech tlaku předpočítaná hodnota vycházející z imaginární částice s ideálním počtem sousedů. Výhodou tohoto algoritmu je, že poskytuje výsledky srovnatelné s WCSPH, ale stačí mu k tomu větší časový krok mezi jednotlivými kroky simulace – algoritmus je tedy dle autorů výrazně rychlejší, při zachování obdobné kvality simulace. Zdrojem případných nepřesností v simulaci může být fakt, že algoritmus nepřepočítává sousedy pro částice v každém kroku korektivní části simulace, ale pracuje s původními sousedy.

3.1.1 Integrační krok

Integračním krokem se označuje poslední část simulace, která na základě vypočtených sil upraví rychlost a následně pozici částic, a tím dokončí jeden krok simulace. Nejznámější řešení integračního kroku je Eulerova integrační metoda a tzv. Leap frog metoda (lze volně přeložit jako skákající žába).

Eulerova integrační metoda (viz. obr 3.7)paralelně a o stejný časový krok Δt (v angl. literatuře označován jako timestep), upraví rychlost i pozici částic viz (3.9):

$$\begin{aligned} \vec{v}(t + \Delta t) &= \vec{v}(t) + \frac{\partial \vec{v}}{\partial t}(\Delta t) \\ X(t + \Delta t) &= X(t) + \Delta t \vec{v}(t + \Delta t), \end{aligned} \tag{3.9}$$

kde $\vec{v}(t)$ a $X(t)$ značí rychlost a pozici v čase t.

Algorithm 2 PCISPH

```
1  while animating do
2     for all i do
3        find neighborhoods N_i(t)
4     for all i do
5        compute forces F^{v,g,ext}(t)
6        initialize pressure p(t) = 0.0
7        initialize pressure force F^p(t) = 0.0
8     while (ρ*_err(t+1) > η) || (iter < minIterations) do
9        for all i do
10          predict velocity v*_i(t+1)
11          predict position x*_i(t+1)
12       for all i do
13          predict density ρ*_i(t+1)
14          predict density variation ρ*_err(t+1)
15          update pressure p_i(t)+= f(ρ*_err(t+1))
16       for all i do
17          compute pressure force F^p(t)
18    for all i do
19       compute new velocity v_i(t+1)
20       compute new position x_i(t+1)
```

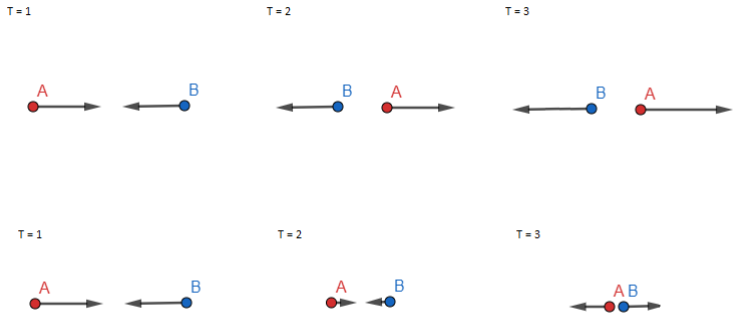
Obrázek 3.5: Pseudokód PCISPH algoritmu [15].

Integrace pomocí Eulerovy metody se může stát nestabilní, její stabilita souvisí s velikostí časového kroku. V případě, že se částice pohybují moc rychle, může dojít k tomu, že na sebe při jednom integračním kroku vůbec nepůsobí, protože jsou mimo vyhlazovací vzdálenost, ale při dalším jsou těsně vedle sebe, nebo sebou dokonce proletěly (potom by došlo k jejich zdánlivému odražení na opačnou stranu než by bylo správně (částice sebou proletí a následně je odpudivá síla ještě urychlí v jejich původním směru), viz obr. 3.6). Aby se předešlo těmto problémům, někteří autoři udávají limitace maximální velikosti časového kroku. Například [17] navrhuje, aby byl maximální časový krok proporcionální vyhlazovací délce vypočten takto:

$$\Delta t = min\frac{h}{c}. \tag{3.10}$$

Další způsoby stanovení či omezení časového kroku lze najít v [18], [19].

Metoda Leap frog (viz. obr 3.8) vypočítává rychlost a pozici jednotlivých částic o polovinu časového kroku od sebe – odtud pochází její název, kdy nejsou hledané hodnoty nikdy počítány pro stejný okamžik. Výpočet v mezikrocích zvyšuje přesnost simulace. Tato metoda je podobně jako Eulerova jednoduchá



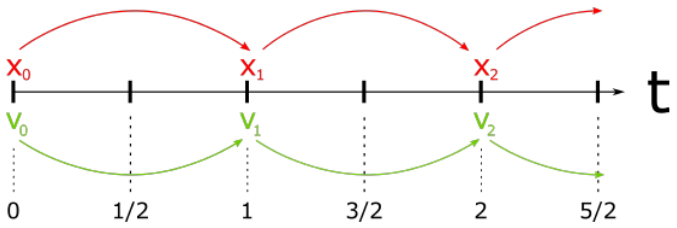
Obrázek 3.6: Porovnání možného chování pří příliš velkém časovém kroku (nahoře) a dostatečném časovém kroku (dole).

na implementaci, což napomáhá jejímu častému použití. Výpočet rychlosti a pozice probíhá takto:

$$\begin{aligned} \vec{v}(t + \frac{\Delta t}{2}) &= \vec{v}(t - \frac{\Delta t}{2}) + \frac{\partial \vec{v}}{\partial t}(\Delta t) \\ X(t + \Delta t) &= X(t) + \Delta t \vec{v}(t + \frac{\Delta t}{2}). \end{aligned} \tag{3.11}$$

Pro zahájení integrace je třeba pomocí Eulerovy metody získat $\vec{v}(\frac{-1}{2})$:

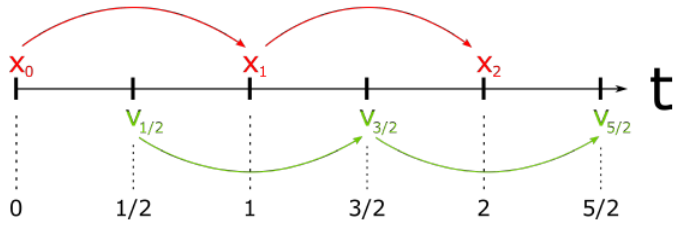
$$\vec{v}(\frac{-1}{2}) = \vec{v}(0) - \frac{\partial \vec{v}}{2\partial t}(\Delta t). \tag{3.12}$$



Obrázek 3.7: Eulerova integrace.

3.2 Simulované vlastnosti lávy

V této kapitole bude popsáno, které vlastnosti a chování lávy jsou v rámci této práce simulovány, a budou popsána možná řešení pro simulaci daných vlastností které jsou popsány v literatuře. Některé z nich jsou běžnou součástí simulací tekutin (nejčastěji vody), některé jsou pro simulování lávy částečně specifické. Jde především o:



Obrázek 3.8: Leapfrog integrace.

- Hustotu
- Tlak
- Viskozitu
- Teplotu
- Skupenství
- Tření

Je důležité si uvědomit, že simulace, kterou se snaží tato práce vytvořit, nemá za cíl být zcela přesná z fyzikálního hlediska, což platí i o spoustě podobných simulací jiných látek na základě SPH. Jejich cílem bývá, aby výsledný dojem ze simulace působil co nejreálněji – proto se například při simulaci vody může vynechat simulace tření s překážkami, protože tato síla je relativně zanedbatelná, ale při simulaci kapaliny jako je láva už její zanedbání není možné a je třeba ji simulovat.

Při výpočtu většiny hodnot/vlastností v této simulaci, které nejsou konstantní, obecně platí, že pro každou částici se hodnota vypočte na základě přírůstků získaných z okolních částic.

Na rozdíl od simulace vody, jejíž základní simulace může k předvedení své kvality zůstat pouze uzavřená do hraničního kvádru, k simulaci lávy je zapotřebí dalších částic. Krom simulace lávy samotné se jedná o statické částice reprezentující povrch, po kterém se láva pohybuje, a dynamické částice kouře, které jsou obvykle generovány při dosažení dostatečné teploty. Více o nich bude zmíněno v části věnované skupenství.

3.2.1 Hustota

Výpočet hustoty je základním a nejspíše nejjednodušším krokem simulace. Pro každou částici se pouze připočte přírůstek na základě vzdálenosti částic pomocí W_{poly6} (3.6):

$$\rho_i = m_i \sum_j W_{poly6}(r_{ij}, h). \tag{3.13}$$

Problém, s kterým se lze při tomto výpočtu setkat, je nedostatek sousedních částic. Pokud je částice osamostatněná v prostoru, bude její hustota nulová, v případě malého počtu sousedů bude potom informace nepřesná. V případě, že částice nemá žádného souseda, je vhodné nastavit jako její hustotu hodnotu rovnající se standardní hustotě v klidu pro danou kapalinu (můžeme se tak např. vyhnout chybám způsobených dělením nulovou hustotou).

3.2.2 Tlak

Tlak působící na částici se vypočte s pomocí dříve získané hustoty. Müller [13] ve své práci při výpočtu tlaku vychází ze zákona ideálního plynu:

$$p = k_0 \rho, \tag{3.14}$$

kde $\mathbf{k_0}$ je hodnota měnící se v závislosti na množství plynu a jeho teplotě. K samotnému výpočtu Müller využívá upravenou variantu navrženou v [20] :

$$p_i = k_0(\rho_i - \rho_{rest}), \tag{3.15}$$

kde $\mathbf{p_i}$ je tlak působící na částici $\mathbf{i}$, $\boldsymbol{\rho_i}$ je hustota částice $\mathbf{i}$ a $\boldsymbol{\rho_{rest}}$ je konstantní hustota pro danou kapalinu v klidu a $\mathbf{k_0}$ je konstanta, kterou autor nastavuje dle simulované tekutiny.

Další často využívanou rovnicí pro výpočet tlaku je Taitova stavová rovnice 3.16, jejíž první použití v SPH můžeme najít v [21] (první vydání z roku 1967)

$$p_i = b((\frac{\rho_i}{\rho_{rest}})^\alpha - 1), \tag{3.16}$$

kde b udává konstantu, jejíž hodnota závisí na rychlosti šíření zvuku v simulované kapalině [16], α je nastavitelná konstanta, obvykle v rozmezí 3-7.

Vzhledem k podmínce relativní nestlačitelnosti kapaliny při použití PCISPH se k výpočtu tlaku váže ještě jeho upravení, aby se tato podmínka zajistila. Po odhadu hustoty se vypočte odchylka odhadované hustoty 3.17 od ideální hustoty v klidu a také se vypočte referenční hodnota γ, která simuluje ideální částici s okolím zaplněným dostatkem sousedů. Výsledný tlak je v jednotlivých iteračních krocích dopočítáván pomocí rovnice 3.18. Iterace probíhají, dokud není největší z odchylek menší než nejvyšší přijatelná míra stlačitelnosti kapaliny (nastavitelná hodnota) [15].

$$\rho_{err_i} = \rho_i - \rho_0 \tag{3.17}$$

$$p_i += \gamma \rho_{err_i} \tag{3.18}$$

3.2.3 Teplota

Při simulování teploty jde především o simulovaní přenosu tepla mezi jednotlivými částicemi. Ať už jde o částice lávy samotné, nebo pevné částice reprezentující povrch. V práci [7] se vychází z Furierova zákona o vedení tepla:

$$q = -k\nabla T, \tag{3.19}$$

kde **q** značí hustotu tepelného toku, **k** součinitel tepelné vodivosti a $\nabla\mathbf{T}$ gradient teploty. Fourierův zákon říká, že vektor hustoty tepelného toku q je úměrný gradientu teploty ∇t a má opačný směr. Rovnice je potom pro potřeby SPH upravena do tvaru:

$$\frac{\partial T}{\partial t} = k\nabla T. \tag{3.20}$$

Autoři podotýkají, že "přímý výpočet laplaciánu teploty je složitý z důvodu vyhlazovacího efektu jádrové funkce a v praxi se nejdříve vypočte gradient a z něj se následně vypočte laplacián pomocí rovnic":

$$\nabla T_i = \sum_{j!=i} m_j \frac{T_j - T_i}{\rho_j} \nabla W_{ij} \tag{3.21}$$

$$\Delta T_i = \sum_{j!=i} m_j \frac{\nabla T_j}{\rho_j} \nabla W_{ij}. \tag{3.22}$$

V práci [3] autoři vycházejí ze stejných základů, ale jejich úprava pravé části rovnice 3.20 pro potřeby SPH se liší a to:

$$\begin{aligned} \Delta(k_i T_i) = 2 \sum_{j!=i} \frac{m_j}{\rho_j} (k_j T_j - k_i T_i) \frac{x_{ij} \cdot \nabla W_{ij}}{x_{ij} \cdot x_{ij} + 0.01h^2} \\ + 2 \sum_{b} \frac{m_b}{\rho_b} (k_b T_b - k_i T_i) \frac{x_i b \cdot \nabla W_{ib}}{x_{ib} \cdot x_{ib} + 0.01h^2}, \end{aligned} \tag{3.23}$$

kde $\mathbf{k_i}$ značí dynamickou hodnotu tepelné vodivosti, která se mění na základě teploty a skupenství dané částice, **j** značí částici lávy, **b** částici statickou (povrch sopky), $\mathbf{x_i}$ pozici částice **i**.

3.2.4 Viskozita

Viskozita představuje vnitřní tření látky. Látky s vyšší viskozitou mají menší tendenci podléhat deformaci na základě působení okolních sil. Pro výpočet síly působící na částici kvůli vnitřnímu tření se dle Müllera používá vzorec vycházející pouze z rozdílu rychlosti částic a jejich koeficientu viskozity [13]. Konkrétně:

$$f_i^{visc} = \mu \sum_{j!=i} m_j \frac{v_i - v_j}{\rho_j}, \nabla^2 W_{spikyV}(r_{ij}, h) \tag{3.24}$$

kde $\boldsymbol{\mu}$ značí koeficient viskozity. Koeficient viskozity látky nicméně není konstantní veličinou. Obecně vzato viskozita se zvyšující se teplotou klesá a je tedy vhodné, aby koeficient viskozity byl dynamickou veličinou. V [22] je představena implicitní metoda pro simulaci vysoce viskózních kapalin, která je použita v práci[3], kde je koeficient viskozity vypočten na základě korelace mezi teplotou a viskozitou [23]:

$$\log(\log(\mu_i + \gamma)) = q - y \log(T_i), \tag{3.25}$$

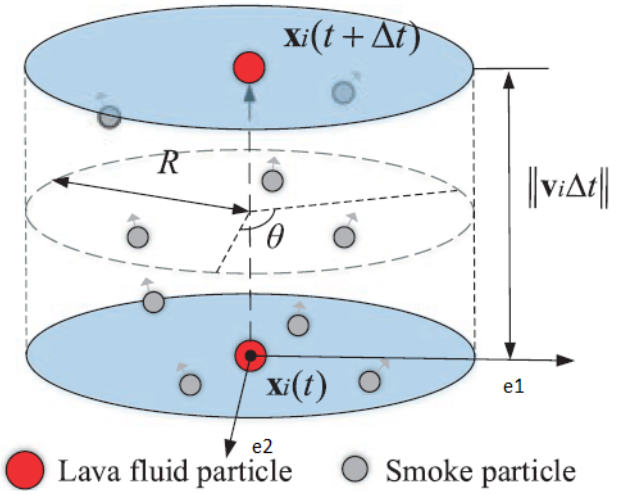
kde $\boldsymbol{\mu_i}$ je koeficient viskozity částice $\mathbf{i}$, $\boldsymbol{\gamma}$ je konstanta běžně v rozsahu od 0,6 do 0,9 a $\mathbf{q}$ s $\mathbf{y}$ jsou specifické parametry.

3.2.5 Skupenství

Při erupci lávy na povrch dochází k jejímu ochlazení při kontaktu s chladnějším vzduchem a povrchem. Při ochlazování postupně dochází ke změně lávy z kapalné látky na látku tuhou. Pro zachování této skutečnosti je nutno simulovat změnu skupenství a umožnit změnu kapalné látky na pevnou a naopak. Způsoby popsané v práci [24] a [3] přidělují každé částici hodnotu (dle typu částice) T_{melt} a a $T_{solidify}$ určující teplotu tání, respektive tuhnutí. V první ze zmíněných prací platí, že $T_{melt} = T_{solidify}$ a ke změně skupenství tak dochází okamžitě při dosažení krajní teploty. V práci [3] autoři umožňují $T_{melt} > T_{solidify}$ a představují přechodný stav látky. V tomto přechodném stavu je simulován fakt, že změna skupenství není vždy okamžitá a skutečná teplota v době přechodu do jiného skupenství částečně fluktuuje. Pro částice v přechodném stavu mění stav na základě lineární interpolace mezi hodnotou tepelné vodivosti pro jednotlivá skupenství. Při dosažení opačné hodnoty je změna skupenství dokončena.

Při skutečné sopečné erupci dochází také k uvolňování plynů, které jsou pod vysokým tlakem stlačené uvnitř magmatu. Po potřeby simulace sopečných plynů tedy nedochází ke změně skupenství částic, ale uvolňování plynů je simulováno pomocí generování nových částic reprezentující kouř v případě, že částice lávy dosáhne určité teploty T_{smoke}. Při vytvoření nových částice kouře se v práci [3] (jejichž metoda vychází z [25]) jejich pozice určuje na základě několika náhodných hodnot a je vybrána z válce, jehož podstavy prochází pozicí částice (jež generuje kouř) a předpovězenou budoucí pozicí částice (v rámci PCISPH) viz obr. 3.9

Nejprve se utvoří dva vektory $\mathbf{e_1}$ a $\mathbf{e_2}$ ortogonální k vektoru rychlosti $\mathbf{v_i}$. Pomocí těchto vektorů a pozice částice $\mathbf{x_i}$ vytvoříme rovinu definující podstavu válce. Druhá rovina je vytvořena pomocí pozice $\mathbf{x_i(t + \Delta t)}$. Pozice nově vzniklých částic je vypočtena na základě tří náhodných hodnot $X_r, X_\Theta, X_h \in [0...1]$). Vzdálenost od středu k okraji podstavy $\mathbf{r}$ se získá jako $\mathbf{r = R\sqrt{X_r}}$, kde obvykle platí, že $\mathbf{R = h}$. Azimut jako $\boldsymbol{\Theta} = \mathbf{X_\Theta 2}\pi$ a výška



Obrázek 3.9: Zobrazení válce pro generování částic kouře. [3].

$\mathbf{h} = \mathbf{X_h} \cdot \|\mathbf{\Delta v_i}\|$. Počáteční pozice nové částice $\mathbf{x_s}$ je vypočtena dle rovnice 3.26 a rychlost $\mathbf{v_s}$ dle rovnice 3.27, kde $\mathbf{e_1}$ a $\mathbf{e_2}$ zastávají vektory rychlosti.

$$x_s = x_i + r\cos\Theta e_1 + r\sin\Theta e_2 + hv_i \tag{3.26}$$

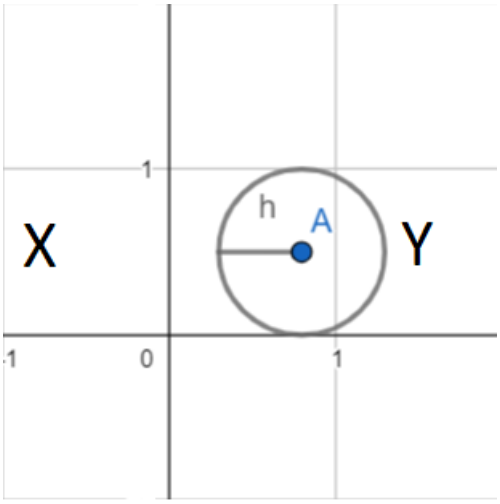
$$v_s = r\cos\Theta e_1 + rsin\Theta e_2 + v_i \tag{3.27}$$

3.3 Urychlení simulace

Jak bylo vysvětleno v sekci 2.0.2 a dalších, při výpočtu vlastností jednotlivých částic se jejich hodnota vypočte na základě okolních částic. Pokud bychom ovšem srovnávali každou částici se všemi ostatními, dostali bychom se na složitost $\mathcal{O}(n^2)$. Naivní řešení evidentně není použitelné pro větší množství částic (pro deset tisíc částic to dělá sto milionů porovnání).

Pro urychlení výpočtu se proto velice často využívá rozdělení prostoru do mřížky. Při použití mřížky o velikosti buňky navázané na vyhlazovací vzdálenost, lze pro každou částici k buňce do které daná částice patří projít pouze sousední buňky (dohromady tedy 27 buněk mřížky ve 3D prostoru). Počet procházených buněk lze ještě snížit, při správně zvolené velikosti mřížky. Pokud například zvolíme jako velikost mřížky dvojnásobek vyhlazovací vzdále-nosti, jsme schopni snížit počet prohledávaných buněk na 8. Pokud od pozice částice v jedné z os odečteme či přičteme vyhlazovací vzdálenost a zjistíme, že buňka s takto upravenou pozicí by spadala do jiné buňky, nemusíme již protilehlou buňku v dané ose procházet. Pokud bude částice vzhledem k jedné z os přesně uprostřed buňky, nemuseli bychom v této ose procházet dokonce žádné další

buňky, znázornění viz 3.10. Tímto způsobem se snižuje složitost na $\mathcal{O}(kn)$. Ne-



Obrázek 3.10: Znázornění možnosti vyřazení buněk z okolí částice. Po zjištění, že částice zasahuje do buňky Y, můžeme s jistotou rozhodnout, že částice v buňce X není nutno procházet.

výhodou tohoto postupu je, že hledání sousedů se musí několikrát opakovat. Při výpočtu hustoty, tlaku, teploty, výpočtu sil apod. Proto se můžeme ještě setkat se zrychlením pomocí listu sousedů. Na začátku každé iterace SPH se vypočtou sousedé pomocí mřížky a následně se na dané částici uloží do listu sousedů. Při dalších výpočtech se tedy sousedé nemusí znovu hledat, ale pouze se projde tento list.

Další zrychlení představuje paralelizace výpočtů. Jde zejména o paralelizaci výpočtů vlastností částic. Místo např. výpočtů hustoty pro jednu částici za druhou, můžeme bezpečně počítat hustotu pro několik částic paralelně. Vzhledem k tomu, že dochází pouze ke čtení hodnot z okolních částic, nehrozí že by došlo k přepisu hodnot více vlákny najednou (tzv. *race conditions*) – každé vlákno bude přepisovat pouze hodnoty pro svojí částici.

Mezi další možnosti pro urychlení vyhledávání sousedů patří rozdělení částic do hashovací tabulky nebo stavba kd-stromu a vyhledávání pomocí algoritmu pro **k** nejbližších sousedů [26]. Stavba hierarchických datových struktur nad částicemi nicméně není příliš častá, protože strukturu je nutno pře-stavět pro každý krok simulace.

Pomocí listu sousedů a paralelizace výpočtů se dá na CPU simulovat desetitisíce až statisíce částic v reálném čase, v posledních letech se nicméně paralelizace výpočtů přesunula z procesoru na grafické karty, které umožňují provádět výpočty až pro milióny částic současně. Výpočty na GPU pomocí rozhraní CUDA umožňují zrychlení simulace o jeden až dva řády [27].

3.4 Ohraničení simulace a kolize

Vzhledem k častému využití mřížek při simulaci, dochází k nutnosti uzavřít simulaci na konkrétní oblast prostoru. S tím přichází nutnost tuto hranici nějakým způsobem definovat a detekovat s hranicí kolize. K tomu se nejčastěji využívá těchto metod (či jejich kombinací) :

- Statických ohraničujících částic
- Uzavření simulace do ohraničujícího kvádru

U řešení pomocí statických částic přidáme do simulace částice, jejichž pozice je neměnná, ale většinu ostatních vlastností mají zachovanou. Díky tomu budou při příliš těsném kontaktu odpuzovat simulované částice a zajistí, že ty zůstanou uvnitř mřížky. U druhé metody uzavřeme simulaci do kvádru jehož rozměry odpovídají rozměrům mřížky. V případě, že některá z částic poruší tuto hranici, je tato částice posunuta zpět a její zrychlení je upraveno v opačném směru (viz naivní 3.1 a pokročilejší 3.2 řešení).

```
void Particles::forceBoundary(Particle*& p)
{
        if (p->pos.x < SIM_WIDTH_MIN_X)
        {
                p->acc.x *= BOUND_DAMPING;
                p->pos.x = SIM_WIDTH_MIN_X;
        }
        if (p->pos.x > SIM_WIDTH_MAX_X)
        {
                p->acc.x *= BOUND_DAMPING;
                p->pos.x = SIM_WIDTH_MAX_X;
        }
        if (p->pos.y  < SIM_HEIGHT_MIN_Y)
        {
                p->acc.y *= BOUND_DAMPING;
                p->pos.y = SIM_HEIGHT_MIN_Y;
        }
        if (p->pos.y > SIM_HEIGHT_MAX_Y)
        {
                p->acc.y *= BOUND_DAMPING;
                p->pos.y = SIM_HEIGHT_MAX_Y;
        }
        if (p->pos.z  < SIM_DEPTH_MIN_Z)
        {
                p->acc.z *= BOUND_DAMPING;
                p->pos.z = SIM_HEIGHT_MIN_Z;
        }
```

```
        if (p->pos.z > SIM_DEPTH_MAX_Z)
        {
                p->acc.z *= BOUND_DAMPING;
                p->pos.z = SIM_DEPTH_MAX_Z;
        }
}
```

Listing 3.1: Funkce pro udržení částice uvnitř hraničního kvádru. Acc značí zrychlení, pos pozici a bound_damping (-1,0) určuje odrazivost stěny kvádru.

```
diff = bound_dist - (p->pos.x - SIM_WIDTH_MIN_X);
if (diff > EPSILON) {
        norm = vec3(1.0, 0, 0);
        dot = dot(norm, p->vel)
        adj = ext_stiff * diff - bound_damping * dot;
        p->acc.x += adj * norm.x;
}

diff = bound_dist - (SIM_WIDTH_MAX_X - p->pos.x);
if (diff > EPSILON) {
        norm = vec3(-1.f, 0, 0.f);
        dot = dot(norm, p->vel)
        adj = ext_stiff * diff - bound_damping * dot;
        p=>acc.x += adj * norm.x;
}
```

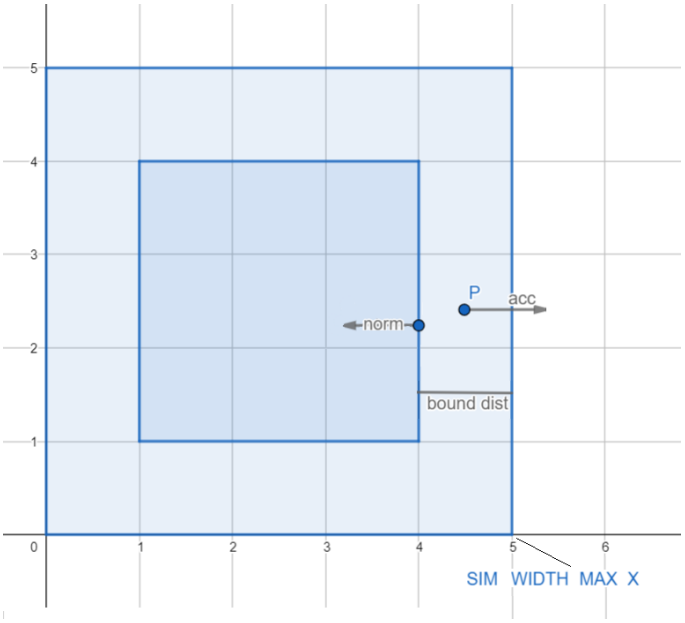
Listing 3.2: Pokročilejší kód pro udržení částice uvnitř hraničního kvádru (zobrazuje výpočet v ose x)

V ukázce 3.2 můžeme vidět řešení, které umožňuje nastavit pružnost (**ext_stiff**) hraničního kvádru. Díky nastavení pružnosti mohou částice při dostatečné rychlosti dočasně proniknout hranicí kvádru. Čím dále proniknou, tím větší je působící síla, která je vrací zpět do ohraničeného prostoru (ext_stiff * diff, která je následně vynásobena normálou). Vzhledem k tomu, že pokud bychom skutečně umožnili průnik částic za hranici kvádru, nenacházely by se už v prostoru mřížky, je nutno přidat hodnotu bound_dist, díky které se hraniční výpočet začne provádět ještě předtím, než částice skutečně hraniční kvádr protnou. Situaci znázorňuje obr. 3.11

3.5 Vizualizace SPH

Přestože vizualizace výsledků simulace není hlavním cílem této práce, data je i tak nutno nějakým způsobem zobrazit, jinak by výsledky bylo velice těžké pro čtenáře i pro programátora, vytvářejícího simulaci, posoudit a zhodnotit. Je tedy vhodné uvést alespoň základní přehled vizualizačních technik.

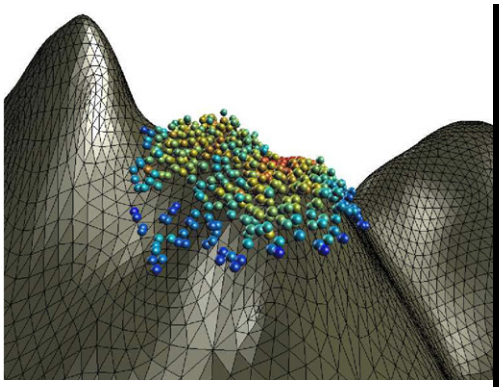
Nejjednodušším způsobem vizualizace simulace SPH je pomocí bodů či koulí - jednou pro každou simulovanou částici. Takovéto základní zobrazení je dostatečné pro posouzení, zda simulace z hlediska pohybu, viskozity, přesunu sil apod. funguje, a pomocí různých barevných módů umožňuje také kontrolu sledovaných hodnot jako je teplota, hustota či tlak. Tuto vizualizační techniku (obr. 3.12) můžeme nalézt například v práci [28] a může být zcela dostatečná například při simulaci písku, který se za určitých podmínek chová jako kapalina.



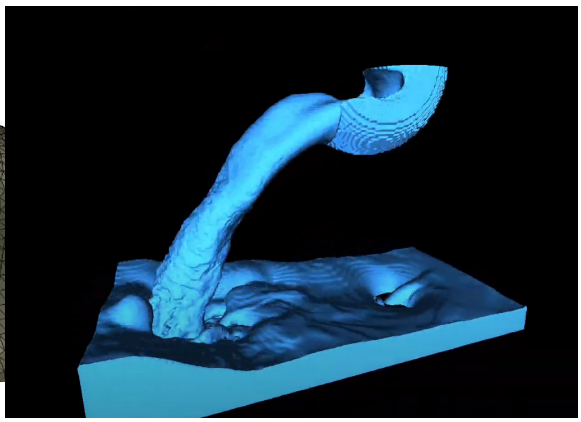
Obrázek 3.11: Ilustrace hraniční situace. Rozdíl mezi plochami obou čtverců definuje prostor, ve kterém dochází k úpravě zrychlení dle kódu 3.2.

Dalším vhodným a často využívaným vizualizačním prostředkem je algoritmus **marching cubes**. Tato vizualizační technika již zobrazuje povrch simulované látky a dosahuje tak lepších výsledků při zobrazení kapalin. Algoritmus marching cubes pro triangulaci povrchu využívá mřížky, pro simulaci pomocí SPH je tak nutno tuto mřížku nad daty vytvořit, zatímco u Eulerovských metod simulace je možno využít simulační mřížku. Algoritmus postupně prochází buňky mřížky a dle hodnot v jejích vrcholech rozděluje prostor na vně simulované látky a uvnitř simulované látky. Podle počtu vrcholů, které jsou uvnitř či vně, je následně dané buňce přiřazena jedna z 15 možných konfigurací trojúhelníků. Tuto metodu můžeme najít například v práci [3] a vidět na obr. 3.13.

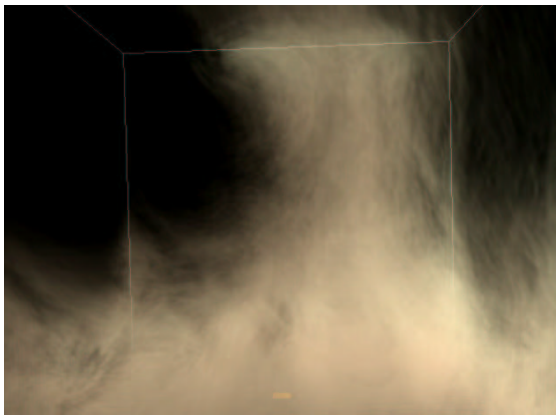
Vzhledem k tomu, že marching cubes algoritmus vytváří pouze triangulovaný povrch simulované látky, nehodí se pro vizualizaci plynů. K jejich vizualizaci je třeba použít pokročilejších volumetrických metod [29], které obvykle vytvářejí výsledný obraz pomocí voxelové mřížky skrz kterou je sledován paprsek od kamery.



Obrázek 3.12: Vizualizace SPH pomocí koulí na pozicích částic. [30]



Obrázek 3.13: Vizualizace SPH pomocí algoritmu marching cubes. [31]



Obrázek 3.14: Vizualizace SPH pomocí volumetrického renderingu. [32]

Kapitola 4

Analýza a návrh

V předchozích částí práce došlo k seznámení s SPH a způsobu jeho fungování, popisu problémů souvisejících se simulací lávy a jejich možným řešením. Z těchto poznatků jsem došel k požadavkům, které by měla výsledná aplikace splňovat:

- Aplikace by měla umožnit simulaci toku lávy po terénu v uzavřeném prostoru.
- Aplikace by měla umožnit pohyb kamery po scéně.
- Aplikace by měla obsahovat několik různých scén vycházejících z výškové mapy.
- Aplikace by měla umožnit přidávat nové částice do scény.

a požadavky které by měla splňovat samotná simulace:

- Simulace by měla být ovlivňována silami vycházejícími z tlaku, gravitace, viskozity a tření.
- Simulace by měla obsahovat hraniční kvádr, který uzavírá simulaci.
- Simulace by měla umět načíst do scény statické částice (reprezentující horu) i částice reprezentující tekutinu.
- Simulace by měla být schopna zpracovat předávání tepoty mezi částicemi.
- Simulace by měla dynamicky měnit viskozitu a tepelnou vodivost v závislosti na teplotě částice.
- Simulace by měla umožnit generování částic kouře při dosažení určité teploty.

- Simulace by měla umožnit tuhnutí a tání částic reprezentujících kapalinu v závislosti na teplotě.

- Simulace by měla běžet v reálném čase na moderním CPU s alespoň šesti jádry pro 20 tisíc částic alespoň na 30 FPS.

- Simulace by měla využívat paralelizace.

4.1 Zvolené řešení

Po prvotním testování a seznámení se s několika zdrojovými kódy různých druhů SPH simulací jsem zvolil SPH na základě práce Müllera [13] pro jeho relativně přímočarou implementaci a dostatek informací ohledně vyhlazovacích funkcí a konstant. PCISPH by bylo zřejmě k simulaci vhodnější a jeho základní verzi jsem při vývoji aplikace také napsal, ale vzhledem k mnoha neznámým hodnotám se mi nepodařilo simulaci zprovoznit. Při výběru řešení pro vlastnosti lávy, které nejsou v práci Müllera obsaženy budu vycházet z práce [3], či prací na které se tato práce odkazuje a upravovat výpočty, aby byly vhodné pro použití v SPH na základě Müllerovi práce. Celá aplikace bude napsána v C++.

K urychlení simulace jsem se rozhodl využít klasické mřížky, ale bez použití listu sousedů. List sousedů se nicméně zdál ideální pro výpočet přenosu teploty mezi statickými částicemi - list stačí v tomto případě vygenerovat při počátku dané scény a nijak se dále nemusí měnit.

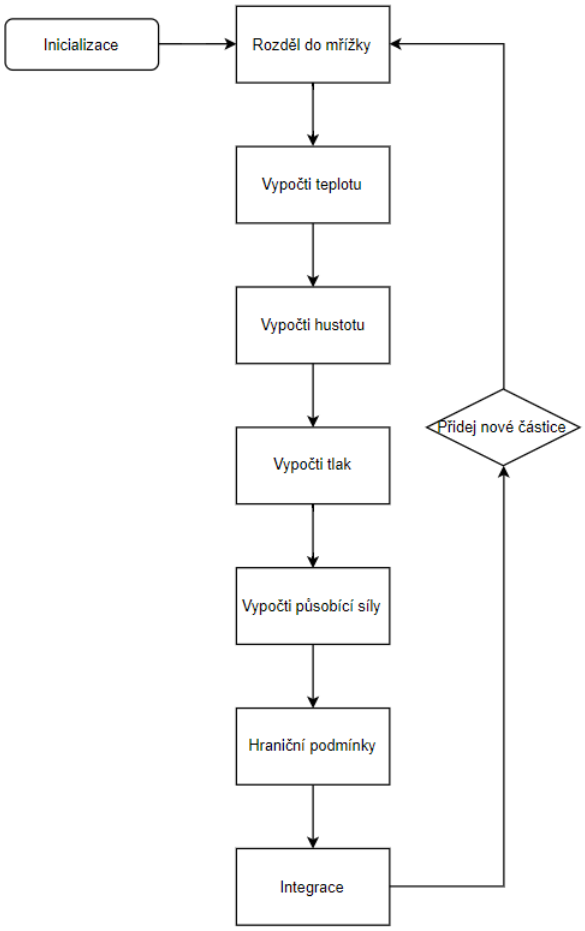
K zobrazení bude použito OpenGL a částice budou zobrazené pomocí textury kruhu s průhledným pozadím natočené ke kameře za pomocí jednoduchých shader programů v jazyce GLSL.

Z použití běžné statické mřížky potom plyne nutnost výběru způsobu uzavření simulace. Rozhodl jsem se využít metody uzavření simulace do hraničního kvádru - konkrétně pokročilejší verzi popsanou v sekci 3.4. Vzhledem k nutnosti využití statických částic jako terénu tedy dochází ke kombinaci obou zmiňovaných metod, statické částice ale nejsou v tomto případě použity k ohrazení simulace.

Co se týče paralelizace, samotné částice kouře nebudou interagovat s ostatními částicemi a na základě toho jsem se rozhodl oddělit část simulace kouře a zbytek simulace do paralelně běžících vláken. K další paralelizaci na úrovni výpočtu vlastností jednotlivých částic využiji moderních možností jazyka C++ – konkrétně paralelizace průběhu cyklů. Rozhraní CUDA a výpočty na grafické kartě jsem zvažoval, ale vzhledem k tomu, že s výpočty na GPU nemám žádné zkušenosti, rozhodl jsem se zůstat u výpočtů na CPU.

4.2 Struktura aplikace

Diagram na obrázku 4.1 představuje hlavní jádro aplikace z hlediska kroků SPH simulace. Tento diagram představuje základní jádro simulace, před tím než bylo rozděleno do několika vláken – podrobnější diagram popisující toto rozdělení bude uveden později. Samotné rozdělení není identické se skutečnou implementací a je spíše ilustrační, upravené pro větší přehlednost.



Obrázek 4.1: Diagram základní struktury simulační smyčky.

4.2.1 Průběh simulace

Zde budou blíže popsány jednotlivé kroky simulace, jež budou ještě podrobněji rozebrány v části implementace.

Inicializace. Počáteční krok SPH simulace. Prostor simulace je rozdělen

na 3 mřížky, jednu pro každý typ částic. Jsou vygenerovány statické částice z výškové mapy (černobílý obrázek určující výšku terénu na základě odstínu šedé v daném bodě). V rámci tohoto kroku jsou načteny počáteční statické i dynamické částice, které jsou rozmístěny do jednotlivých buněk. Jsou před počítaná pole sousedů statických částic a v neposlední řadě je vygenerován zásobník částic kouře (program jich dovoluje jenom omezený počet a dynamická alokace za běhu by mohla program zpomalovat).

Rozdělení do mřížky. Dynamické částice lávy jsou rozděleny do buněk na základě pozice. Samotná mřížka je datovou strukturou která jednotlivé buňky ukládá do jednorozměrného pole (konkrétně jde o vektor) a indexy k nim jsou vypočteny z 3D souřadnic. Každé pozici v ohraničeném 3D prostory tedy přiřazuji právě jednu buňku v tomto poli. Jednorozměrné pole je ideální pro rychlý přístup do paměti. Jednotlivé buňky jsou potom také definovány jako vektory, které mají předem rezervovanou omezenou kapacitu, aby se předešlo zbytečnému zvětšování vektoru. List by v tomto případě také připadal v úvahu, ale, chtěl jsem aby data v buňce zůstaly v paměti u sebe. Při dotazu na sousedy částice se přistoupí pouze do nezbytných 8 buněk v okolí.

Výpočet teploty. V této částí probíhá převod tepla mezi dynamickými i statickými částicemi, také dochází ke změně vlastností látky - konkrétně ke změně viskozity, tepelné vodivosti, koeficientu ovliňujícím tření a změně skupenství. Ihned po výpočtu teploty následuje vygenerování částic kouře. Při generování částic kouře vycházím z práce [25], převod tepla je kombinací metod popsaných v [3] a [7].

Výpočet hustoty. Hustota je vypočtena dle vzorců z Müllerovy práce. Počítáno je i se statickými částicemi.

Výpočet tlaku. Původně jsem se rozhodl k výpočtu tlaku používat zákon ideálního plynu, ale následně jsem přešel k Taiotovým rovnicím, protože při jejich použití vypadala simulace přesvědčivěji. Výpočet tlaku a hustoty probíhá ve skutečnosti v jednom kroku, pro úsporu výpočetního času.

Výpočet působících sil. V této části dochází k výpočtu sil působících na částici na základě hustoty, tlaku, viskozity a tření. Samotné tření budu pouze simulovat pomocí výpočtu podobného s výpočtem viskozity, v případě že částice lávy je v kontaktu s pevnou částicí (ať již ztuhlé lávy, nebo statickou částicí). Gravitační síla je připočtena až v rámci integračního kroku.

Hraniční podmínky. V této části je upravena akcelerace částice vzhledem k její možné pozici poblíž hraničního kvádru. Pokročilejší metoda řešení hraničních podmínek popsaná v části 3.4 upravuje pouze momentální změnu rychlosti pro daný krok simulace a umožňuje částici, aby se dostala za okraj

vnitřního hraničního kvádru znázorněného na obrázku 3.11. Díky těmto vlastnostem umožňuje simulovat pružnost hranice, ale zároveň nezabraňuje, aby se částice pohybující se příliš vysokou rychlostí nedostaly za skutečný hraniční kvádr. Aby se omezila tato možnost, je v tomto kroku také testována a omezena maximální rychlost částice.

Integrace. Poslední krok simulační smyčky, ve kterém je vypočtena nová pozice částic pomocí Eulerovy metody popsané v části 3.1.1. Simulace se posouvá o časový krok. Cílem je zvolit co největší časový krok při zachování stability simulace.

Generování nových částic. V případě, že je zpracován podnět od uživatele, jsou do simulace přidány a zařazeny nové částice lávy.

4.2.2 Průběh programu

Následující diagram na obrázku 4.2 znázorňuje výpočetní smyčku aplikace.

Inicializace OpenGL. Počáteční nastavení okna aplikace, příprava zásobníků (bufferů) pro vykreslování částic, načtení vertex a fragment shaderu.

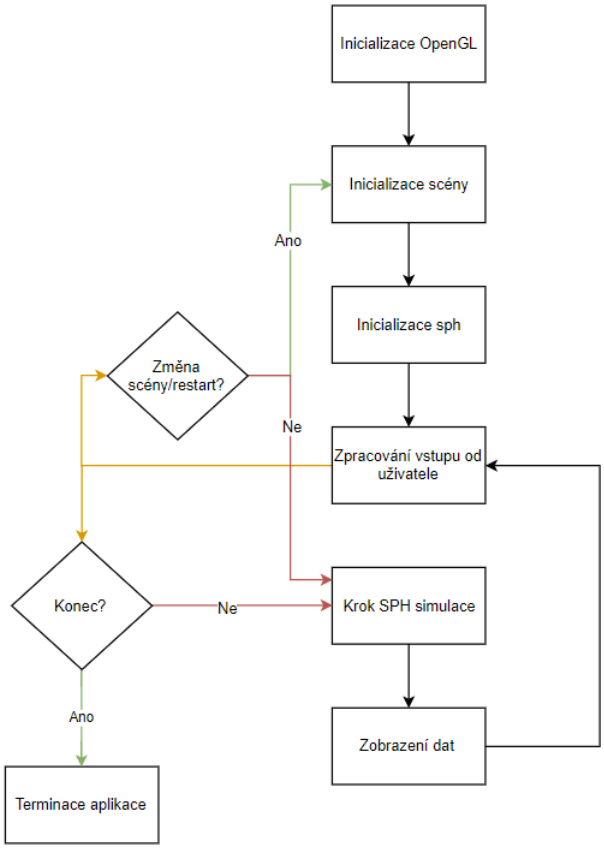
Inicializace scény. V této části probíhá načtení výškové mapy, pokud jí daná scéna používá, následuje vygenerování statických částic dle výškové mapy a dynamických částic lávy dle počátečního nastavení ve scéně.

Inicializace SPH. Tato část je popsána v předchozí části textu. SPH solveru jsou předány vygenerované částice, proběhne inicializace a následně je nastaveno několik dynamických parametrů, které se liší scénu od scény. Jde například o možnost zapnout či vypnout generování kouře, interakci se statickými částicemi, velikost časového kroku simulace apod.

Inicializace SPH. Jsou zpracovány vstupy od uživatele. Jde o možnou změnu kamery a pozice, generování nových částic, restartování scény, změnu scény, změnu způsobu vykreslování (možnost přepnout na mód zobrazující hustotu) a možnost ukončení aplikace.

Krok SPH simulace. Je popsán v předchozí části textu.

Zobrazení dat. Prochází se jednotlivé typy částic a jejich poloha a barva se ukládají do příslušných bufferů. Následuje vykreslení částic.



Obrázek 4.2: Diagram průběhu aplikace.

Kapitola 5

Implementace

V této části bude popsána implementace programu. Budou detailně popsány důležité datové struktury, třída provádějící simulaci a jednotlivé kroky simulace. Na konci bude popsáno, jakým způsobem byl program převeden pro běh na více vláknech současně. Základ SPH simulace je částečně postaven na práci [33] a [34].

5.1 Třídy a struktury

V této části budou popsány důležité třídy a struktury programu.

5.1.0.1 Grid3d

Struktura reprezentující mřížku postavenou nad prostorem simulace. V předchozí části jsem pro přehlednost popsal ,že mřížka je implementována jako jednorozměrný vektor, kde se pod jednotlivými indexy ukrývá obsah buňky. Samotná buňka je ovšem také definována jako vektor, ve skutečnosti tedy jde o dvourozměrný vektor. Část implementace mřížky je v ukázce 5.1. Proměnné **w, h, d** ukládají šířku, výšku a hloubku mřížky, **xDiv, yDiv, zDiv** potom ukládají velikost jednotlivých buněk v dané ose, jež je rovna dvojnásobku vyhlazovací vzdálenosti (všechny tři hodnoty jsou tedy identické). **Operátor()** umožňuje přístup ke konkrétní buňce – její pozice ve vektoru je vypočtena na základě vstupních parametrů reprezentujících pozici částice. Ukázka kódu 5.2 představuje implementaci vyhledávání sousedních mřížek. Vstupním parametrem je pozice částice k níž hledáme sousední buňky (a buňku, do které spadá samotná částice). Nejprve je připraven vektor, který bude obsahovat všechny buňky. Následně je otestováno, jestli by částice posunutá v daných osách o vyhlazovací vzdálenost (xDiv/2 – v případě osy X) spadala do jiné buňky než částice původní. Pokud ano, můžeme vynechat buňky v dané ose v opačném směru. Na konci je naplněn výstupní vektor všemi neprázdnými buňkami, které splnily vyhledávací kritéria.

```
struct grid3d {
        vector<vector<Particle*>> data;
        unsigned w, h, d, xDiv, yDiv, zDiv;

        vector<Particle*>& operator()
        (const float& x, const float& y, const float& z)
        {
            return data[getZ(z)*w*h + getY(y) * w + getX(x)];
        }
        int getX(const float& pos) const
        {
            return static_cast<int>(pos / (xDiv));
        }
        int getY(const float& pos) const
        {
            return static_cast<int>(pos / (yDiv));
        }
        int getZ(const float& pos) const
        {
            return static_cast<int>(pos / (zDiv));
        }
};
```

Listing 5.1: Část implementace struktury reprezentující mřížku.

```
vector<vector<Particle*>*> getNeighbourCells(vec3& pos)
{
    vector<vector<Particle*>*> v;
    v.reserve(8);
    int gridX = getX(pos.x);
    int gridY = getY(pos.y);
    int gridZ = getZ(pos.z);
    int incX = 0;
    int decX = 0;
    int incY = 0;
    int decY = 0;
    int incZ = 0;
    int decZ = 0;
    if (getX(pos.x - (xDiv / 2)) < gridX) {
        incX = 0;
        decX = 1;
    }
    else if (getX(pos.x + (xDiv / 2)) > gridX) {
```

```
            incX = 1;
            decX = 0;
        }
        if (getY(pos.y - (yDiv / 2)) < gridY) {
            incY = 0;
            decY = 1;
        }
        else if (getY(pos.y + (yDiv / 2)) > gridY) {
            incY = 1;
            decY = 0;
        }
        if (getZ(pos.z - (zDiv / 2)) < gridZ) {
            incZ = 0;
            decZ = 1;
        }
        else if (getZ(pos.z + (xDiv / 2)) > gridZ) {
            incZ = 1;
            decZ = 0;
        }
        for (int i = gridX - decX; i <= gridX + incX; i++)
            for (int j = gridY - decY; j <= gridY + incY; j++)
                for (int k = gridZ - decZ; k <= gridZ + incZ; k++)
                {
                    if (i >= 0 && j >= 0 && k >= 0
                        && i < w && j < h && k < d)
                        if (getCellSize(i, j, k) > 0)
                            v.push_back((&getCell(i, j, k)));
                }
        return v;
}
```

Listing 5.2: Implementace nalezení relevantních sousedních buněk mřížky.

5.1.1 Particle

Tato struktura reprezentuje jednotlivé částice ve scéně. Jde o nejdůležitější strukturu v aplikaci. Většina změn v SPH probíhá na základě sumy hodnot vlastností jednotlivých částic. Přehled většiny proměnných s případnou vysvětlivkou k dispozici v tabulce 5.1.

Dynamická Viskozita je vypočtena jako:

$$viscosity = 3.5f(1.f - (temperature/(tempMax + 0.25tempMax))); \quad (5.1)$$

Název parametru	Typ	Popis
position	vec3	
velocity	vec3	
force	vec3	
density	float	
presure	float	
temperature	float	
temp_diff	float	
friction	float	lineárně závisí na teplotě
conductivity	float	logaritmicky závisí na teplotě
viscosity	float	lineárně závisí na teplotě
neighbours	int	Počet sousedů, využito ke ztrátě tepla do "vzduchu"
inContactWithSolid	bool	využito při změně skupenství - částice může být statická až v případě, že se dotýká jiné statické částice, jinak by se mohla zaseknout ve vzduchu
neighboursArray	vector <Particle*>	seznam statických sousedů statických částic pro urychlení předávání tepla u částic terénu
life	float	životnost částice kouře
state	State	skupenství

Tabulka 5.1: Proměnné struktury Particle.

5.1.2 SPHSolver

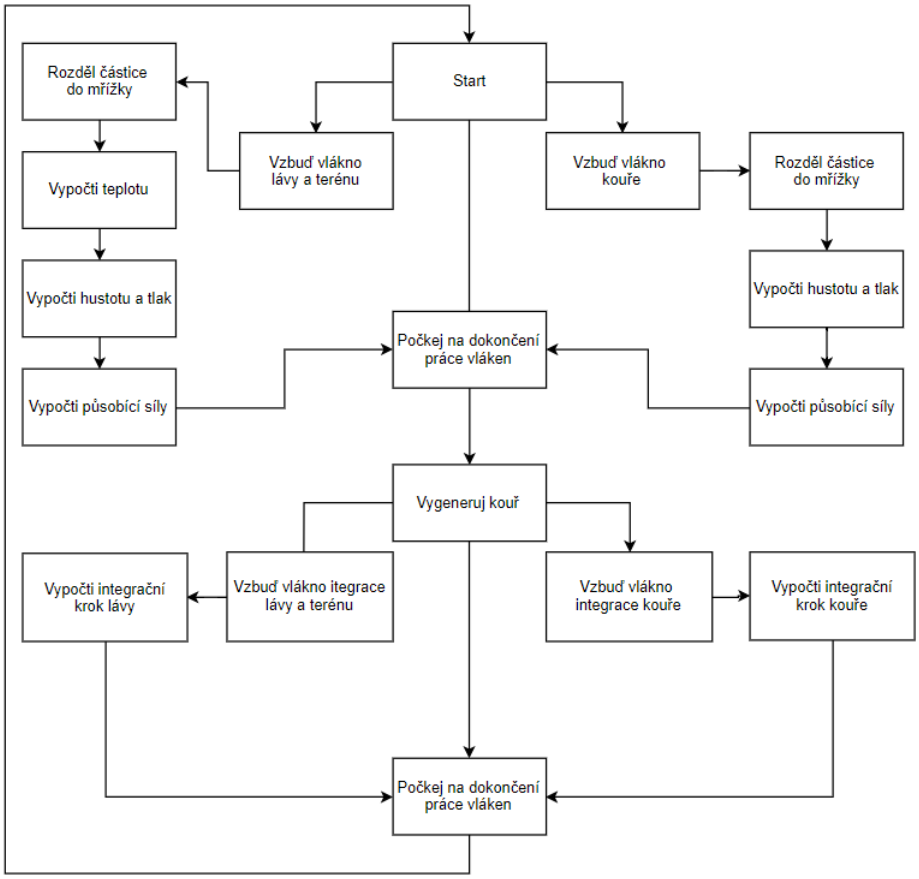
Tato třída se stará o průběh celé simulace. Obsahuje nutné konstanty pro výpočet, jako jsou jádrové funkce, vyhlazovací délka, hmotnost částic (v této implementaci se hmotnost nemění), integrační krok atd., tyto hodnoty jsou v tabulce 5.2. Znalost těchto hodnot je zcela zásadní při snaze o vlastní implementaci této práce. V tabulce ještě chybí hodnoty určující velikost simulovaného prostoru. Pro tzv. *dam break* scény jde o 60, 60, 30 pro šířku, výšku a hloubku a o 128, 60, 128 pro další scény. Třída SPHSover také udržuje dříve zmíněné mřížky a seznamy všech částic a obsahuje funkce pro výpočet hustoty, tlaku, působících sil, přenosu tepla, integrace, generování kouře a další.

Třída SPHSolver obsahuje čtyři vlákna, která zpracovávají: simulační krok pro lávu s výjimkou integrace, simulační krok pro kouř s výjimkou integrace, integraci částic lávy a integraci částic kouře. Celá simulační smyčka je na diagramu 5.1.

ComputeDensityPressure – výpočet hustoty a tlaku probíhá dle práce Müllera, ale za použití Taitovi rovnice. Za zmínku stojí, že v hodnotě hustoty si uchovávám její převrácenou hodnotu, protože hustotou se v SPH výpočtech běžně dělí, což je výrazně pomalejší operace než násobení.

Název proměnné	**Hodnota**	**Typ**	**Popis**
GRAVF	(0, -9.8, 0)	vec3	Gravitační síla
initStiff	0.5	float	Tuhost, hodnota pro zákon ideálního plynu
initStiff2	200	float	Tuhost, hodnota pro Taitovi rovnice
extStiff	20000	float	Tuhost hraničního kvádru
H	0.01	float	Vyhlazovací vzdálenost
HSQ	H^2	float	
REST_DENS	1000	float	Hustota v klidu
REST_DENS _SMOKE	2000	float	Hustota kouře v klidu
MASS	0.00020543	float	Hmotnost
MASS_SMOKE	MASS /1.80	float	Hmotnost kouře
borderRadius	0.004	float	Délka hranice před hraniční krychlí
speedLimit	100	float	Limit rychlosti
speedLimitSQ	speedLimit^2	float	
POLY6	365 / (64.f*PI*H^9	float	Jádrová funkce hustoty
SPIKY_GRAD	-45 / (PI*H^6)	float	Jádrová funkce tlaku
VISC_LAP	45 / (PI*H^6)	float	Jádrová funkce viskozity
solidTempTreshold	300	float	Teplotní hranice pro pevnou látku
gasTempTreshold	1055	float	Teplotní hranice pro generování plynu
tempMin	1	float	Minimální teplota
tempMax	1500	float	Maximální teplota
simScale	0.004	float	Hodnota pro přepočet pozice
DT = 0.001f	0.003	float	Integrační krok

Tabulka 5.2: Proměnné a konstanty třídy SPHSolver



Obrázek 5.1: Diagram znázorňující simulační smyčku.

ComputeForces – výpočet působících sil pro pár částic $\mathbf{p_i}$ a $\mathbf{p_j}$ které jsou ve vzdálenosti $\mathbf{r} < \mathbf{H}$ probíhá dle výpočtu z práce [34], kde *pterm* značí složku tlaku, *dterm* složku hustoty a *vterm* složku viskozity, viz ukázka kódu 5.3. Vzhledem k tomu, že tato funkce je poslední volanou funkcí před integrací, je v ní také zjišťováno, jestli jsou částice lávy, jejichž teplota je pod bodem tuhnutí, v kontaktu s jinou pevnou a nepohyblivou částicí. V takovém případě se nastaví proměnná $inContactWithImmobileSolid$ na kladnou hodnotu a částice se v integračním kroku přeskočí. Tento přístup zajišťuje, že částice se dále účastní výpočtů, stane se statickou a zároveň nemůže přejít ve statickou částici uprostřed letu ve vzduchu.

```
float pterm = -0.5f * (H - r) * SPIKY_GRAD *
            *(pi->pressure + pj->pressure) / r;

float dterm = (H - r) * pi->dens * pj->density;

float vterm = VISC_LAP * pi->viscosity;

pi->force += ((pterm * rij + vterm*(pj->vel - pi->vel)) *
            *dterm);
```

Listing 5.3: Část implementace funkce pro výpočet působících sil.

Integrate – v integračním kroku dojde k vynásobení působící síly hmotností částice, následně je aplikováno omezení rychlosti částice a připočtena gravitační síla. Výsledná síla je potom poslána do funkce $forceBoundary$, která upraví sílu dle blízkosti částice k okraji simulace. Výsledná pozice je připočtena takto: $p- > pos+ = (p- > vel * p- > vel * DT)/simScale$.

ComputeTemperature – funkce, která obstarává převod tepla, změnu skupenství, dynamicky mění vlastnosti částic dle teploty. Vzhledem k absenci částic vzduchu, které by zaplňovaly všechen volný prostor, s kterými by mohla také probíhat výměna tepla, je zda ztráta tepla do okolní atmosféry naimplementována na základě počtu souseních částic. Čim méně sousedů, tím větší ztráta tepla. Výpočet předávání teploty mezi páre částic p_i a p_j které jsou ve vzdálenosti $r < H$ probíhá kombinací metod v [3] a [7] viz 5.4.

```
float r = glm::length(rij);

float grad_Wij = (H - r)* VISC_LAP;
float temp_dif = ((pj->thermalConduct*pj->temp) -
               - (pi->thermalConduct*pi->temp));

float laplac_add = (MASS / (1 / pj->dens)) *
                 * temp_dif * grad_Wij;

pi->tempDiff += laplac_add;
...
float dif = 2 * pi->temp_dif / 100000;
pi->temp += dif*DT;

/// lose some heat to environment
if (pi->temp > 20) {
    float tempLostSpeed = 2;
  if (pi->neighbours < 22)
    tempLostSpeed = 4;
```

```
  if (pi->neighbours < 15)
    tempLostSpeed = 7;
  if (pi->neighbours < 10)
    tempLostSpeed = 10;
  if (pi->neighbours < 5)
    tempLostSpeed = 50;
  pi->temp -= (pi->temp * (tempLostSpeed / 10)) * DT;
}
pi->temp = std::clamp(pi->temp, tempMin, tempMax);
if (pi->temp < solidTempTreshold) {
    pi->state = Solid;
}
if (pi->state==Solid && pi->temp > solidTempTreshold){
    pi->state = Fluid;
}

pi->computeViscosity();
pi->computeFriction();
pi->computeThermalConductivity();
```

Listing 5.4: Část implementace funkce pro výpočet předávání tepla

5.2 Paralelizace

I když oddělení simulace lávy a kouře do samostatných vláken výpočet zrychluje, zcela nejzásadnější je paralelizace při procházení jednotlivých částic. Místo jednoduchého for cyklu, který by procházel jednu částici za druhou, k ní hledal sousedy atd., probíhají tyto výpočty paralelně pro několik částic najednou za využití možností C++17. To přináší možnost využít pro *for_each* cyklus tzv. *execution policy* – volně přeloženo jako spouštěcí pravidla. Stačí zavolat cyklus s parametrem *std* :: *execution* :: *par_unseq* a výpočet bude probíhat paralelně. Je nutné zajistit, že nedochází k zápisu do stejných hodnot najednou. Vzhledem k tomu, že se ale v rámci cyklů zapisují vždy jen hodnoty ke konkrétní částici, tento problém nenastává.

Klávesa	**Popis**
W,A,S,D	Pohyb
Myš	Rozhlížení se
Mezerník	Znovu přidání počátečních částic
R	Restart scény
V	Generování nových částic
C	Reset kamery
1-6	Scény
X	Výpis pozice
I	Běžné barevné zobrazení
O	Obarvení částic dle hustoty

Tabulka 5.3: Tabulka s ovládáním aplikace

5.3 Ovládání

Aplikace se ovládá pomocí myši a klávesnice, konkrétní přehled je v tabulce 5.3.

Kapitola 6

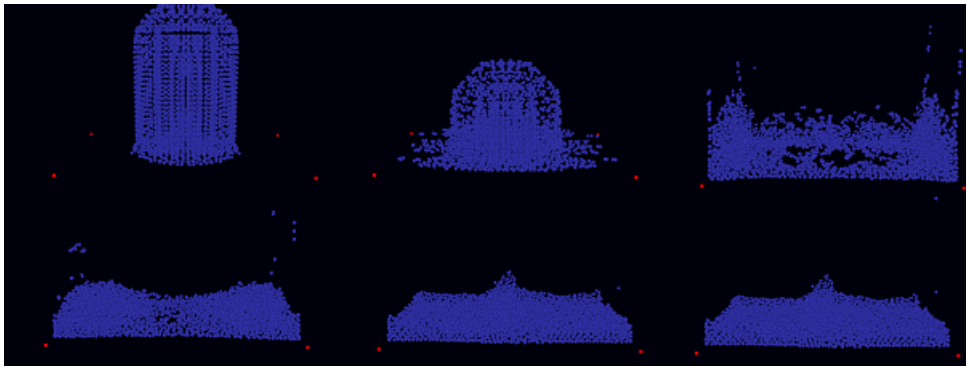
Výstup práce

6.1 Výsledky

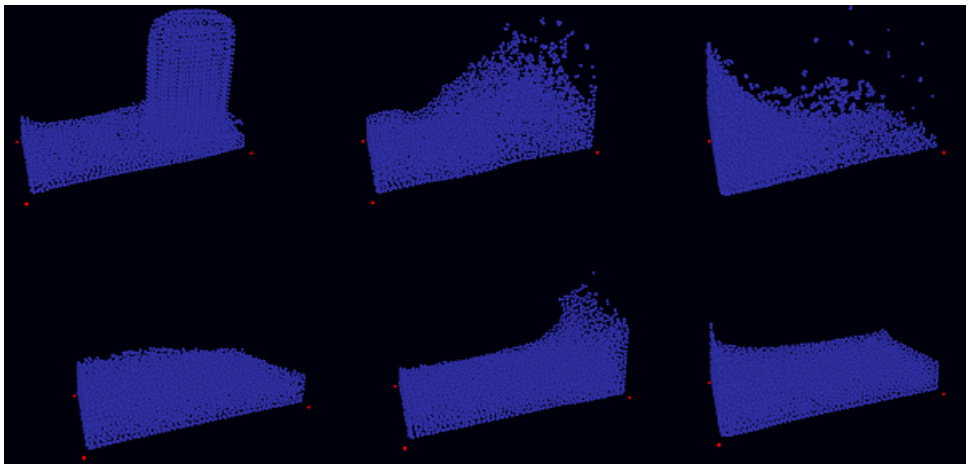
Aplikace obsahuje 3 scény se simulací vody - jde o standardní test SPH solveru a 3 scény simulující lávu. První se sopkou a okolím, druhý na testování dynamické viskozity a třetí na testování změny z pevného na kapalné skupenství. Následuje několik obrázků zobrazující výsledky této práce a v příloze práce jsou k dispozici videa. Na obrázcích 6.1, 6.2 a 6.3 je klasická testovací scéna SPH simulací *Dam break*, neboli protržení hráze, červené tečky značí hranici simulace.

V průběhu práce jsem se podrobně seznámil s několika odlišnými přístupy k řešení SPH simulace. Kontrolní scény simulující vodu i scény simulující lávu jsou přesvědčivé a vizuálně zajímavé. Jejich důvěryhodnost snižují pouze nedostatky způsobené stlačitelností kapaliny, které vyplývají z použité metody. I když se mi nepodařilo implementovat simulaci pomocí PCISPH, v jeho zprovoznění mi bránilo pouze nedostatek informací o klíčových konstantách, ne nedostatek zkušeností v programování. Autory práce [3] se mi bohužel nepodařilo kontaktovat. Myslím si, že implementace tření s pevnými látkami na základě vzorců pro viskozitu podává dostatečně přesvědčivé výsledky a dynamická viskozita funguje částic funguje zcela správně.

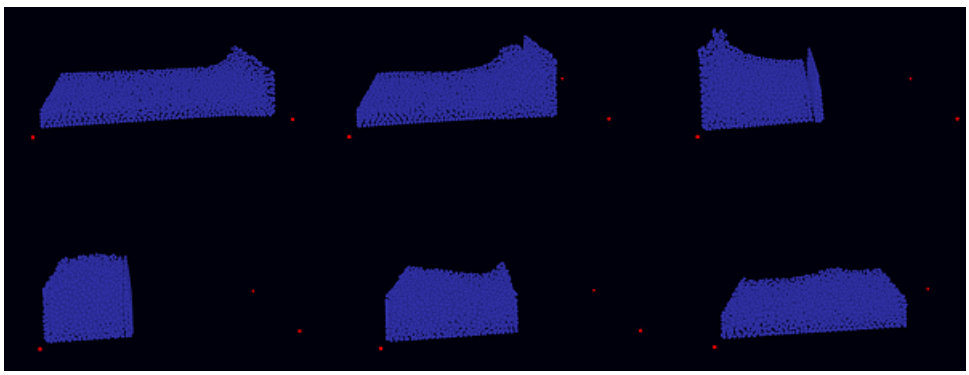
Pro případné rozšíření aplikace bych nejspíš umožnil uživateli více způsoby ovlivňovat simulovanou scénu. Ať už jde o dynamické přenastavení vlastností látky, větší kontrolu nad generovanými částicemi, či možnost načíst z příkazové řádky vlastní výškovou mapu. Za jasný úspěch také považuji paralelizaci simulace, díky čemuž se mi podařilo simulaci proti první implementaci SPH mnohonásobně zrychlit.



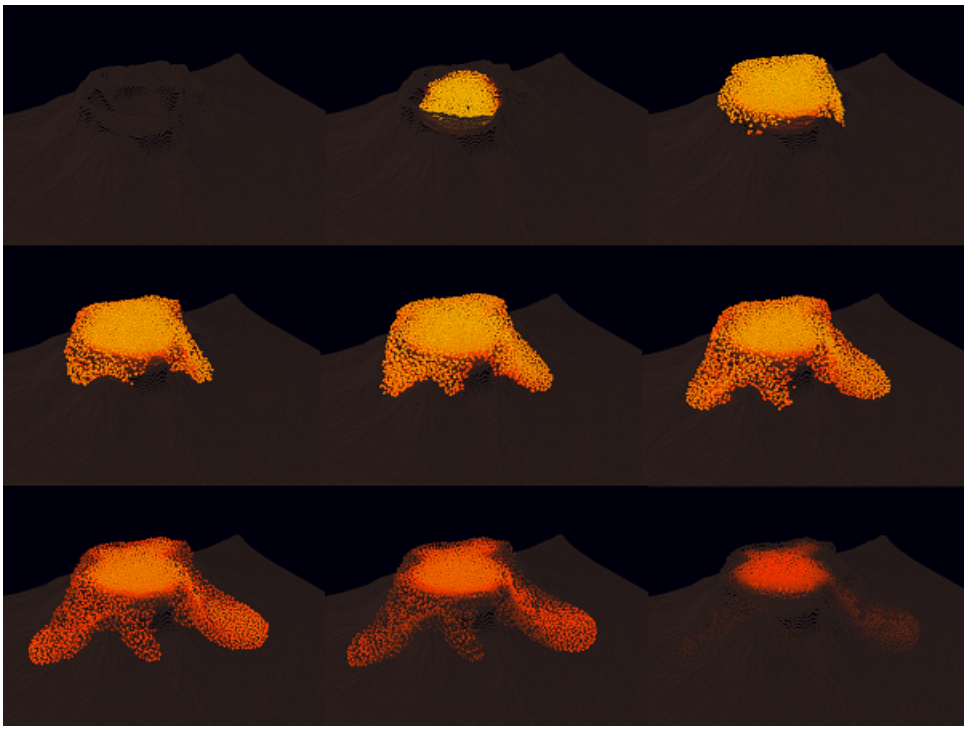
Obrázek 6.1: Dam break scéna.



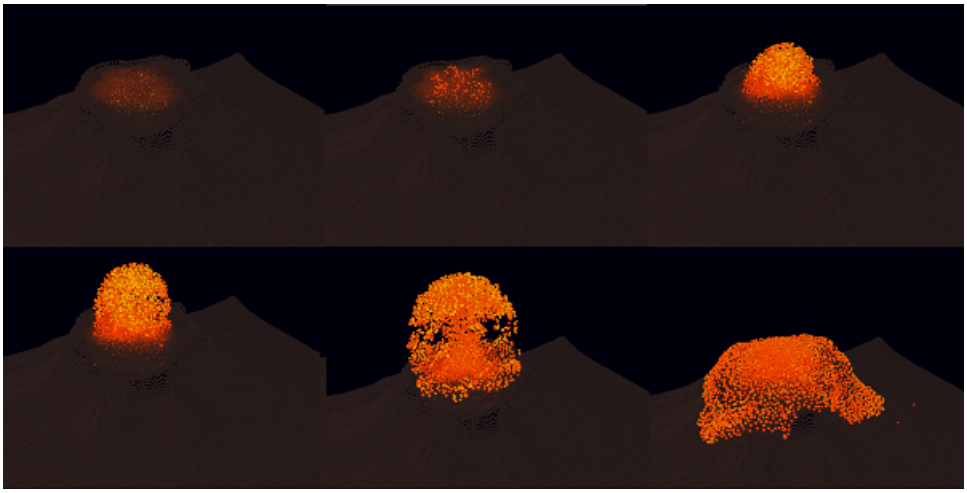
Obrázek 6.2: Dam break scéna – padající částice míří do boční stěny.



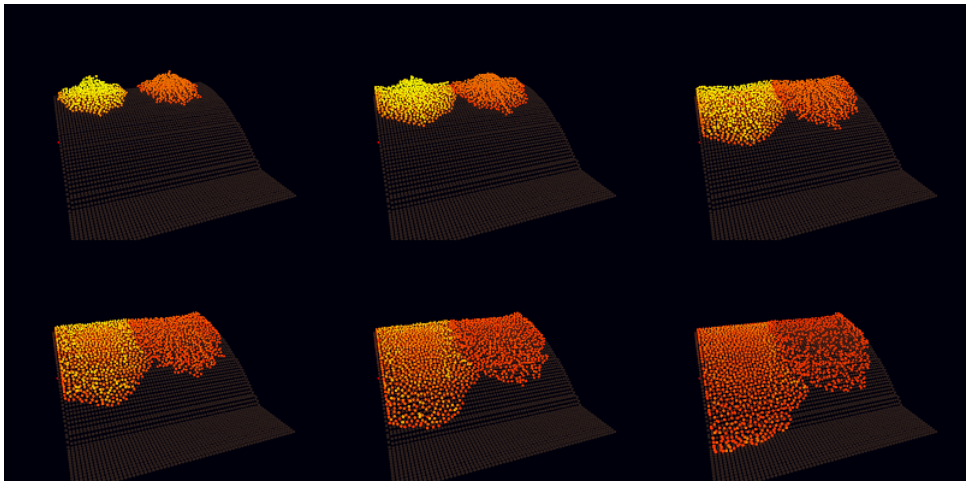
Obrázek 6.3: Dam break scéna s pohyblivou stranou hraničního kvádru.



Obrázek 6.4: Scéna s výbuchem vulkánu a následné ochlazování.



Obrázek 6.5: Zobrazení opětovného výbuch vulkánu po ohřátí ztuhlých částic na povrchu a jejich přeměny zpět na kapalinu.



Obrázek 6.6: Porovnání viskozity lávy dle teploty.

Scéna	**# č. tekutiny**	**# č. kouře**	**# č. terénu**	**FPS**
Dam break	4096	0	0	**260**
	8192	0	0	**190**
	12288	0	0	**150**
	16384	0	0	**110**
	20480	0	0	**90**
	40960	0	0	**40**
Volcano	0	0	65536	**110**
	10000	0	65536	**65**
	10000	8096	65536	**55**
	16000	8096	65536	**45**
	20000	0	65536	**40**
	20000	8096	65536	**36**

Tabulka 6.1: Tabulka s počtem snímků za sekundu a počtem částic. Testováno na procesoru Intel i7-8750H s grafickou kartou Geforce 1050Ti.

6.1.1 Testování

Program byl testován na počítači s procesorem Intel i7-8750H s 6 jádry a grafickou kartou GeForce 1050ti. Výsledky testování rychlosti simulace jsou v tabulce 6.1. Cíl 30 FPS pro 20000 částic byl s rezervou splněn.

6.2 Závěr

Cílem této práce bylo na základě prostudované literatury o SPH a SHP zabývajících se simulací lávy navrhnout a implementovat aplikaci, která by umožnila simulovat pohyb lávy a přidružené jevy v terénu definovaném výškovou mapou.

Práce byla úspěšně navržena a implementována a byly dosaženy hlavní předem stanovené cíle. Byla naimplementována aplikace simulující pohyb lávy na základě *Smoothed particle hydrodynamics* metody. Aplikace dokáže simulovat jevy jako generování kouře, převod teploty a dynamické vlastnosti simulované látky v závislosti na teplotě. Díky moderním možnostem C++ se podařilo aplikaci paralelizovat na více výpočetních jader a dosáhnout několikanásobného zrychlení aplikace oproti stavu bez paralelizace.

Aplikace poskytuje dostatečné vizuální výsledky a případné sporné otázky, jako je například rychlost ztráty teploty do okolí, jsou jednoduše nastavitelné. I přes jednoduchou vizualizační techniku je vizuální stránka aplikace uspokojivá.

Práci by přidalo na kvalitě, kdyby byla zvolena prediktivně–korektivní metoda, což by odstranilo jistou pružnost v kapalině, která je charakteristická pro klasické SPH algoritmy. V rámci této práce se podařilo naplnit všechny hlavní cíle. Výstupem je funkční program, který kvalitně simuluje tok lávy a přidružené jevy a je využitelný i pro simulaci jiných kapalin, při změně několika hodnot.

Bibliografie

[1] Pavel Bokr. "Sopečná činnost a sopky". In: (). URL: http://www.gweb.cz/clanky/clanek-60/ (cit. 18. 01. 2019).

[2] Jan Petránek. *Encyklopedie geologie*. České Budějovice: Jih, 1993. ISBN: 80-900-3512-4.

[3] Shenfan Zhang et al. "Hybrid modeling of multiphysical processes for particle-based volcano animation". In: *Computer Animation and Virtual Worlds* 28.3-4 (2017), e1758. DOI: 10.1002/cav.1758. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cav.1758. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cav.1758.

[4] Jos Stam. *Real-Time Fluid Dynamics for Games*. 2003. URL: http://www.dgp.toronto.edu/people/stam/reality/Research/pdf/GDC03.pdf.

[5] L. Keszthelyi. "A preliminary thermal budget for lava tubes on the Earth and planets". In: 100 (říj. 1995), s. 20411–20420. DOI: 10.1029/95JB01965.

[6] C. Del Negro, L. Fortuna a A. Vicari. "Modelling lava flows by Cellular Nonlinear Networks (CNN): preliminary results". In: *Nonlinear Processes in Geophysics* 12.4 (2005), s. 505–513. DOI: 10.5194/npg-12-505-2005. URL: https://www.nonlin-processes-geophys.net/12/505/2005/.

[7] Dan Stora et al. "Animating Lava Flows". In: *Proceedings of the 1999 Conference on Graphics Interface '99*. Kingston, Ontario, Canada: Morgan Kaufmann Publishers Inc., 1999, s. 203–210. ISBN: 1-55860-632-7. URL: http://dl.acm.org/citation.cfm?id=351631.351687.

[8] Chenfanfu Jiang et al. "The affine particle-in-cell method". In: *ACM Transactions on Graphics (TOG)* 34.4 (2015), s. 1–10.

[9] R. A. Gingold a J. J. Monaghan. "Smoothed particle hydrodynamics: theory and application to non-spherical stars". In: *Monthly Notices of the Royal Astronomical Society* 181.3 (pros. 1977), s. 375–389. ISSN: 0035-8711. DOI: 10.1093/mnras/181.3.375. eprint: http://oup.prod.sis.lan/mnras/article-pdf/181/3/375/3104055/mnras181-0375.pdf. URL: https://doi.org/10.1093/mnras/181.3.375.

[10] Leon B Lucy. "A numerical approach to the testing of the fission hypothesis". In: *The astronomical journal* 82 (1977), s. 1013–1024.

[11] Joe J Monaghan. "Smoothed particle hydrodynamics". In: *Annual review of astronomy and astrophysics* 30.1 (1992), s. 543–574.

[12] Wikimedia Commons. *SPHInterpolationColorsVerbose.svg — Wikimedia Commons, the free media repository″*. [Online; accessed 22-May-2020]. 2019. URL: https://commons.wikimedia.org/w/index.php?title=File:SPHInterpolationColorsVerbose.svg&oldid=347897717.

[13] Matthias Müller, David Charypar a Markus Gross. "Particle-based fluid simulation for interactive applications". In: *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Eurographics Association. 2003, s. 154–159.

[14] Markus Becker a Matthias Teschner. "Weakly compressible SPH for free surface flows". In: *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Eurographics Association. 2007, s. 209–217.

[15] Barbara Solenthaler a Renato Pajarola. "Predictive-corrective incompressible SPH". In: *ACM transactions on graphics (TOG)*. Sv. 28. 3. ACM. 2009, s. 40.

[16] Joe J Monaghan. "Simulating free surface flows with SPH". In: *Journal of computational physics* 110.2 (1994), s. 399–406.

[17] Gui-Rong Liu a Moubin B Liu. *Smoothed particle hydrodynamics: a meshfree particle method*. World scientific, 2003.

[18] Joseph P Morris, Patrick J Fox a Yi Zhu. "Modeling low Reynolds number incompressible flows using SPH". In: *Journal of computational physics* 136.1 (1997), s. 214–226.

[19] Joseph J Monaghan a Andrew Kos. "Solitary waves on a Cretan beach". In: *Journal of waterway, port, coastal, and ocean engineering* 125.3 (1999), s. 145–155.

[20] Mathieu Desbrun a Marie-Paule Gascuel. "Smoothed particles: A new paradigm for animating highly deformable bodies". In: *Computer Animation and Simulation'96*. Springer, 1996, s. 61–76.

[21] C.K. Batchelor a G.K. Batchelor. *An Introduction to Fluid Dynamics*. Cambridge Mathematical Library. Cambridge University Press, 2000. ISBN: 9780521663960. URL: https://books.google.cz/books?id=Rla7OihRvUgC.

[22] Daniel Morikawa et al. "Improvements in highly viscous fluid simulation using a fully implicit SPH method". In: *Computational Particle Mechanics* 6.4 (2019), s. 529–544.

[23] Christopher J Seeton. "Viscosity–temperature correlation for liquids". In: *Tribology letters* 22.1 (2006), s. 67–78.

[24] Barbara Solenthaler, Jürg Schläfli a Renato Pajarola. "A unified particle model for fluid–solid interactions". In: *Computer Animation and Virtual Worlds* 18.1 (2007), s. 69–82.

[25] Markus Ihmsen et al. "Unified spray, foam and air bubbles for particle-based fluids". In: *The Visual Computer* 28.6-8 (2012), s. 669–677.

[26] Maximilian Volkan Baloglu. *Parallelisation and Performance Analysis of a TreeSPH Code for Galaxy Simulations*. 2014.

[27] Alexis Hérault, Giuseppe Bilotta a Robert A Dalrymple. "Sph on gpu with cuda". In: *Journal of Hydraulic Research* 48.S1 (2010), s. 74–79.

[28] Ondřej Novák. "Simulace viskózních kapalin". Dipl. České vysoké učení technické v Praze, Fakulta elektrotechnická, 2007.

[29] Yue Gao et al. "Simulating gaseous fluids with low and high speeds". In: *Computer Graphics Forum*. Sv. 28. 7. Wiley Online Library. 2009, s. 1845–1852.

[30] Afonso Paiva et al. "Particle-based viscoplastic fluid/solid simulation". In: *Computer-Aided Design* 41.4 (2009), s. 306–314.

[31] *FluidX3D - Free-Surface VoF LBM on the GPU visualized with Marching Cubes in Real Time*. URL: https://www.youtube.com/watch?v=1ww8qRCMc4s.

[32] Jos Stam. "Stable fluids". In: *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. 1999, s. 121–128.

[33] *Implementing SPH in 2D*. URL: https://bigtheta.io/2017/07/08/implementing-sph-in-2d.html.

[34] Rama Hoetzlein. *Fluids v.2*. URL: http://www.rchoetzlein.com/.