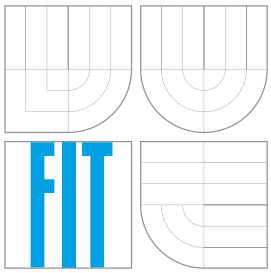
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

SIMULACE KAPALIN VE 2D

SIMULATION OF FLUIDS IN 2D

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

VLASTIMIL PAZDERA

VEDOUCÍ PRÁCE
SUPERVISOR

prof. Dr. Ing. PAVEL ZEMČÍK,

BRNO 2017

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Pazdera Vlastimil**
Obor: Informační technologie
Téma: **Simulace chování kapalin ve 2D**
Simulation of Fluids Behavior in 2D
Kategorie: Počítačová grafika

Pokyny:

1. Nastudujte dostupnou literaturu na téma simulace chování kapalin se zaměřením na simulaci ve 2D (v řezu).
2. Vyberte vhodnou metodu pro simulaci chování kapalin, která by byla implementovatelná i na mobilním zařízení s omezeným výpočetním výkonem.
3. Navrhněte způsob implementace vybrané simulační metody a diskutujte její vlastnosti.
4. Implementujte metodu a demonstrujte ji na vhodném příkladě.
5. Diskutujte dosažené vlastnosti a možnosti pokračování práce.

Literatura:

- Dle pokynů vedoucího práce

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Zemčík Pavel, prof. Dr. Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Tato práce se zabývá simulací kapalin ve 2D řezu na mobilních zařízeních pomocí techniky Smoothed Particle Hydrodynamics implementovaném v herním enginu Unity. Výsledný program je použitelný na mobilních zařízeních jako stavební prvek her a interaktivních aplikací. Pomocí parametrů lze měnit vlastnosti kapaliny jako např. viskozita. Práce se zaměřuje na co největší využitelnost v mobilních aplikacích a zohledňuje požadavky a limity zařízení k dosažení co nejlepších vizuálních a fyzikálních vlastností kapaliny.

Abstract

This thesis deals with the simulation of fluids in 2D cut for mobile devices using technique Smoothed Particle Hydrodynamics implemented in game engine Unity. The resulting program is usable on mobile devices as a structural element of the games and interactive applications. With the parameters you can alter the properties of liquids for example viscosity. The work focuses on the greatest applicability in mobile applications and takes into account the requirements and limits of the devices to achieve the best visual and physical properties of the simulated liquid.

Klíčová slova

Simulace kapalin v reálném čase, simulace ve 2D řezu, interaktivní simulace, fyzikální simulace, částicový systém, Smoothed Particle Hydrodynamics.

Keywords

Simulation of fluids real-time, interactive simulation, physical simulation, simulation in 2D, particle system, Smoothed Particle Hydrodynamics.

Citace

Vlastimil Pazdera: Simulace kapalin ve 2D, bakalářská práce, Brno, FIT VUT v Brně, 2017

Simulace kapalin ve 2D

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením prof. Dr. Ing. Pavla Zemčíka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Vlastimil Pazdera
18. května 2017

Poděkování

Na tomto místě bych rád poděkoval prof. Ing. Pavlu Zemčíkovi, Dr. za vedení a užitečné rady pro vypracování této práce.

Obsah

Kapitola 1

Úvod

Kapaliny nás obklopují a hrají důležitou roli v každodenním životě. Proto je žádoucí jejich přítomnost i ve virtuálním prostředí, kde může interakce s nimi podpořit realističnost virtuální scény. Pro dosažení věrohodnosti je nutné, aby chování kapaliny a její fyzické projevy odpovídaly co nejvíce skutečnosti. Simulace kapalin ve virtuálním prostředí je však složitý úkol, který vyžaduje speciální techniky vyvinuté právě k tomuto účelu. Pro dosažení realistických fyzikálních nebo alespoň částečně realistických výsledků je též třeba mnoho výpočetního výkonu. Jelikož se kapaliny a interakce s nimi hojně uplatňují v počítačových hrách, animovaných filmech nebo v interaktivních aplikacích, má v současné době ve světě simulací svoje důležité místo vývoj nových technik, jež budou mít realističtější výsledky a budou méně náročné na výpočetní zdroje počítače.

Základní kameny výpočtů dynamiky tekutin položili Claude Navier a George Stokes, kdy už v 19. století nezávisle na sobě zformulovali rovnici popisující dynamiku kapalin. Tato rovnice je též známá jako Navier-Stokesova. Z této rovnice, která komplexně popisuje chování kapalin, čerpají poznatky i techniky simulace kapalin na počítačích. Od té doby je na simulace nahlíženo dvěma základními pohledy - Lagrangeovým a Eulerovým.

Cílem práce bylo prostudovat oblast simulace kapalin. Na základě znalosti technik a jejich vlastností vybrat takovou, kterou lze použít pro simulaci ve 2D řezu a která bude implementovatelná i na mobilních zařízeních. Dále navrhnout způsob její implementace, realizovat ji a na jejím základě poté vytvořit ukázkovou aplikaci, jež bude demonstrovat dosažené výsledky.

V Kapitole 2 jsou vysvětleny základní pojmy jako kapalina a její vlastnosti, dále je zde popsáno základní rozdělení technik, vysvětleny pohledy a možnosti pro simulaci kapalin s různými vlastnostmi, které lze použít v této práci použít. Dále jsou zde popsány dostupné prostředky a technologie pro implementaci a možnosti vykreslování povrchu simulované kapaliny.

V kapitole 3 jsou formulovány cíle práce. Také jsou zde popsány důvody volby metod vhodné pro náš případ simulace. Dále je zde popsán návrh řešení této metody, její klady a zápory, možná úskalí a problémy s nimiž je možné se setkat během řešení jednotlivých kroků.

Kapitola 4 se zabývá implementací zmíněné metody podle předchozího návrhu řešení. Je zde popsáno řešení složitějších otázek při implementaci zadaného programu.

V kapitole 5 jsou provedeny testy implementace, zhodnocení vizuálního projevu, zhodnocení provedených optimalizací. Závěru jsou shrnuty výsledky práce.

Kapitola 2

Teoretický základ práce

V části 2.1 je vysvětleny pojmy tekutiny, kapaliny a jaké mají vlastnosti. V části 2.2 je vysvětlena rovnice chování všech tekutin ze které vychází simulační metody. V části 2.3 jsou uvedeny a popsány základní pohledy reprezentace kapalin při simulacích. Stručně jsou popsány některé metody používané pro tento účel. Část 2.4 popisuje blíže techniku simulace viskózních kapalin, která je využita v této práci. Další část 2.5 popisuje některé vhodné prostředky pro implementaci programu. Poslední část 2.6 se zabývá problematikou vykreslování celistvého objemu a povrchu kapaliny.

2.1 Co je to kapalina

Kapalina je látka složená z molekul, které mohou volně proudit prostorem, a jsou ovlivňovány některými silami přítomnými v tomto prostoru. Patří společně s plyny(i plazmou) mezi tekutiny. Společnou vlastností všech tekutin je tekutost, neboli neschopnost udržet svůj stálý tvar kvůli vlivu smykového napětí. Na všechny tekutiny lze také z fyzikálního hlediska nahlížet jako na látky složené z molekul, které mohou volně proudit prostorem, a jsou ovlivňovány některými silami přítomnými v tomto prostoru.K tekutinám se většinou řadí také sypké látky, které jsou sice pevného skupenství, ale splňují kritérium tekutosti.

Kapaliny se však od plynů liší především stlačitelností a rozpínavostí. Plyny jsou rozpínavé, kdežto kapaliny vytvářejí volnou hladinu. Kapalná tělesa si zachovávají i pří proměnném tvaru stálý objem, a jsou-li v klidu , vytvářejí v tíhovém poli Země volný vodorovný povrch. Plynná tělesa nemají stálý tvar ani objem, a proto nejsou schopna vytvořit volný povrch. tvar a objem plynného tělesa se mění podle tvaru a objemu nádoby v níž se nachází.

Kapaliny jsou velmi málo stlačitelné. Naopak plyny jsou stlačitelné velmi jednoduše, ovšem zvětší-li se objem nádoby, vyplní plyn celý její prostor.

Kapaliny a plyny se liší různou tekutostí. Vzájemná pohyblivost částic je větší u plynů a tedy je tekutější, ale ani všechny kapaliny nemají stejnou tekutost Například snadněji se přelévá voda než olej nebo hustý sirup. Příčinnou různé tekutosti u látek je vnitřní tření neboli vizkozita tekutin[9].

Kapaliny nás obklopují a hrají důležitou roli v každodenním životě. Realističnost zobrazení a interakce s kapalinami může výrazně podpořit realističnost v počítačových hrách nebo interaktivních aplikacích. Zde je často na úkor plynulosti a zachování přijatelného počtu snímků za vteřinu třeba zanedbat vysokou přesnost simulace, která zde není oproti přesným vědeckým fyzikálním simulací vyžadována. Ačkoli se chování těchto látek zdá v běžném životě jednoduché, jejich simulace ve virtuální scéně je velmi komplexní a tudíž velmi

náročná.

2.1.1 Vlastnosti kapalin

Jejich nejvýnamější společnou vlastností je neschopnost udržet svůj stálý tvar díky snadnému vzájemnému pohybu částic. Nemají tedy stálý tvar a zaujímají tvar nádoby, vníž se nacházejí a dají se tedy i přelévat. Jejich molekuly jsou v neuspořádaném pohybu a nejsou pevně vázané [5]. Tohoto komplexního chování je tedy třeba dosáhnout, aby bylo chování věrohodné.

Viskozita

Speciální vlastností tekutin je viskozita, což je veličina charakterizující vnitřní tření mezi molekulami látky a závisí především na velikosti přitažlivých sil mezi nimi. Tato vlastnost se projevuje pouze tehdy, pokud není tekutina v klidu. Kapaliny s větší přitažlivou silou mezi částicemi mají větší viskozitu, tedy větší brzdění pohybu samotné tekutiny nebo těles v tekutině. Jako příklad kapalin s velkou viskozitou můžeme uvést třeba med, hustý sirup nebo glycerín, mezi kapaliny s malou viskozitou patří např. voda.

Tlak

Tlak je důležitá veličina, která charakterizuje stav tekutiny v klidu. Tlak odkazuje na normálové síly, kterými kapalina působí na nádobu a ostatní objekty nacházející se v ní.

Hustota

Hustota vyjadřuje jaké množství hmoty je v každé malé částici v kapalině. I jednotlivé částice mohou mít rozdílnou hustotu, tudíž lze simulovat smíchání rozdílně hustých látek - jako přilití oleje do vody atd. olej by měl postupně vystoupat nahoru na hladinu vody.

Teplota

Teplota určuje kolik tepla je přítomno v jednotlivých částečkách objemu kapaliny. Sama o sobě neovlivňuje pohyb částic v kapalině, ale může ovlivňovat tlak a hustotu, které pohyb ovlivňují.

Všechny jevy, ve kterých hrají roli tekutiny tedy mají společný znak a tím je proudění částic. Se simulací kapalin souvisí i simulování kouře či ohně, kde se částice pohybují podobným způsobem ve vzduchu.

2.2 Rovnice chování tekutiny

Jak již bylo zmíněno v úvodu, tyto rovnice tvoří základ výpočtu dynamiky všech tekutin. Lze je aplikovat na různé typy tekutin, tato práce se však zabývá pouze tzv. Newtonovskými kapalinami - takovými látkami, které jsou nestlačitelné a dynamická viskozita je konstanta úměrnosti mezi napětím a rychlostí deformace. Materiálovou konstantou charakterizující newtonovskou látku je viskozita[6].

Jak uvádí [1], je tok částic tekutiny řízen tzv. nestlačitelnou Navier-Stokesovou rovnicí v Laplaceově formě (soustava parciálních diferenciálních rovnic). Tyto rovnice počítají se

třemi základními vlastnostmi tekutin - rychlostí, hustotou a tlakem. Rovnice jsou zapsány jako:

$$\frac{\delta \vec{u}}{\delta t} + \vec{u} + \bigtriangledown \vec{u} + \frac{1}{\rho} \bigtriangledown p = \vec{p} + v \bigtriangledown \cdot \bigtriangledown \vec{u} \tag{2.1}$$

$$\bigtriangledown \cdot \vec{v} = 0 \tag{2.2}$$

Kde $\bigtriangledown$ je operátor gradientu, $\bigtriangledown \cdot$ je diferenciální operátor divergence a $\bigtriangledown \cdot \bigtriangledown$ je Laplaceův operátor, který je definovaný jako divergence gradientu daného skalárního pole.

Popis proměnných:

- $\vec{u}$ je rychlost tekutiny
- ρ je hustota tekutiny (pro vodu asi 1 000 kg/m3, vzduch asi 1.3 kg/m3)
- p je tlak (síla plošné jednotky, kterou tekutina působí na okolí)
- $\vec{g}$ je gravitační zrychlení (obyčejně vektor (0.0, -9.81, 0.0) m/s2) předpokládá se, že osa Y směřuje vertikálně nahoru a osy X a Y jsou horizontální.
- v je tzv. kinematická viskozita, která určuje, jakou mírou je tekutina viskózní.
- t je čas

První diferenciální rovnice 2.1 se nazývá rovnice hybnosti a charakterizuje pohyb tekutiny při působení ostatních sil na ni. Druhou diferenciální rovnicí **??**navStok2) je podmínka nestlačitelnosti, která zajišťuje zachování objemu jednotlivých částic a simuluje tak nestlačitelnost skutečné tekutiny.

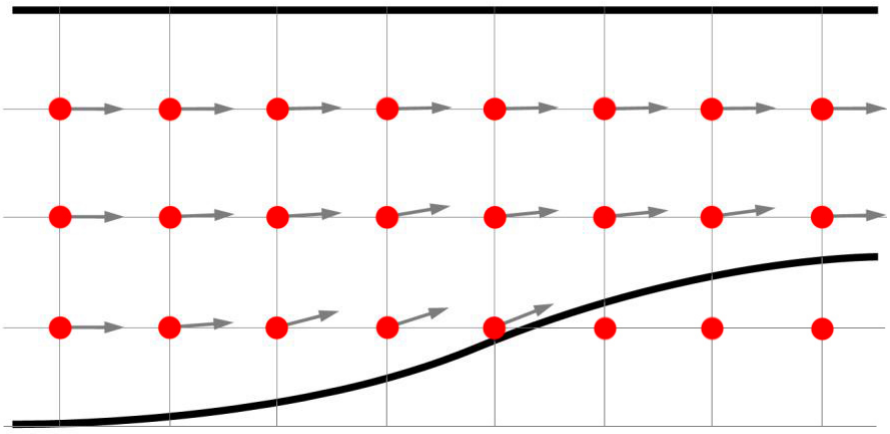
2.3 Základní metody reprezentace kapalin

Existují dva základní pohledy na to, jak lze popsat pohyb kapalin - Eulerův pohled a Lagrangeův pohled. Liší se od sebe jak reprezentací kapalin v prostoru, tak jejich vyhodnocením. Zatímco Eulerův pohled používá k reprezentaci pevnou 2D nebo 3D mřížku od sebe navzájem rovnoměrně rozmístěných bodů, v nichž se počítají pseudoveličiny, Lagrangeův pohled využívá kolekci mezi sebou interagujících částic, které se volně pohybují a navzájem se ovlivňují. Oba pohledy jsou užitečné, často se také kombinují [5].

2.3.1 Eulerův pohled

Eulerova mřížka používá pevné nepohyblivé body ve fixním 2D nebo 3D poli (krychle složená z menších krychlí) k reprezentaci toku nebo pohybu tekutiny. Přes tyto body se v čase kapalina pohybuje. Pouze v těchto bodech měříme všechny veličiny jako je tlak, hustota, teplota atd.

V tomto typu simulace musí být celá tekutina zachycena v mřížce. Kapalina se tedy nemůže dostat mimo. Často je třeba modelovat pomocí mřížky situaci, kdy kapalina není v prostoru ideálního tvaru čtverce či krychle, a tudíž scéna obsahuje prázdné buňky, které jsou nevyužity. Metoda je tedy velice náročná na paměť. Lze však použít některé optimalizační techniky například dělení prostoru do oktanového stromu, aby zaplněné části prostoru měli větší buňky a okraje drobnější.

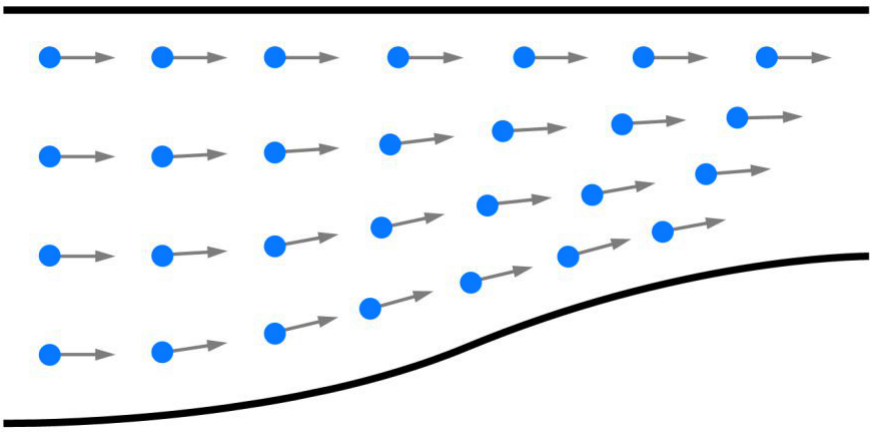


Obrázek 2.1: reprezentace kapaliny pomocí Eulerova pohledu

Často se používá k velmi přesným simulacím toku kapalin, ale je jen málo známých použití v interaktivních aplikacích, kde se pro svoje nevýhody a fixní souřadnice nevyužívá skoro vůbec. Také se zde obtížněji vyhodnocují reakce na kolize s okolními předměty, přitom rychlé a snadné vyhodnocení kolizí předmětů je např: ve hrách či filmech při interakci s kapalinou velmi žádoucí. Metoda má také problémy se zachováním hmoty.

2.3.2 Lagrangeův pohled

Tento pohled využívá k reprezentaci tekutiny pohyblivé body, které se mohou volně pohybovat prostorem a nejsou vázány žádnou mřížkou [3]. Těmto bodům se říká částice. Každá částice reprezentuje malé množství kapaliny o určité hmotnosti a reprezentuje tak část jejího objemu. Masa kapalina je tak tvořena množinou částic pohybujících se v prostoru a ovlivňujících se navzájem silami. Každá částice také nese údaje o své pozici a rychlosti. Jiné vlastnosti těchto částic a celé kapaliny se dají získat z vyhodnocování dráhy jejich pohybu [5].



Obrázek 2.2: reprezentace kapaliny pomocí Lagrangeova pohledu

Největší nevýhoda částicových simulací je obtížná reprezentace hladkého povrchu kapalin. Toho lze však dosáhnout použití většího množství drobnějších částic, což však činí simulaci náročnější. Obecně platí, že čím více bodů budeme vyhodnocovat, tím více bude simulace výpočetně náročnější. Obtížně se také dosahuje povrchového napětí. Výhodou je však obecně menší výpočetní náročnost simulace oproti Eulerovu přístupu.

2.4 Smoothed particle hydrodynamics

Smoothed particle hydrodynamics (dále jen SPH) je technika, založená na Lagrangeově pohledu využívající množinu částic k jejich reprezentaci objemu kapaliny. Vyvinuli ji R.A. Gingold, J.J. Monaghan a L.B. Lucy v roce 1977 a detailně je popsána v publikaci[7]. Původně byla vytvořena k řešení a simulaci složitých astrofyzikálních problémů, lze ji však použít i k simulaci viskózních kapalin.

Jedná se o druh řešení nestlačitelné Navier-Stokesovy rovnice kapaliny. Metoda pracuje na principu konvolučního filtrování, kdy počítáme s parametry jednotlivých částic a jejich blízkých sousedních částic a pomocí tzv. vyhlazovacích jader (smoothing kernels) interpoluje veličiny, které při sledování drah částic v prostoru těžko určíme. Pomocí těchto interpolací lze vypočítat tlak, hustotu a další atributy v místě částice. Podle vypočtených hodnot se upravuje chování a parametry částic. Jak je uvedeno v práci [2] praktických metod řešení SPH je mnoho.

2.4.1 Výpočet SPH

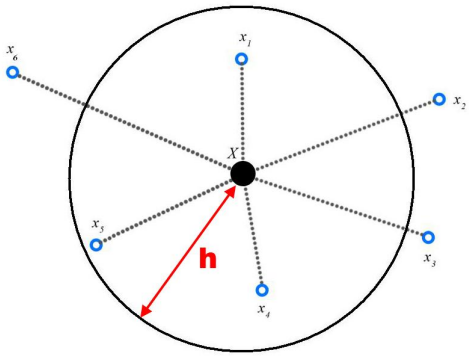
SPH je interpolační metoda pro částicové systémy. Cílem je získání aproximované hodnoty skalární veličiny. Skaláry, které chceme znát, jsou hustota, tlak a teplota v každém místě tekutiny. Abychom tyto hodnoty pro konkrétní bod získali, vypočítáme vážený průměr z parametrů částic v okolí tohoto bodu. Jehož každá částice má danou svoji pozici v prostoru a skalární veličinu A_j. Potom je možné pro libovolný bod v prostoru (X [x, y, z]) tuto veličinu interpolovat pomocí několika nejbližších částic v poloměru r užitím vyhlazovacího jádra (W) tak, že:

$$A_s(r) = \sum_j m_j \frac{A_j}{\rho_j} W(r - r_j, h) \tag{2.3}$$

kde j iteruje přes všechny částice, m_j je hmotnost částice j, r_j je její pozice v prostoru, ρ_j je hustota částice a A_j je hodnota skalární veličiny na pozici r_j , kterou chceme vypočítat. Funkce $W(r, h)$ je zvaná vyhlazovací jádro s vyhlazovacím poloměrem h. Částice se které budou nejvíce výpočet veličiny ovlivňovat určuje právě vyhlazovací poloměr h.

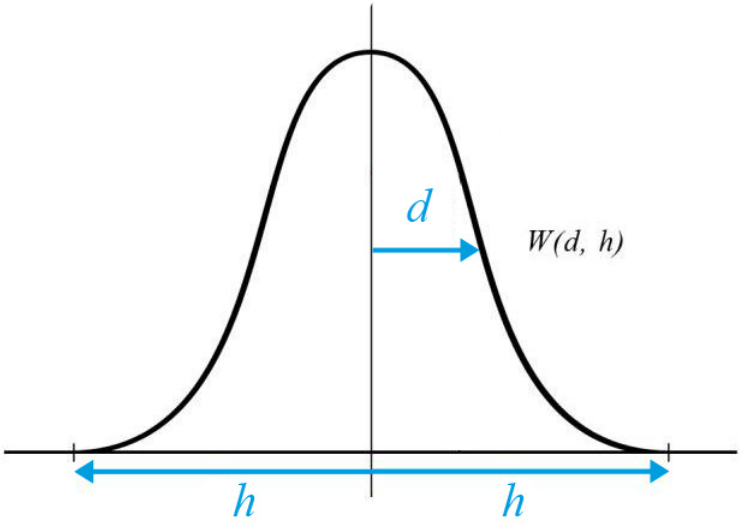
Jádro ovlivňuje výpočet tak, že funguje jako filtr a bližší částice mají na právě počítanou veličinu částice vetší vliv, než ty vzdálené. Tedy normalizuje vzdálenost od částice od bodu $X[x, y, z]$. Pokud je vzdálenost menší než poloměr h, můžeme vypočítat váhu částice P_j pomocí vyhlazovacího jádra W. Pokud je vzdálenost menší než vyhlazovací poloměr, váha částice je nula. V praxi se často používá omezení podle vzorce 2.4, kde pokud se částice nachází mimo poloměr vyhlazování a její váha je nula, není třeba ji do výpočtu zahrnovat, což šetří výpočetní zdroje.

$$W(r, h) = 0, |r| > h \tag{2.4}$$



Obrázek 2.3: Určení vzdáleností od bodu X ve vyhlazovacím poloměru h. Do výpočtů jsou zahrnuty pouze částice x_1, x_4 a x_5

Dále musí musí být obor hodnot jádra nezáporný na celém jeho definičním oboru. Většinou se používají jádra tvaru zvonu. Grafem takového jádra je pak Gaussova křivka jako je na obrázku 2.4.



Obrázek 2.4: Znázornění výpočtu váhy pomocí vyhlazovacího jádra W s Gaussovou křivkou o poloměru h ve vzdálenosti d od počítané pozice v prostoru

2.5 Dostupné technologie pro simulaci na Smartphonech

2.5.1 Nativní vývoj

Android

Android je plně otevřená open-source platforma. Jejím hlavním vývojářem je firma Google. Je licencovaná pod licencí Apache / MIT, jenž umožňuje jej rozšiřovat a používat pro jaký-

koliv účel. Vývojáři tak mají přístup ke kódu celé platformy. To umožňuje velice podrobně implementaci programů přizpůsobit zařízení. Na vývoj na systém Android není potřeba zakoupit žádná licence, a je tedy je možné jej ihned použít. Systém byl vytvořen tak, aby fungoval na široké škále hardware.

Na platformě Android se vyvíjí pomocí SDK kitu, které obsahuje knihovny, emulátor a debugger. Dále je možné využít ADT (Android Development Tools – vývojářské nástroje pro Android) plugin pro vývojové prostředí Eclipse, který umožní vývojářům ovládat SDK, ladit aplikace, a to buď v emulovaném telefonu nebo přímo v zařízení. U emulovaných telefonů lze nastavit různé rozlišení displejů nebo různý hardware.

Hlavním jazykem pro vývoj na Android je jazyk Java, v některých případech je možno použít jazyky C/C++ a poté pomocí NDK (Native Development Kit – nativní vývojářský balík) ji přeložit přímo do nativního kódu. Po napsání aplikace v jazyce Java je kód přeložen do bytekódu pro běh na virtuálním stroji.

iOS

iOS je operační systém který vznikl v roce 2007 výhradně pro zařízení firmy Apple. Jazyk pro platformu iOS se jmenuje Objective-C. Vznikl v 80. letech minulého století jako objektová nadstavba nad jazykem C. Jedná se tedy o objektově orientovaný jazyk, jehož syntaxi převzali autoři z jazyka Smalltalk. Protože je kompilátor Objective-C zpětně kompatibilní, lze pro iOS vyvíjet také v jazyce C, což se moc v praxi nepoužívá. Pro vývoj aplikací je určené vývojové prostředí Xcode, které lze spustit pouze na počítačích se systémem Mac OS X. Pro vývoj a překlad aplikací je tedy nutné vlastnit počítač s operačním Mac OS X.

Windows phone

Operační systém vyvíjený společností Microsoft. Je to nástupce dříve úspěšného systému pro mobilní zařízení Windows Mobile. V současnosti jsou používány verze 7 a 8. Tento OS také sjednocuje uživatelské rozhraní mezi chytrými telefony, osobními počítači a tablety. Pro vývoj Windows Phone aplikací je určena technologie Silverlight. Druhá možnost jak tvořit aplikace, je pomocí Microsoft XNA, který je určen pro tvorbu 2D a 3D her využívajících DirectX. Obě technologie využívají framework .NET a je možné programovat v jazycích C#, F#, VisualBasic .NET. Jako vývojové nástroje se používá Microsoft Visual Studio s emulátorem telefonů a nástrojem Microsoft Expression Blend pro tvorbu grafiky.

Nevýhodou nativního řešení je však vývoj zaměřený pouze na jeden zvolený operační systém. Výsledný program se bude muset psát v každém jazyce znovu a vyvíjet pro každou platformu zvlášť se všemi náležitostmi a použitím nástrojů specifických pro každou platformu. Proto je výhodnější použít multiplatformní engine, který podporuje spoustu mobilních platforem, jako jsou ty popsáné dále.

2.5.2 Multiplatformní vývoj

Pro vývoj her, různých grafických aplikací a simulací je však možné použít i multiplatformní enginy. Jejich obrovskou výhodou je to, že lze program implementovat pouze jednou v jednom jazyce a poté jej portovat na širokou škálu různých platforem, což vývojářům značně usnadňuje práci a je to pro ně výhodné i z ekonomického hlediska vývoje. Oblíbenost těchto herních enginů dokazuje i fakt, že jsou používány od nezávislých herních studií po nejznámější firmy na poli her. V následující části jsou popsány dva nejvhodnější kandidáti pro tvorbu simulačního programu.

Cocos2D

Cocos2D je open sourcový multiplatformní engine. Je používaný k tvorbě her, aplikací a ostatních multiplatformních interaktivních pogramů. Skládá se z několika větví - Cocos2d-objc, Cocos2d-x, Cocos2d-html5 a Cocos2d-XNA, z ničchž každá je cílená na jinou platformu a jejich API (aplikační rozhraní) podporuje různé jazyky. Stejně jako všechny 2D grafické enginy, všechny verze pracují na základě manipulace a vykreslování spritů ve scéně, což je vlastně jednoduchý 2D obrázek. Sprity jsou skládany do scény, které tvoří menu nebo herní úrovně hry. Sprity mohou být řízeny kódem, událostmi, akcemi nebo animacemi.

Podporované platformy a jazyky:

- Větev Cílová platforma Jazyk API
- Cocos2d Windows, OS X, Linux Python 2.6, 2.7 or 3.3+,Objective-C
- Cocos2d x iOS, Android, Tizen, Windows 8, WP 8, Linux, Mac OS X C++, Lua, JavaScript
- Cocos2d-ObjC iOS, Mac OS X, Android Objective-C, Swift
- Cocos2d-html5 HTML5-ready browsers JavaScript
- Cocos2d-xna WP 7 & 8, Windows 7 & 8, Xbox 360 C#

Pro vývoj na mobilní zařízení je tedy nejvíce vhodná větev Cocos2d-x, která pokrývá nejrozšířenější mobilní platformy současnosti ale podporuje také operační systémy PC.

Unity

V této podsekci je více přiblížen herní engine Unity3D, protože je dále v této práci použit k implementaci programu. Je zde volně citováno z webových stránek [10] a online dokumentace k aplikačnímu rozhraní a uživatelského manuálu [11]. Jsou zde dále popsány důležité části editoru, části aplikačního rozhraní, pojmy a prvky využité při implementaci.

Unity je multiplatformní herní engine vyvinutý společností Unity Technologies. Je využíván pro vývoj počítačových her a různých simulací pro PC, konzole, mobilní platformy, chytré televize, virtuální realitu, webové prohlížeče využívající WebGL. V současnosti podporuje 27 různých platforem. S enginem se většinou pracuje přes Unity Editor, ve kterém lze aplikaci vytvořit jen jednou a pak exportovat na jakoukoliv platformu. Aplikační rozhraní je poskytováno v jazycích C# Javascript a ve starších verzích také jazykem Boo, a lze je v rámci jednoho projektu používat současně. Jeho velkou výhodou je také silná komunita vývojářů sdružující se na oficiálním fóru, kde lze při tvorbě čerpat nespočet rad, informací a zkušeností od úspěšných profesionálních vývojářů.

Pojmy a architektura projektu

Projekt se skládá ze scén, které fungují jako samostatné herní úrovně hry nebo menu. Základní stavební jednotkou ve scéně je pak *GameObject*. Interakce těchto objektů ve scéně potom tvoří samotnou herní náplň.

GameObject

Herní objekty jsou většinou pohyblivé a nepohyblivé entity jako například panáček, auto, strom atd. Tyto objekty spolu interagují ve scéně, jejich vzhled a chování lze řídit přidáváním množství tzv. komponent. Mohou však být i neviditelné a ovládat pouze chování scény či jiných objektů. Objekty lze také skládat do dědičné hierarchie např: panáček se skládá z rukou, nohou a trupu.

Component

Komponentou se rozumí prvek, co přidává nějakou vlastnost hernímu objektu, jako je geometrie kolizního modelu objektu, vykreslení povrchu nebo například skripty - tedy instance tříd programů. V následujícím odstavci jsou popsány důležité komponenty s nimiž je možné se při práci setkat nejčastěji.

Camera

V každé scéně se nachází implicitně jedna kamera. Kamera zaznamenává a zobrazuje prostor a předměty v něm hráči. Do scény lze umístit několik kamer a každou nastavit pomocí masky vrstev tak, aby vykreslovala zvolenou vrstvu s objekty. Lze nastavit i pořadí vykreslování jednotlivých vrstev.

Script

Jsou to instance tříd skriptů v GameObjectu. Pomocí nich je implementováno chování. Důležitou vlastností také je, že jimi lze programově ovládat většinu vlastností ostatních komponent. Jejich vzhled a chování lze řídit skripty - instancemi tříd programů. Skripty lze psát v jazycích C# , Javascript nebo Boo. Většina tříd, které programátor vytvoří, zdědí vlastnosti z vestavěné třídy MonoBehaviour, kde jsou implementovány hlavní třídy a metody pro práci v enginu. Pro práci a programovému řízení komponent je tedy tato třída nutností.

Typická stavba souboru skriptu (zde v jazyce C#) má předefinovány základní metody, které je nutné použit při implementaci:

Awake

Metoda se volá jako první a slouží k inicializaci podobně jako konstruktor. Volá se během nahrávání scény do paměti. Většinou se používá na inicializaci referencí mezi skripty i na inicializaci běžných proměnných. Volá se však jen jednou po aktivování nebo instanciaci komponenty.

Start

Druhá metoda, jenž také slouží k inicializaci. Volá se ale až poté, když je celá scéna nahrána do paměti zařízení. Stejně jako Awake se volá jednou po aktivování komponenty. Také se používá na inicializaci běžných proměnných.

Update

Hlavní metoda, která se volá každý snímek, pokud je komponenta skriptu aktivní. Implementuje se v ní většina herní logiky a chování, tedy to, co chceme aby bylo prováděno cyklicky každý snímek. Například načítání vstupů, pohyby s objekty z hlediska enginu nefyzikálními - ty, co nevyužívají fyzikální komponenty enginu. Protože závisí na počtu snímků za vteřinu, časy provádění mezi snímky se často liší. U provádění simulací fyziky však chceme konstantní interval, aby nedocházelo k chybnému nebo nelogickému chování. K řízení chování fyzikálních objektů proto slouží metoda FixedUpdate.

FixedUpdate

Nevolá se však každý snímek, ale volá se v konstantních intervalech nezávisle na nich. Používá se k řízení komponent, které používají fyzikální část enginu.

Transform

Transform, který má schopnost uchovávat a měnit pozici, rotaci a měřítko GameObjectu ve scéně. Každý objekt i ten bez komponent musí mít právě jednu tuto základní komponentu. Transform komponenta také dovoluje koncept zvaný *rodičovství*, který je užitečný při práci s GameObjecty.

Sprite Renderer

Sprity jsou vykreslovány komponentou Sprite Renderer se používá k vykreslování 2D objektů zvaných Sprity. Sprity jsou v podstatě 2D textury. Tato komponenta se používá k vykreslování ve 2D. Ve 3D, kde se používá komponenta Mesth Renderer, se většinou nepoužívají, lze je však použít také. Sprity mají při vykreslování několik výhodných technik, které usnadňují vykreslování a šetří výkon grafické karty. Různé Sprity se totiž dají kombinovat do větších textur tzv. *Atlasů*, které se poté vykreslí naráz. Všechny Sprity však musí splňovat kritéria. Všechny Sprity v Atlasu musí používat k vykreslování stejný materiál a musí být ve stejné vykreslovací vrstvě. Pokud splňují podmínky, má celý Atlas jeden Drawcall - čili jednotku vykreslení. Současná zařízení vydrží z grafického hlediska okolo desítek Drawcallů, průměrné zařízení asi okolo 50.

Profiler

Unity Profiler je nástroj který slouží k optimalizaci scény a celé hry. Měří kolik času tráví program v různých úsecích celé hry jako je vykreslování, animací nebo herní logiky. Dá se tak jednoduše analyzovat výkon GPU, CPU, paměti, vykreslování nebo audia a odhalit slabá místa, které se hodí přepracovat efektivnějším řešením.

Data se dají sbírat, zobrazovat i nahrávat v reálném za běhu hry v editoru nebo přímo ze zařízení ve vestavěném oknu Profileru. Sesbíraná data jsou poté zobrazena v časové ose jak je zobrazeno na obrázku xxxx.....x..x.x..x takže můžeme vidět přímo snímky nebo oblasti ve kterých dochází k výkyvům výkonu - tam, kde výpočet trval delší dobu než u ostatních snímků. Kliknutím do okna profileru a vybráním snímku, llze v dolní části vidět podrobnější popis statistik zvoleného snímku. Měření v Unity Profileru funguje tak, že jsou do programátorem vytvořeného kódu na určitá místa vkládány dodatečné měřící instrukce. Při standardním režimu to nemá však téměř žádný negativní dopad na výkon. Profiler má i druhý režim - Deep Profiling. V tomto režimu jsou výpisy mnohem podrobnější, že lze vysledovat i metodu či funkci nebo jejich části, které dělají problémy, což je velmi užitečné. To však má velmi negativní vliv na výkon aplikace a v některých případech to vyžaduje silný hardware.

2.6 Vykreslování povrchu kapalin

V následující části práce jsou popsány některé vhodné metody pro vykreslování výsledků simulace. Je kladen důraz na metody, které bude jednoduché implementovat a použít při tvorbě programu, jelikož se tato práce se nesoustředí na dokonalou kvalitu vykreslování.

2.6.1 Metaballs

Metaballs je technika vytváření tzv. isosurface ve 3D prostoru. Bez větších obtíží ji lze však použít i ve 2D prostoru. Tuto techniku vynalezl Jim Blinn v 80. létech 19. století.

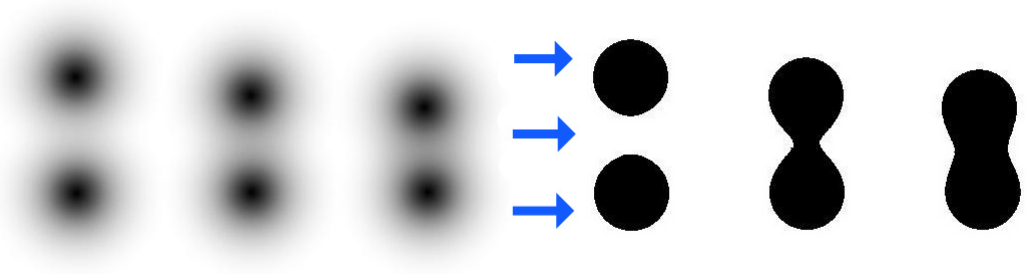
V podstatě je každý objekt metaballs funkce v prostoru která bere jako vstup x y z souřadnice bodu a jejím výstupem je desetinná hodnota. Poté se rozhoduje na základě prahové hodnoty (anglicky *threshold*), jaký bude výstup. Bod v prostoru, jehož výsledek po dosazení do funkce má hodnotu pod tento práh má hodnotu 1 a body které mají hodnotu větší než práh jsou 0. Toto dělí prostor do 2 částí - obsazenou, tedy vyplněnou, a prázdný prostor (většinou vzduch). Spojitý povrch se pak vykresluje tam, kde se dva a více bodů setkávají nebo jsou v určité blízkosti. Tomuto povrchu se říká isosurface. Tento povrch může mít rozdílný vzhled, jenž závisí na použité funkci[4].

Většinou se pro kulaté objekty ve 3D prostoru používá funkce definovaná jako:

$$f(x, y, z) = \frac{1.0}{(x^2 + y^2 + z^2)} \tag{2.5}$$

Funkce má hodnotu 0 v počátku a pro každé další body je jejich hodnota jejich vzdálenost od počátku. Po vybrání pomocí metody prahování je vykreslena perfektní koule. Nicméně pokud teď umístíme 2 body blízko sebe v prostoru, zjistíme, že tímto postupem se vykresluje prostor mezi nimi jak je naznačeno na obrázku 2.5. A pokud tyto 2 body překryjeme, vykreslí se koul, jejíž objem s bude rovnat velikosti součtu objemů obou dvou bodů.

Avšak pokud je ve scéně spousta bodů, je vzorec 2.5 pro vykreslování příliš náročný. Proto se používají i různé jiné funkce.



Obrázek 2.5: Princip techniky metaballs. Mezi 2 texturami částic dochází k vykreslení povrchu mezi nimi, pokud překročí určitou vzdálenost mezi sebou

2.6.2 Marching squares

Dalším kandidátem na vykreslování povrchu kapaliny ve dvoudimenzionálním prostoru je algoritmus Marching squares - metoda pochodujícího čtverce, jenž vychází ze známého algoritmu pro vykreslování Marching Cubes - metoda pochodující kostky. Algoritmus funguje na principu řezání krychle, která se pohybuje v 3D mřížce v prostoru okolo všech částic. Řezáním krychle na základě hustoty rozmístění částic v prostoru se vytváří síť polygonů, která po skončení algoritmu vytvoří povrch okolo každého shluku částic. Analogicky funguje ve dvoudimenzionálním prostoru algoritmus Marching squares. Zde dochází k řezání čtverce v prostoru okolo částic a metoda je tedy trochu méně výpočetně náročnější.

Nevýhodou obou algoritmů je však jejich celková výpočetní náročnost v porovnání s ostatními technikami. Jejich výhodou je ovšem kvalita výsledku. Vetšinou se používají k velmi kvalitnímu vykreslování povrchů, ale pro vykreslování v reálném čase na mobilních zařízeních, kde je k dispozici oproti osobnímu počítači jen omezený výkon, se příliš nehodí.

Kapitola 3

Návrh řešení

Následující kapitola formuluje problémy a nastiňuje návrh jejich řešení. Uvádí také zvolené metody a postupy pro implementaci. Dále je zde popsán návrh řešení této metody, její klady a zápory, možná úskalí a problémy s nimiž je možné se setkat během řešení jednotlivých kroků. Samotná implementace řešení je v následující kapitole.

3.1 Formulace problému

Cílem je realisticky vypadající simulace kapaliny ve 2D řezu uskutečnitelná na hardware mobilních zařízení ideálně alespoň při 60 snímcích za vteřinu. Simulovat se budou pouze viskózní tekutiny jako je voda, sirup atp. Snahou je dosáhnout co možná největšího množství částic ve scéně, což povede k detailnějšímu zobrazení kapaliny. Po fyzikální stránce nemusí být simulace zcela přesná, jelikož interaktivita a výpočetní jednoduchost aplikace bude upřednostněna nad přesností, která pro účely her není vyžadována. Cílem je též dosáhnou uspokojivé realističnosti výsledného zobrazení povrchu kapaliny.

Bude tedy nutné zvolit metodu, která by byla implementovatelná i na mobilních zařízeních a zároveň splňovala všechny výše uvedené požadavky. Jako nejvhodnější metoda se jeví SPH. Všechny její studované implementační algoritmy z jsou navrženy a vyhodnocovány ve 3D prostoru, je však možné je využít bez problémů i pro 2D řez zanedbáním jednoho rozměru. Tedy z těchto algoritmů zvolit tu, která by byla výpočetně jednoduchá a dosahovala dobrých výsledků i na mobilních zařízeních. Některé vzorce použité při implementaci bude vhodné programově optimalizovat, kvůli ušetření prostředků a jejich efektivnímu využívání a zanalyzovat pomocí vhodného nástroje tj. navrhnout metodu nebo postup který zrychlí výpočty - jako například příklad třídící algoritmus tabulky, možnost nemazání celé tabulky po kroku simulace, ale rehashovat tabulku atp. Dále bude nutné zvolit vhodné programové vývojové prostředky pro cílovou platformu a metodu vykreslení povrchu částic jako kapaliny.

3.2 Volba algoritmu

K řešení problému byla zvolena metoda Smoothed particle hydrodynamics popsaná v části 2.4 založená na reprezentaci kapaliny pomocí shluků částic. Volba této metody umožňuje jednoduše a tvárně upravovat vlastnosti kapaliny parametry a změnou koeficientů. Lze ji také teoreticky použít na neomezeně velkou scénu. Poskytuje nejen realistické zobrazení toku ale i hladiny kapaliny.

Na kvalitu simulace bude mít vliv také počet částic, a proto bude třeba dosáhnout jejich co největšího počtu ve scéně za rozumnou cenu výkonu. Při vytváření obsahu scény a náplně hry je však také nutné počítat se zatížením scény výpočty ostatních objektů animací atd. a proto bude počet částic při využití kapaliny ve scéně s mnoha výpočetně složitými objekty ještě menší než při samotné simulaci v prázdné demonstrační scéně.

3.2.1 Popis algoritmu

Jedno z vhodných řešení SPH metody je založeno na algoritmu z článku [2], kterým lze simulovat kapaliny různé viskozity a dále také simulovat viskoelastické látky jako je třeba pryž nebo želé. Tato práce je však soustředěna na simulaci kapalin, a proto zde viskoelastické chování nebude řešeno. Nicméně algoritmus je vymyšlen tak, že lze simulaci viskoelastických látek doimplementovat pouhým přidáním jednoho kroku do hlavní smyčky programu.

Díky své jednoduchosti je tedy k našemu účelu vhodný. Představuje robustní a stabilní metodu výpočtu pro simulaci. Metoda přináší stabilní řešení nestlačitelnosti a předchází problému shlukování částic. Klíčová procedura je nazvána *double density relaxation* a je založena na výpočtu dvou různých hustot v místě částice - jedna při výpočtu zohledňuje počet sousedních částic, a druhá kvantifikuje počet sousedních částic. Poskytuje též možnost hladké zobrazení hladiny. Ta však může být ovlivněna nastavením některých koeficientů částic. Dále přirozeně dosahuje povrchového napětí, jehož sílu lze také koeficienty ovlivňovat. Stabilita řešení zůstává i při velkých časových intervalech mezi snímky. Nicméně po dosažení jisté hodnoty intervalu bude metoda stejně nestabilní, a proto ji lze nasadit jen při jistých hodnotách fps.

3.2.2 Simulační krok

Tato metoda využívá schéma tzv. predikce pozice. Částice je v čase posunována tak, že na začátku simulačního kroku je nejprve posunuta do předpokládané pozice, a poté je v průběhu dílčích kroků algoritmu z této pozice postupně posunována do cílové pozice pomocí získaných impulzů. Ty se postupně aplikují na atributy částice. Na konci simulačního kroku je tak částice umístěna na pozici, kterou získala působením sil sousedů a kolizemi s okolní geometrií.

Části jednoho simulačního kroku simulace jsou nastíněny v algoritmu [**?**]:

algClavetPresny 1: Simulační krok algoritmu

```
1 aplikování gravitace;
2 aplikace viskozity;
3 uložení počáteční pozice každé částice;
4 výpočet plasticity;
5 výpočet elasticity;
6 double density relaxation;
7 vyhodnocení kolizí;
8 aktualizace rychlost všech částic);
```

Nejdříve se upraví rychlost částice aplikováním gravitace a spočtením viskozity z jejich atributů. Následně se uloží jejich počáteční pozice a posunou se na místo v prostoru podle jejich rychlosti. Poté lze aplikovat posuny z kroků pro výpočet viskoelastické chování složené z výpočtů plasticity a elasticity. Poté se aplikuje hlavní výpočet pro dosažení zachování objemu, metoda proti shlukování a povrchového napětí. Nakonec jsou vyhodnoceny kolize

s okolním prostředím a předměty a jsou přepočteny rychlosti částic po celém simulačním kroku.

Implementace bude tedy vycházet z tohoto základního postupu simulace. Místo implementace v trojdimenzionálním prostoru bude upraven do dvou dimenzí. To však nijak vyhodnocování algoritmu neovlivní. Konverze z 3D prostoru na 2D by měla mít pozitivní dopad na náročnost simulace co se týče paměťových a výpočetních prostředků, jelikož odpadnou výpočty třetího rozměru ve všech vzorcích.

Dále je tedy třeba modifikace algoritmu. Bude třeba vypustit viskoelastické chování, protože tento krok výrazně zatěžuje výpočetně systém a není třeba při simulování viskózních kapalin. Jak je vidět, z užití techniky SPH pro simulaci vyplývá implementační problém, a tím je vyhledávání sousedních částic.

3.3 Vyhledávání sousedních částic

Efektivní hledání všech částic v interakčním rádiu h částice je problémem ve všech částicově založených metodách. Pokud bychom porovnávali vzdálenosti naivním způsobem tj. pro každou částici vypočítat vzdálenost od všech ostatních a vybrat jen ty v poloměru h, bylo by to výpočetně velmi náročné. Ani na osobním počítači není pak počet částic uspokojivý, obzvláště pak na mobilních zařízeních. Kvadratická složitost tohoto naivního řešení $O(N)^2$ je z hlediska výkonu nepřípustné. Simulované množství částic ve scéně se díky takovéto složitosti rapidně sníží. Proto je žádoucí snížit počet porovnávaných částic jen na ty nejbližší, u nichž je pravděpodobnost výskytu v interakčním rádiu nejvyšší. Naštěstí existují techniky pro rozdělení prostoru na menší podprostory a vyhledávat v těch, co jsou nejblíže částici v dosahu vyhlazovacího poloměru h. Stejné řešení se často využívají u řešení kolizí objektů ve scéně.

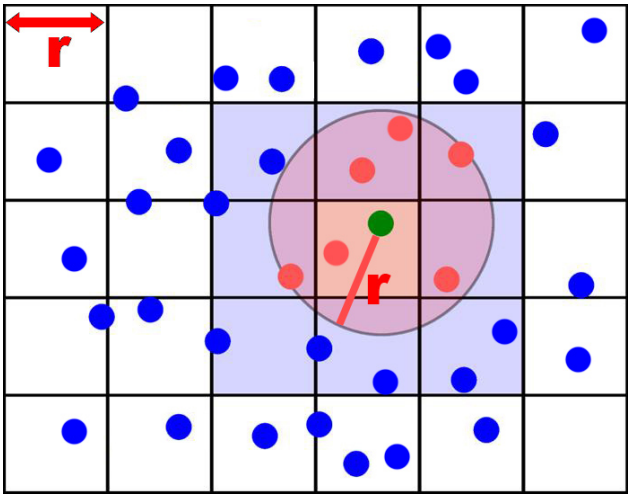
Protože budeme uvažovat rádius h během simulace konstantní a stejný pro každou částici, lze jako řešení tohoto problému využít speciální prostorovou hashovací tabulku tzv. *spatial hash table*, která rozdělí 2D prostor scény na menší podprostory - čtvercové buňky, které budou mít délku hrany shodnou s délkou poloměru interakčního rádiu h.

Prostorová tabulka tedy funguje na principu třídění elementů v prostoru do buněk tabulky na základě pozice elementu v prostoru. Často se využívá také při zjišťování kolizí mezi předměty v prostoru. Nejprve se rozdělí prostor na mřížku čtvercových buněk o hraně čtverce h, a poté se pomocí hashovací funkce zjistí příslušnost objektů do dané buňky. Buňky poté obsahují objekty, které jsou v prostoru blízko u sebe, tedy v jedné buňce.

Vzhledem k tomu, že se částice může vyskytovat i na kraji buňky k zjištění všech sousedních částic v interakčním rádiu h musíme zpracovat i okolních 8 buněk. Dohromady se tedy bude zpracovávat 9 buněk, jak ilustruje obrázek 3.1 protože může dojít k situaci kdy je částice na kraji buňky a je třeba tedy porovnávat tuto částici i s částicemi v sousedních buňkách. Zmenšuje se tak počet porovnání. Hashovací funkce prostorové tabulky bude volena podle zdroje [12]. Funkce bude konvertována na 2D souřadnice jako:

$$hash(x, y) = (\ x\ *\ 73856093\ xor\ y\ *\ 83492791\ )\ mod\ n \tag{3.1}$$

Toto řešení je dostatečně obecné i pro rozsáhlé prostory a nasazení tabulky tedy není závislé na konkrétním umístění kapaliny a částic ve scéně. Vzorec 3.1 také vykazuje málo kolizí v mapování objektů do buněk tabulky v případě nekonečného prostoru.



Obrázek 3.1: Rozdělení prostoru scény na buňky pomocí prostorové hashovací tabulky. Vyhledávání sousedních částic poté probíhá jen v 9 buňkách vyznačených modrou a růžovou barvou

Pokud by však došlo ke kolizi v mapování a částice z rozdílných oblastí scény by byly namapovány do stejné buňky, mohlo by to negativně ovlivnit simulaci. Díky algoritmu SPH však nedojde k ničemu, co by ovlivnilo výsledky simulace, neboť metoda musí stejně počítat vzdálenosti mezi částicemi, a tedy i kdyby byly částice špatně roztříděny, neovlivní negativně výpočty veličin, jen budeme provádět zbytečná porovnání. V případě kolize pouze zatížíme simulaci zbytečnými výpočty, čemuž se chceme z hlediska výkonu vyhnout.

Hlavní nevýhodou prostorové hashovací tabulky je fakt, že se bude muset během každého simulačního kroku tabulka s roztříděnými částicemi vyprázdnit a znovu naplnit roztříděnými částicemi do buněk podle hashovací funkce, protože jak se pozice částice v čase mění, částice cestuje z jedné buňky do druhé. Tuto změnu je proto nutné reflektovat každý simulační krok. Částice tedy bude přesunovat z jedné buňky do jiné a to je nutné zaznamenat a částici správně zařadit. Tyto přesuny mohou být časově náročné a je vhodné najít pro náš případ nějakou optimalizaci.

Předpokládá se, že vyhledávání sousedních částic bude časově nejsložitější část simulačního kroku. Měřením a optimalizacemi implementace se budu zabývat v jiné kapitole. Díky zadání z hlediska řešení v prostoru 2D místo ve 3D přináší také výhodu zjednodušení výpočtu a z toho vyplývající ušetření výpočetních prostředků, jelikož místo prohledávání 27 buněk s možnými sousedy se jich bude prohledávat pouze 9. Také se v hashovací funkci nebudou zpracovávat třírozměrné vektory ale jen dvojrozměrné, což výpočty s vektory urychlí. Na mobilních zařízeních je toto zjednodušení výpočtů velmi vítáno.

3.4 Volba enginu

Aby šla simulace využít na co nejvíce zařízeních s různými operačními systémy a mohla tak oslovit co nejvíce potencionálních hráčů nebo zákazníků, je volba enginu zásadní. Výhodné je při vývoji na mobilní zařízení zvolit multiplatformní herní engine, jež by poskytoval schopnost uniformního vytváření pro více operačních systémů najednou. V takových enginech je možné napsat program v jednom jazyce a bez jakýchkoli větších modifikací vytvářet aplikace na různé druhy zařízení, jak bylo popsáno v sekci 2.5 a následně zvolit cílovou plat-

formu. Poskytují tedy širokou škálu zařízení a možných nasazení simulace.

V této práci kladu důraz na přenositelnost i mezi většinou mobilních platforem. Nakonec bylo vybráno Unity. Bylo zvoleno z důvodu jednoduchosti a předchozích pozitivních zkušeností autora. Při implementaci budu tedy vycházet ze základních znalostí z teoretické části práce.

3.5 Vyhodnocení kolizi

Reakce na kolize s předměty bude stejně jako v práci [2] založena na síle impulzu vypočtené mezi částicí a objektem při kolizi. Impulzy ve formě vektorů jsou založeny na kolizním modelu zvaným *zero-restitution* s mokrým třením.

$$v^n ormal = (\overline{v} \cdot \hat{n})\hat{n} \tag{3.2}$$

$$v^t angent = \overline{v} - \overline{v}^n ormal \tag{3.3}$$

$$I = \overline{v}^n ormal - \mu \overline{v}^t angent \tag{3.4}$$

Výpočet vyžaduje znalost vektoru rychlosti částice $\vec{v_i}$, rychlost objektu v bodu kontaktu s částicí $v_b\vec{o}dy$ a normálového vektoru $\vec{n}$ objektu. Z toho je zjištěna relativní rychlost částice $\overline{v} = \vec{v_i} - v_b\vec{o}dy$. Ta je rozdělena na normálovou a tangentovou složku. Impulz je vypočten podle vztahu 3.4, kde u je třecí parametr povolující nebo zakazující klouzání po povrchu objektu.

Pokud je vlivem velké rychlosti částice během počítání kolize již pronikla částice dovnitř těla objektu, je na konci vyhodnocení impulzu po jeho aplikaci na rychlost částice extrahována z těla objektu a posunuta do pozice těsně před objekt blízko původního místa kolize.

Vzhledem k žádoucí interakci kapaliny s okolními předměty se proto nabízí možnost využití *spatial hash table* i pro uložení pozice geometrie objektů. Vytvořila by se nová tabulka pro objekty, které mohou kolidovat s částicemi. V každé buňce tabulky budou pomocí hashovací funkce uloženy objekty překrývající zcela nebo částečně buňku. Při vyhodnocování kolizí částice s objekty se tak budou porovnávat pouze objekty, které jsou nejblíže částici, tedy v buňce s částicí. Pro každou částici je podle její pozice vyhledána v tabulce buňka s objekty a pro ty jsou vyhodnoceny kolize a spočteny a aplikovány impulzy.

3.6 Vykreslení povrchu kapaliny

Jako metoda vykreslování byla vybrána technika Metaballs z důvod své jednoduchosti. Jelikož se práce nesoustřeďuje na grafické výsledky vykreslení kapalin, metoda bohatě postačuje pro vykreslení kapaliny. Bude tedy implementován shader pracující na tomto principu v jazyce Cg / HLSL a začlenit jej do scény.

3.7 Využití více vláken

Další možností jak zvýšit efektivitu výpočtu a množství částic kapaliny ve scéně je navržení algoritmu simulace s využitím paralelních výpočtů. využití paralelních výpočtů více vláken. Jak je vidět v práci [8] při využiti 4 vláken na 4 jádrovém zařízení se zvýší počet částic ve scéně asi 3x. Nicméně je nutné vybrat anebo navrhnout algoritmus tak, aby se nechoval konkurentně ale paralelně. Při využití více vláken je třeba vyřešit problém jejich synchronizace

při přístupu ke sdíleným prostředkům. K tomu však existují nástroje v každém moderním programovacím jazyce.

Avšak je nutné cílit aplikaci na zařízení s určitým počtem jader. I když to zvyšuje výkon a realističnost simulace a počet použitých částic ve scéně při stejné snímkové frekvenci, limituje to nasazení aplikace pouze na zařízení s těmito parametry. Vícevláknový program na 2 nebo 1 jádrovém smartphonu může za určitých okolností být spíše na škodu. Je nutno si to rozhodnout v době vytváření hry nebo aplikace a zvolit cílenou skupinu zařízení.

Během pokročilých fází práce bylo zjištěno, že aplikační rozhraní Unity není zabezpečeno pro operace více vláken - je *thread non-safe*. Nelze tudíž přistupovat k aplikačním voláním mimo hlavní vlákno, jinak za běhu program vyvolá výjimku. Výsledný kód není také spolehlivý, protože jestli bude volání v danou chvíli fungovat je dílem náhody. Proto urychlení výpočtů pomocí více vláken nemůže být použito. Je to jedno z úskalí volby simulačního enginu. To se negativně odrazí na výkonu programu na vícejádrových zařízeních, kterých je v dnešní době drtivá většina.

Kapitola 4

Implementace

Následující kapitola popisuje praktickou část práce - vytvoření programu na mobilní zařízení, jenž bude simulovat chování kapaliny v prostoru 2D. Při řešení se vychází z poznatků z kapitol 2 a 3. Obsahuje základní popis implementace algoritmu pomocí pseudokódu a rozebírá jednotlivé části implementovaného simulačního kroku.

Vývoj probíhal na platformě Windows v herním enginu Unity v jazyce C#. Řešení využívá aplikačního rozhraní enginu, standardních prostředků jazyka C#. Základem každého objektu je zdědění vlastností a metod z bázové třídy *MonoBehaviour*, jež poskytuje všechny potřebné třídy a prostředky k implementaci v prostředí Unity. Program je řešen objektově-orientovaným paradigmatem. Respektuje uspořádání a návrh virtuální scény při pozdějším používání v aplikacích.

4.1 Unity

Tento herní engine byl zvolen z důvodu jednoduchosti a předchozích pozitivních zkušeností autora. Je základem mnoha herních společností zabývajících se vývojem her převážně na mobilní zařízení a osobní počítače. Jak již bylo zmíněno v teoretickém základu práce, umožňuje vývoj her na více jak 20 platforem mezi které patří: osobní počítače, mobilní zařízení, herní konzole, zařízení s virtuální a rozšířenou realitu a platformy využívající WebGL.

4.2 Architektura programu

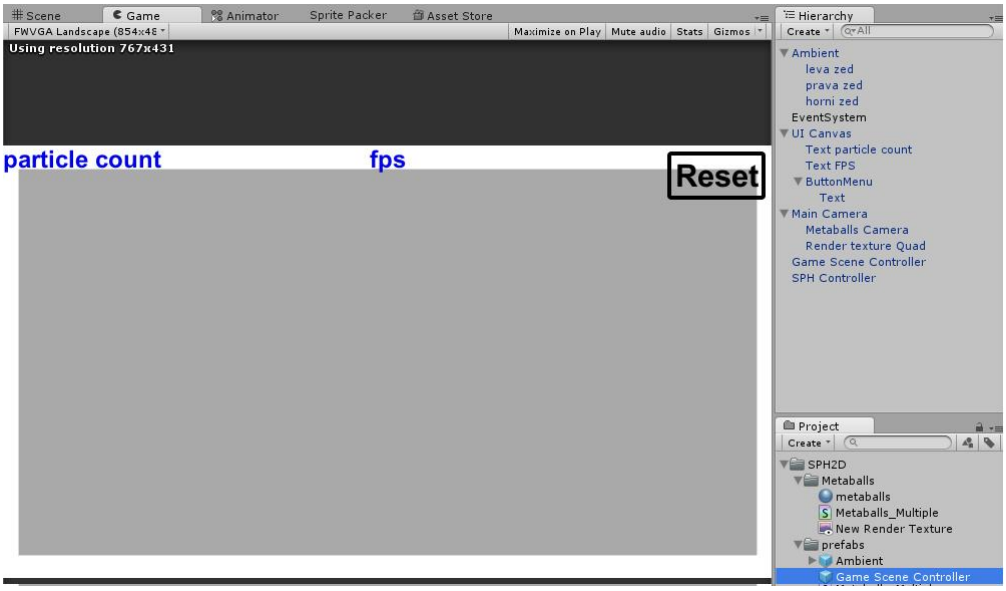
Celý projekt se skládá z 1 scény nazvané *demoScene1*. Ta obsahuje 6 herních objektů, které využívají komponenty enginu, a tvoří logiku a náplň aplikace. Celá simulace je pak implementována pomocí 5 tříd. Jsou to: Fps, Particle, sph, SpatialTable a TouchLogic.

4.2.1 Herní objekty

Všechny jsou uloženy jako tzv. *prefab* - hotové šablony herních objektů s nastavenými komponentami pro použití v dalších scénách či projektech. Nacházejí se ve složce *prefabs* v hlavním adresáři projektu.

SPH Controller

Prvním objektem je *SPH Controller*. Je vytvořen jako prázdný herní objekt. Plní roli kontejneru pro instance třídy *sph*, která implementuje hlavní simulační smyčku algoritmu. Také je

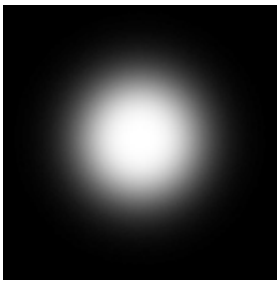


Obrázek 4.1: Scéna s objekty. V části Hierarchy je rodičovská hierarchie herních objektů

v něm instance třídy *TouchLogic*, která obsahuje metody pro zpracování vstupu při dotyku obrazovky.

particle

Druhým herním objektem je samotný objekt částice. Do scény se vkládá až za běhu programu. Obsahuje následující komponenty: Transform, Sprite renderer a instanci třídy *Particle*. Jeho hlavní částí je textura na obrázku 4.2, kterou vykresluje renderer. Její vzhled je dán tím, aby fungovala technika *Metaballs*. Samotná textura je větší než interakční rádius částice h, aby došlo k efektu vykreslení spojení.

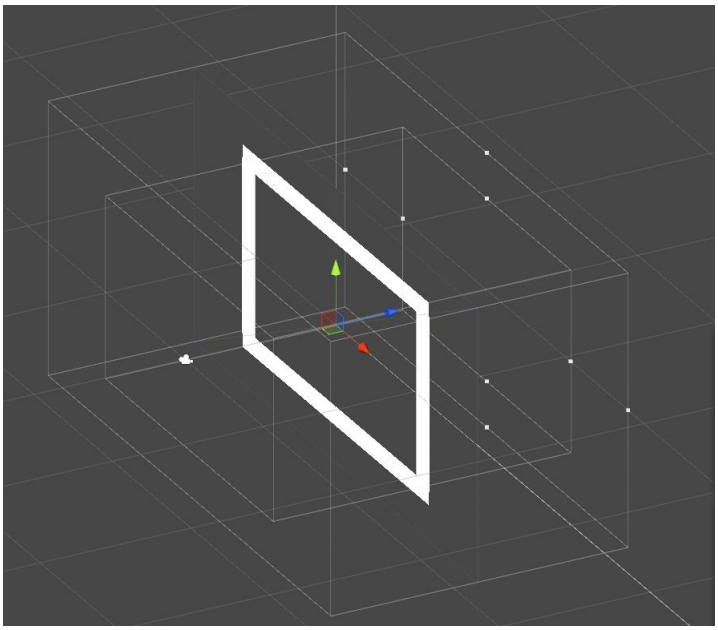


Obrázek 4.2: Textura částice vykreslovaná Sprite rendererem

Main Camera

Třetím je *Main Camera*, což je v podstatě soustava 2 kamer a renderovací textury *Render texture Quad*. Kamery snímají scénu. *Main Camera* je hlavní kamera a snímá vše kromě částic. Ty snímá druhá *Metaballs Camera*, která má nastavenou vlastnost *Culling Mask* komponenty *Camera* na vrstvu s částicemi - vrstvu nazvanou *metaballs*. Poté druhá kamera však svůj výstup neposílá na obrazovku, nýbrž na renderovací texturu uchycenou

na obdélník *quad*. Textura je pak zpracována shaderem implementujícím techniku Metaballs popsané v předchozích kapitolách. Obdélník je přivrácen kolmo ke kamerám jak je patrné na obrázku **??**. Tím je implementováno vykreslování povrchu objemu kapaliny ve 2D v místech, kde se částice pohybují. Kamery jsou ve vztahu rodič-potomek, takže kam budeme přemisťovat tu hlavní, tam půjde i ta na částice i textura. Tak je možné pohybovat s kamerou volně ve scéně díky tomuto obecnému řešení.



Obrázek 4.3: Struktura kamer a Quadu pro renderování textury kapaliny

Game Scene Controller

Čtvrtým je *Game Scene Controller*, ve kterém je komponenta instance skriptu třídy *Fps*, která počítá a zobrazuje počet snímků za vteřinu na obrazovku zařízení.

UI Canvas a EventSytem

Pátým je dvojice objektů *UI Canvas* a *EventSytem*, jenž sdružují prvky uživatelského rozhraní do jednoho objektu a poskytují práci s nimi pomocí enginu. Využívají se pro vstup (dotyk obrazovky), výstup a ovládání aplikace.

Ambient

Poslední je prázdný objekt *Ambient*, sdružující bílé textury uměle vyznačující hranice scény.

4.2.2 Třídy

Particle

Třída slouží pro inicializaci a jako datový kontejner pro potřebné atributy částice. Mezi atributy patří seznam sousedních částic, dále vektory rychlosti, pozice, pozice v předchozím kroku, skaláry pro hustoty a tlaky v okolí částice, koeficient.

Fps

Tato třída implementuje počítadlo snímků za vteřinu, které zobrazuje na obrazovku komponentou *Text* v objektu *Text particle count* pro vykreslování textu. Výpočet probíhá v metodě *Update()*, vykreslování je implementováno coroutinou *vykresleni()*.

SpatialTable

Zde je implementována prostorová hashovací tabulka popsaná v sekci 3.3. Jádrem třídy je atribut *listK* - pole obsahující seznamy buňek (ve zdrojovém kódu nazvané jako kyblíky). Podle hashovací funkce pak lze roztřídit objekty částic dle pozice v prostoru do seznamů v poli. Při instanciaci je třeba zadat konstruktoru velikost buňky a jejich počet. Výpočet hashovací funkce je implementován dle vzorce 3.1.

TouchLogic

Třída implementuje logiku dotykového ovládání zařízení, implementovaná jako stisk levého tlačítka myši v metodě *Update()*. Dále je zde implementováno řízení gravitace akcelerometrem.

sph

Celá hlavní část algoritmu simulace je implementována třídou *sph*. Jsou zde implementovány dílčí části simulační kroku. Zevrubnému popis algoritmu se věnuji v další sekci.

4.3 Algoritmus

Zvolený algoritmus pro implementaci SPH je založen na řešení [2]. Pomocí atributů skriptu v editoru lze ovládat chování nastavováním parametrů.

Algoritmus 2: Simulační krok

```
1 aplikování gravitace();
2 nalezení sousedních částic();
3 aplikace viskozity();
4 uložení předchozí pozice();
5 double density relaxation();
6 vyhodnocení kolizí();
7 aktualizace rychlosti();
8 vyhodnocení vstupů();
```

Algoritmus 2 představuje pseudokód, jež popisuje výpočetní části jednoho simulačního kroku z pohledu nejvyšší úrovně programu. Většina funkcí je konstruována cyklem *for* iterujícím přes seznam všech částic a vykonávající nějaký druh výpočtu.

4.3.1 Aplikování gravitace

V tomto kroku se mění vektor rychlosti částice podle gravitace dle vzorce 4.1, kde v_i je rychlost částice, Δt je časový interval mezi snímky a g je gravitace ve scéně.

$$v_i = v_i + \Delta t \cdot g \tag{4.1}$$

Gravitace je získávána z atributu statické třídy *Physics2D.gravity*. Rychlost částice se modifikuje na základě výpočtu pomocí hodnoty vektoru. Vektor gravitace je součástí nastavení 2D scény editoru. Hodnotu gravitačního vektoru lze během simulace měnit. Tím bylo docíleno zasazení do globálního nastavení scény. Tak může kapalina přirozeně reagovat na změnu gravitace společně s ostatními objekty ve scéně.

4.3.2 Nalezení sousedních částic

V této části jsou pro každou částici nalezeny sousední částice v okruhu h. Ve speciálním konstruktoru třídy *sph* je vytvořena a inicializována hashovací tabulka. Ta je implementována třídou *SpatialTable*. Prostor scény je tedy rozdělen na buňky.

Poté se shromáždí částice z buňky ve které je právě zpracovávaná částice a 8 okolních buněk, které obklopují tuto buňku. Poté jsou pomocí vektorů pozic vypočteny vzdálenosti mezi částicí a jejími možnými sousedními. Pokud soused leží v interakčním rádiu h, jsou přidány do seznamu obsahující sousední částice.

4.3.3 aplikace viskozity

V tomto bodě se aplikuje působení viskozity na částici. Aplikuje se jako vnitřní radiální impulz mezi páry částic. Pro každou částici je spočten impulz síly I. Impulz, jenž má formu dvourozměrného vektoru je počítán ze sousedů, je závislý na vzdálenosti obou částic, je získán lineárním kernelem ze vzorce 4.2. Impulz pak modifikuje rychlost obou částic.

$$(1 - r_{ij}/h \tag{4.2}$$

4.3.4 Uložení předchozí pozice

Zde se uloží stará pozice částice do atributu třídy *Particle*. Poté proběhne posun do předpokládané pozice vypočtené podle intervalu mezi snímky *deltatime* a její rychlosti. Neposouvá se však přímo přiřazením do komponenty *Transform*, nýbrž je pouze modifikována atribut *pos* typu vektor. Testováním bylo zjištěno, že přímý posun přiřazením do komponenty je výpočetně náročnější než modifikace atributu, a proto byl pro tyto účely vytvořen.

4.3.5 Double density relaxation

Klíčový krok který je srdcem celé simulace. Funkce výpočte pseudotlak a pseudohustotu zohledněním sousedních částic podle vzorce 4.3. Nejdříve je pro každou částici vypočtena hustota a blízká hustota. Poté je z obou hustot vypočten tlak a blízký tlak. Nakonec je z tlaků vypočten vektor D, který se použije ke změně pozice sousední částice. Všechny D vektory jsou nakumulovány do pomocného vektoru dx a na závěr je částice posunuta na pozice přičtením vektoru dx k vektoru pozice. Také se ihned provede posun částice přiřazením do komponenty.

$$D_{ij} = \Delta t^2(P_i(1 - r_{ij}/h) + P_i^{near}())(1 - r_{ij}/h)^2)\hat{r}_{ij} \tag{4.3}$$

4.3.6 Vyhodnocení kolizí

Hranice scény jsou pevně dány souřadnicemi pro x <-10,10> a y <-5,5>. Kolize jsou z pohledu sil vyhodnocovány tak, jak je popsáno v sekci 3.5, ale kolize a interakce s předměty nebyly implementovány.

4.3.7 Aktualizace rychlosti

Na závěr je spočtena nová počáteční rychlost do dalšího simulačního kroku. Spočte se odečtením vektoru původní pozice částice od vektoru současné pozice a podělením časem, který uplynul mezi dvěma snímky.

4.3.8 Vyhodnocení vstupů od uživatele

v této metodě je implementováno vymazání scény testováním statického instanční proměnné typu *input*. Jelikož vstup je asynchronní událost a tak se musí snímat zde, aby nedošlo k vyvolání výjimek a negativnímu ovlivnění simulace. Natavením atributu se zajišťuje bezpečné odstranění objektů částic ze scény.

4.4 Implementace ovládání gyroskopem

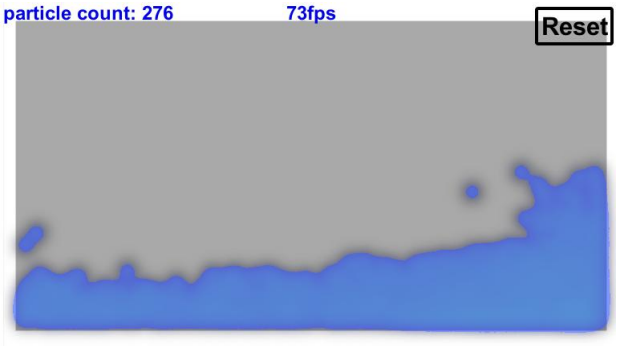
Je implementována ve třídě *TouchLogic* jako modifikace gravitace ve scéně pomocí atributu*Physics2D.gravity*. Funguje tak, že se načte vstupní vektor z akcelerometru zařízení, ten se normalizuje a poté se vynásobí hodnotou 9.81f (přibližná hodnota gravitace nastavená pro naši polohu). Lze také zapínat a vypínat řízení kapaliny akcelerometrem pomocí nastavení boolean hodnoty atributu *AccelerGravity* této třídy.

Kapitola 5

Výsledky a vyhodnocení

5.1 Výsledná aplikace

Výsledek implementace je vidět na obrázku **??**.



Obrázek 5.1: Vzhled UI výsledné aplikace

V pravém horním rohu obrazovky je tlačítko *RESET*, které maže přítomné částice ze scény. V levém horním rohu scény je údaj hlásící počet částic ve scéně. A uprostřed nahoře se nachází měřič fps. Vlastnosti kapaliny lze ovládat změnou údajů v Object Inspektoru v editoru Unity.

5.1.1 Profilování aplikace na PC

Pro usnadnění práce dalo se to dělat pouze na PC, protože detailní profiling aplikace vyžaduje mnohem výkonnější hardware než mobilní zařízení, jelikož zatěžuje simulaci kvůli zasílání zpráv mezi objekty. Počet částic tak spadl z cca 1300 částic na 50.

Cílem bylo zjistit náročnost jednotlivých kroků či operací ve výpočtech simulace. Pomocí Profileru jsem zjistil slabé body a ty se snažil zefektivnit. Optimalizace a testy implementace probíhaly v testovací scéně jménem *demoScene1*, jež je součástí aplikace a instalačního balíčku v příloze.

Profilováním aplikace bylo odhaleno několik překvapivých slabých míst. Zajímavým problémem se v Unity stal cyklus *foreach*, jehož nasazení zatěžovalo aplikaci mnohem více než použití cyklu *for*. Důvodem byly chybné alokace iterátoru iterování přes seznam. Bylo zjištěno, že chyba byla na straně tvůrců enginu Unity, jež vytvářeli knihovny. Byl tedy nasazen cyklus *for*.

Dále byla použita optimalizace prostorové hashovací tabulky. Na začátku simulačního kroku je nutné roztřídit částice do příslušných buněk, jež jsou implementovány jako lineární seznamy. Mazání a opětovnéí řidávání částic do seznamu je však časově náročná operace, a tak byla vymyšlena optimalizace, kdy se v tabulce přesouvají pouze částice, které se v předchozím kroku dostali do jiné buňky. Každá částice si pamatuje v atributu hodnotu buňky, v níž se nachází a tu pak porovná s nově vypočteným hashem. Pokud jsou shodné částice zůstává v seznamu a pokud nejsou klíče stejné, částice se přesune.

Osvědčilo se také využívání využívání pomocných proměnných místo přístupu a práci s API Unity, jež je samozřejmě mnohem pomalejší při přístupu než běžné proměnné. Také bylo zjištěno, že při naivním řešení vyhledávání bylo možné simulovat asi 80 částic ve scéně. Po implementaci pomocí hashovací tabulky, jež je popsána v kapitole 3, bylo dosaženo řádově lepších výsledků.

——Analýzou pomocí vestavěného nástroje *Profiler* bylo také zjištěno, že 50-60% výpočetního času tráví program v části algoritmu pro vyhledávání sousedů, ... Celkově bylo optimalizacemi ve scéně na PC získáno asi o 300 částic více.

5.1.2 Testování aplikace na Smartphonech

Cílem testů bylo posoudit kvalitu simulace, zjistit náročnost aplikace na hardwarový výkon. Další cíl je zjistit hratelné množství částic na průměrném smartphonu. Posuzováno bylo podle snímkové frekvence aplikace. Hodnocení kvality vykreslení povrchu kapaliny bylo posuzované subjektivně.

Testy byly prováděny na 5 zařízeních s operačním systémem Android a to:

- Sony Xperia M s 2 jádrovým procesorem o taktu 1 Ghz, 1 GB RAM, operační systém Android verze 4.3.
- Samsung GALAXY S4 mini se 4 jádrovým procesorem o taktu 1 Ghz, 1 GB RAM, operační systém Android verze 4.4.
- Huawei Ascend G6-L11 se 4 jádrovým procesorem Quad-core o taktem 1,2 Ghz, 1,0 GB paměti RAM, operační systém Android verze 4.3.
- Samsung GALAXY S3 Neo se 4 jádrovým procesorem o taktu 1,4 Ghz, 1,5 GB RAM, operační systém Android verze 4.4.
- Samsung GALAXY A3 se 4 jádrovým procesorem o taktu 1,5 Ghz, 1,5 GB RAM, operační systém Android verze 6.0.1

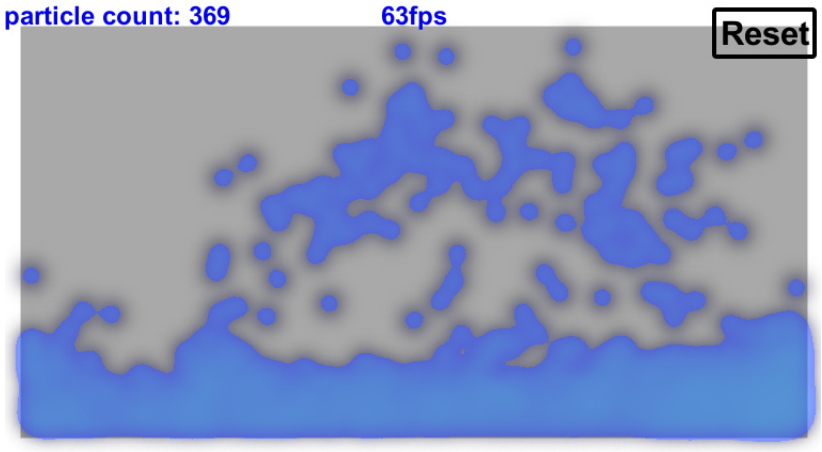
Všechna zařízení patří do kategorie střední až nižší třídy, pro účely testů to však postačuje. Výkonnější zařízení pochopitelně budou vykazovat lepší výsledky a budou schopny simulovat kapalinu detailněji větším počtem částic. Při vývoji aplikací je cílem oslovit co nejširší množství potencionálních uživatelů.

Jak ukazuje tabulka 5.1, na starších zařízeních je dosaženo při 60 snímcích za vteřinu jen málo částic. Na výkonnějších zařízeních, jako je poslední, je situace mnohem lepší a program je použitelnější. Pokud budeme počítat zařazení do hry nebo aplikace, je tedy potřeba tuto dimenzovat pro rychlejší zařízení.

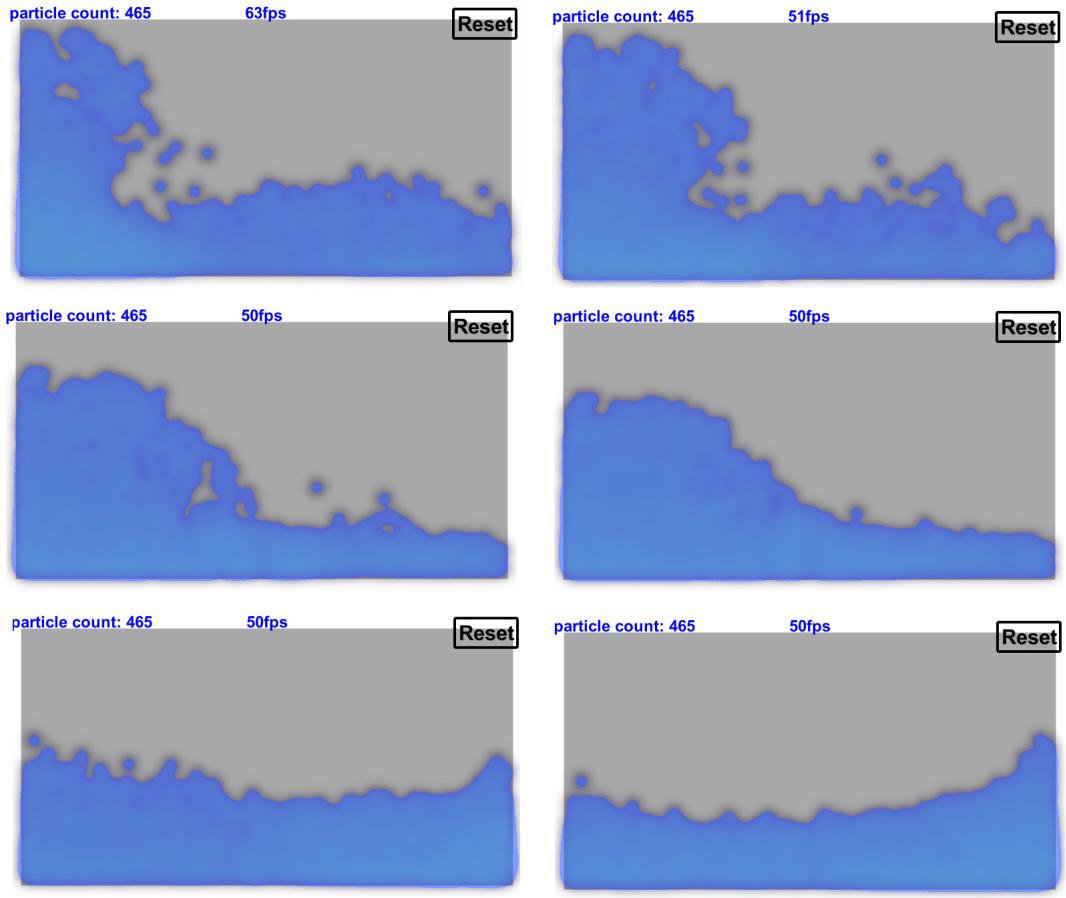
Po grafické stránce tedy není simulace vůbec náročná. V aplikaci je prostor pro rozvoj grafické stránky hry, ale není z hlediska zatížení a využití CPU. Drtivou většinu zdrojů využívá výpočet hlavní smyčky simulace. Při zvoleném měřítku částic jejich množství vyplní sotva polovinu displeje. Situace je samozřejmě lepší na výkonnějších zařízeních.

zařízení/fps	60	45	30
Xperia	90	210	320
Samsung S4	120	220	420
Huawei	140	230	480
Samsung S3	70	100	360
Samsung A3	300	400	600

Tabulka 5.1: Závislost počtu částic na počtu snímků za vteřinu jednotlivých zařízení



Obrázek 5.2: Výsledek implementace při běhu na PC.



Obrázek 5.3: Šíření vlny scénou

Kapitola 6

Závěr

Byl implementován program pro platformu Android simulující chování kapaliny. Výsledný program byl otestován na mobilních zařízeních s různými hardwarovými parametry. V aplikaci je dosaženo asi 300 částic při 60 fps na průměrném zařízení, což je uspokojivé množství pro účely jednoduchých her. Simulace vyžaduje mnohem výkonnější hardware, než bylo původně předpokládáno. Pokud by měla být simulace nasazena ve hrách a interaktivních aplikacích, pro bezproblémový chod aplikace budou tyto vyvíjeny spíše na výkonnější modely.

Při implementaci nebylo kvůli zvolenému hernímu enginu využito paralelizace výpočtů pomocí vláken ani nebylo vytvořeno viskoelastické chování, jež lze ale jednoduše získat vložením dalšího výpočtu do posloupnosti kroků simulace.

Testy simulace ukázaly, že z po grafické stránce a z hlediska vykreslování není simulace vůbec náročná díky zvolené metodě vůbec náročná, avšak není moc graficky kvalitní. Většinu výpočetního času a výkonu zabere CPU jednotce výpočet simulačního algoritmu.

Návaznost na práci by mohla být v podobě implementace viskoelastického chování. Dalším rozšířením práce by se mohla stát implementace kvalitnějšího vykreslování kapaliny pomocí jiných metod.

Literatura

[1] Bridson, R.: *Fluid Simulation for Computer Graphics*. A K Peters, 2008, ISBN 978-1-56881-326-4.

[2] Clavet, S.; Beaudoin, P.; Poulin, P.: Particle-based Viscoelastic Fluid Simulation. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM SIGGRAPH, 2005, s. 219–228.

[3] Dickheiser, M.: *Game Programming Gems 6*. Charles River Media, 2006, ISBN 1-58450-450-1.

[4] Geiss, R.: Metaballs. [Online]. [vid. 2017-4-15].
URL http://www.geisswerks.com/ryan/BLOBS/blobs.html

[5] Gourlay, M. J.: Fluid Simulation for Video Games(part 1). [Online]. Poslední modifikace: 30-4-2012. [vid. 2015-12-28].
URL https://software.intel.com/en-us/articles/fluid-simulation-for-video-games-part-1/

[6] Kincl, J.: Newtonská tekutina. [Online]. [vid. 2017-4-28].
URL https://cs.wikipedia.org/wiki/Newtonsk%C3%A1_tekutina

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

[8] Mansson, D.: *Interactive 2D Particle-based Fluid Simulation for Mobile Devices*. bakalářská práce, KTH Royal Institute of Technology, Stockholm, Švédsko, 2013.

[9] Oldřich Lepil, R. H., Milan Bednařík: Fyzika I pro střední školy. 2007.

[10] Technologies, U.: Unity - Game Engine. 2017, [Online]. [vid. 2017-5-10].
URL https://unity3d.com/unity

[11] Technologies, U.: Unity User Manual. 2017, [Online], [vid. 2017-05-10], [rev. 2017-05-05].
URL https://docs.unity3d.com/Manual/index.html

[12] Teschner, M.; Heidelberger, B.; Mueller, M.; aj.: Optimized Spatial Hashing for Collision Detection of Deformable Objects. In *Vision, Modeling, and Visualization*, 2003, s. 47–54, [Online]. Poslední modifikace: 30-4-2012. [vid. 2016-5-10].
URL
http://matthias-mueller-fischer.ch/publications/tetraederCollision.pdf

Přílohy

Seznam příloh

Příloha A

Obsah CD

A.1 Technická zpráva

Technická zpráva ve formátu pdf se nachází v kořenovém adresáři CD. Zdrojové soubory pro LaTeXa soubor Makefile se nacházejí v adresáři `bp`. Po případných změnách lze technickou zprávu ve formátu pdf vygenerovat pomocí příkazu `make`.

A.2 Balíček pro import do enginu Unity

Balíček s názvem `sph2d.unitypackage` pro import do projektu v Unity se nachází v kořenovém adresáři CD.

A.3 Instalační soubor sph2d.apk

Instalační soubor demonstrační aplikace pro operační systém Android se nachází v kořenovém adresáři CD.

A.4 Zdrojové kódy

Všechny zdrojové kódy se nachází v adresáři `src`.