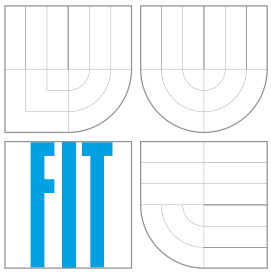
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

INTERAKTIVNÍ SIMULACE CHOVÁNÍ TKANINY AKCELEROVANÁ POMOCÍ GPU

INTERACTIVE CLOTH SIMULATION ACCELERATED BY GPU

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. VOJTĚCH MELICHAR

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JIŘÍ JAROŠ, Ph.D.

BRNO 2016

Zadání diplomové práce/18762/2015/xmelic15

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2015/2016

Zadání diplomové práce

Řešitel: **Melichar Vojtěch, Bc.**
Obor: Počítačová grafika a multimédia
Téma: **Interaktivní simulace chování tkaniny akcelerovaná pomocí GPU**
Interactive Cloth Simulation Accelerated by GPU
Kategorie: Počítačová grafika

Pokyny:

1. Seznamte se s architekturou moderních grafických karet.
2. Osvojte si programovací jazyk CUDA/OpenCL a možnosti jeho spolupráce s grafickou knihovou OpenGL.
3. Prostudujte fyzikální modely vhodné pro realistické simulace tkanin.
4. Navrhněte vhodnou scénu s několika objekty a kusem tkaniny, jejíž chování budete simulovat.
5. Navrženou koncepci realizujte tak, že fyzikální model i vizualizaci bude počítat grafická karta s minimem zásahů ze strany procesoru.
6. Otestujte efektivitu a výkonnost implementace fyzikální simulace chování tkaniny.
7. Zhodnoťte kvalitu vizualizace a dopad na výkonnost.

Literatura:

- Dle pokynů vedoucího.

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

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% 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: **Jaroš Jiří, Ing., Ph.D.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 25. května 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

L.S.

Kotásek

doc. Ing. Zdeněk Kotásek, CSc.
vedoucí ústavu

Abstrakt

Tato práce se zabývá interaktivní simulací chování tkanin s využitím GPU pro obecné výpočty. V první části jsou rozebrány všechny technologie, které jsou následně využity při implementaci programu. Druhá část poté diskutuje různé způsoby řešení simulací. Především se věnuje částicovým systémům, které patří k nejpoužívanějším metodám. Následně je navržen program, který je v rámci této práce také implementován. Implementace proběhla ve čtyřech různých variantách. První variantou je čistě CPU implementace, druhou variantou je optimalizace CPU implementace pomocí technologie OpenMP. Z těchto implementací vychází CUDA implemntace. Poslední zde implementovanou variantou byla optimalizovaná CUDA implementace. Na závěr práce jsou všechny implementace vyhodnoceny z pohledu jejich výpočetní složitosti a vhodnosti pro použití v grafice počítané v reálném čase.

Abstract

This master thesis deals with interactive cloth simulation accelerated by GPU. In the first part there is a description of all technologies used during implementation of a program. The second part discusses various simulation methods. It is mainly focused on particle systems as a most used method. These parts are followed by a design of the program, which is implemented as a part of this thesis. The program was implemented in four variants. The first variant is CPU implementation, which was then optimalized with OpenMP. CUDA implementation is based on these implementations. Last variant implemented in this thesis is optimized CUDA implementation. All these implementations are evaluated from compute complexity point of view and suitability for real time graphics.

Klíčová slova

Simulace tkanin, CUDA, OpenGL, OpenMP, Qt, GPU

Keywords

Cloth simulation, CUDA, OpenGL, OpenMP, Qt, GPU

Citace

MELICHAR, Vojtěch. *Interaktivní simulace chování tkaniny akcelerovaná pomocí GPU.* Brno, 2016. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Jaroš Jiří.

Interaktivní simulace chování tkaniny akcelerovaná pomocí GPU

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jiřího Jaroše, Ph.D. Další informace mi poskytl pan Ing. Tomáš Milet. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Vojtěch Melichar
24. května 2016

Poděkování

Děkuji panu Ing. Jiřímu Jarošovi, Ph.D. za odborné rady a cenné připomínky, kterými přispěl k vypracování této diplomové práce. Dále bych chtěl poděkovat panu Ing. Tomáši Miletovi za jeho externí konzultace.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Význam počítačové grafiky se v posledních několika letech výrazně zvýšil. Počítačová grafika nás obklopuje v každodenním životě a ani si to v mnoha případech neuvědomujeme. Příklady jejího využití můžeme nalézt v různých komerčních vizualizacích produktů, v interaktivních výukových a výstavních pomůckách, v počítačových hrách, ve filmech. Vzhledem k nynějšímu výraznému růstu využívání počítačové grafiky roste souběžně i poptávka po stále dokonalejších vizuálních výstupech.

Simulování většiny jevů, které je nutné vzít v potaz pro kvalitní vizuální výsledek, je již dlouhou dobu dobře prostudovaná oblast. Jediné, co v minulých letech bránilo dosažení dokonalých výsledků, byla velká výpočetní náročnost některých problémů. S rostoucí poptávkou po kvalitních vizuálních výstupech roste i trh pro výrobce grafických karet, které mají za úkol vykreslování počítačové grafiky. Díky tomuto faktu se jednotliví výrobci předhánějí v technické pokročilosti svých produktů, a tak se postupně odstraňují dřívější výpočetní omezení.

Další výzvou pro počítačovou grafiku je přenesení některých fenoménů počítačové grafiky i do oblasti grafiky počítané v reálném čase. Počítačovou grafiku počítanou v reálném čase můžeme nalézt například v počítačových hrách nebo v překotně se vyvíjející virtuální realitě. Výzvou pro počítačovou grafiku v těchto hrách je právě nedostatek času, který je k dispozici na vykreslení jednoho snímku. Na rozdíl od filmů, kde jsou na vykreslení jednoho snímku dostupné hodiny, jsou ve hrách k dispozici pouze desítky milisekund.

Jedním z takových fenoménů, který je sice dobře prostudován, ale je výpočetně náročný, je simulace chování tkanin. Simulace chování tkanin není sama o sobě náročná, ale vyžaduje provedení relativně jednoduchých výpočtů nad velkou množinou dat. A kvalita výstupu je závislá na datové náročnosti. Čím více dat je použito pro simulaci, tím lepší je výsledek. Tento fakt znesnadňoval použití simulace chování tkanin v reálném čase. Vzhledem k technologickému pokroku je však nyní možné i takovou simulaci počítat v reálném čase. Tímto technologickým pokrokem je myšleno využití grafických karet pro obecné výpočty. Grafické karty jsou z podstaty svého účelu vysoce paralelní zařízení. Obsahují totiž velké množství relativně jednoduchých výpočetních jader, která umožňují zpracování velkého množství dat souběžně. Právě taková výpočetní podpora je nutná pro realizaci výpočtu problému, jako je simulace chování tkanin.

V rámci této práce jsou probrány nástroje, které umožňují programování grafických karet. Jedná se jak o nástroje čistě pro vykreslování obrazu (např. OpenGL nebo DirectX), tak i o nástroje pro obecné výpočty na grafické kartě (technologie CUDA).

První část se stručně zabývá současností i historií vývoje grafických karet, která vedla až k použití grafických karet pro obecné výpočty. Dále jsou v této části probrány technologie

OpenGL a CUDA. Součástí popisu technologie CUDA je i popis architektury grafických procesorů Kepler. Součástí první části je také stručný popis technologie OpenMP, která je využita v průběhu implementace pro urychlení simulace na CPU. Poslední zde diskutovanou technologií je framework Qt, který je použitý pro vytvoření grafického uživatelského rozhraní.

Druhá část obsahuje popis problému řešeného v této práci, kterým je interaktivní simulace chování tkanin. Zabývá se hlavně nejpoužívanější metodou simulace tkanin, kterou jsou částicové systémy, jejich popisem a způsobem řešení. Také obsahuje návrh uživatelského rozhraní programu, které je implementováno v rámci práce.

V třetí části je provedena implementace simulace chování tkanin. Simulace je nejprve implementována tak, že její výpočet je celkově proveden na CPU. Následně je běh tohoto kódu optimalizován pomocí technologie OpenMP umožňující paralelní zpracování dat na CPU. Optimalizace kódu pomocí technologie OpenMP usnadní následný přepis do technologie CUDA. Po optimalizaci pomocí OpenMP následuje přepis do technologie CUDA a optimalizace tohoto kódu. Část společná všem těmto implementacím je část vykreslování samotného výsledku simulace pomocí OpenGL.

Závěr práce obsahuje všechny implementace vyhodnocené z pohledu rychlosti výpočtu. Jsou zde také vzájemně porovnány. Předpokládá se, že čistě CPU implementace je nejpomalejší. Rychlejší je implementace optimalizovaná pomocí OpenMP. Výrazně rychlejší než OpenMP implementace je kód přímo přepsaný do technologie CUDA. Následuje optimalizovaná verze CUDA implementace. Předpokladem je, že je o málo rychlejší než CUDA verze bez jakékoliv optimalizace.

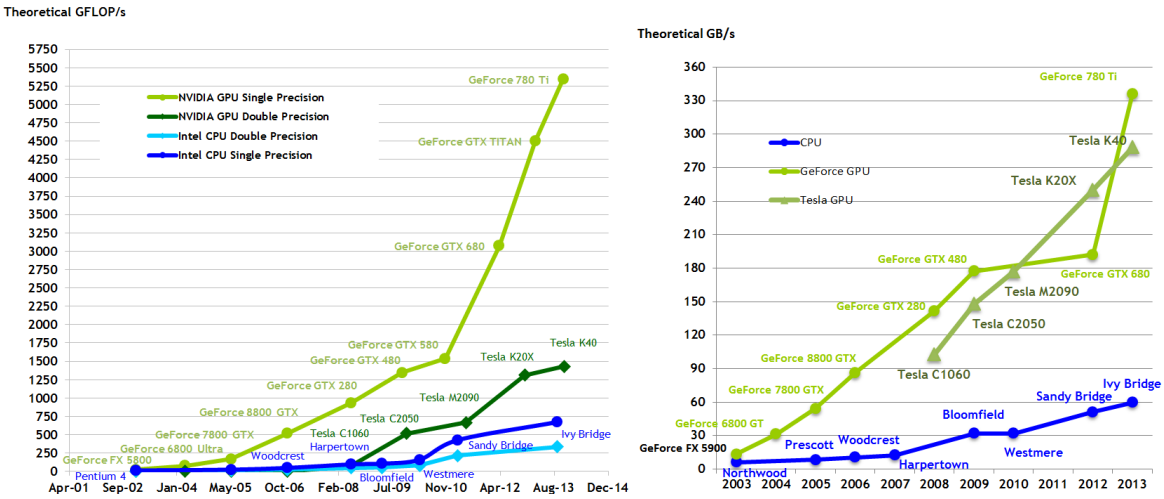
1.1 Cíle práce

Tato práce se zabývá simulací chování tkaniny počítané v reálném čase a jejími výpočetními možnostmi. V průběhu implementace je vytvořeno několik variant simulačního jádra, jmenovitě čistě CPU implementace, CPU implementace optimalizovaná pomocí OpenMP, CUDA implementace a optimalizovaná CUDA implementace. Cílem této práce je vytvořit program, který prezentuje možnosti simulace tkanin. Současně jsou také porovnány jednotlivé výše zmíněné implementace z hlediska jejich výkonu. Za tímto účelem je provedena série měření na dvou testovacích počítačích. Měření probíhala na stolním počítači s dedikovanou grafickou kartou a následně byly stejné testy spuštěny na notebooku s dedikovanou mobilní grafickou kartou.

Kapitola 2

Grafické karty a jejich API

Grafické karty jsou prvotně určeny pro akceleraci grafických úloh, jakými jsou vykreslování 2D a 3D grafiky. Zpracování grafických dat je ze své podstaty paralelní problém. Jedná se totiž o zpracování samostatných primitiv, u kterých nezáleží na pořadí jejich zpracování a ani se vzájemně nijak neovlivňují. Grafické karty a především jejich procesory, zvané GPU (Graphics Processing Unit), jsou pro tyto účely konstruovány. Grafické procesory tedy obsahují velké množství paralelních jednotek, které mohou zpracovávat velké množství dat v jeden okamžik. Díky této architektuře jsou moderní grafické procesory výkonnější než současné CPU [13]. Tento trend je vyobrazen na obrázku 2.1.



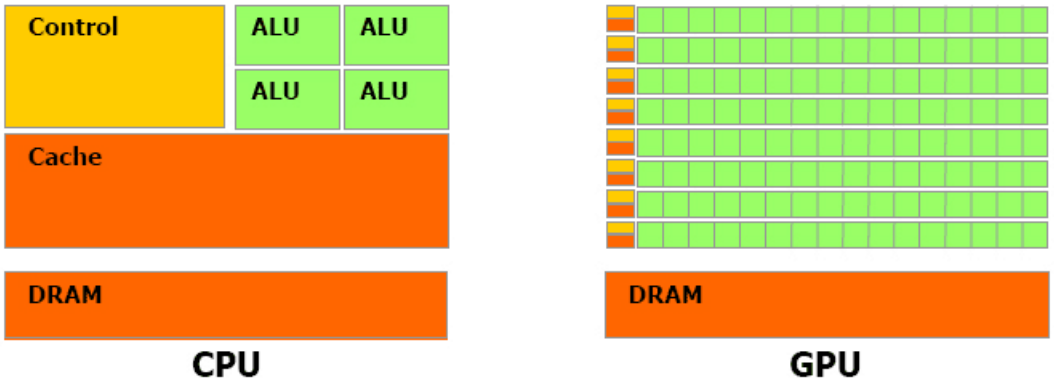
Obrázek 2.1: Srovnání výkonu GPU a CPU [5].

Hlavním důvodem takto velkého rozdílu ve výkonu GPU a CPU je to, že většina plochy CPU je určena pro různé vyrovnávací paměti. Výpočetní jednotky v GPU nejsou tak komplexní jako na CPU, ale na ploše čipu GPU je jich velké množství. Díky tomu je grafický procesor schopen dosahovat mnohem vyšší výpočetní rychlosti. Rozmístění jednotlivých jednotek na ploše čipu GPU a CPU je na obrázku 2.2.

Moderní GPU lze tedy využít pro obecné výpočty, které jsou ze své podstaty paralelní. Pro využívání moderních grafických procesorů pro obecné výpočty se vžil termín GPGPU (General Purpose Graphics Processing Unit), neboli grafický procesor pro obecné výpočty.

Grafické procesory ale nebylo vždy možné využívat pro obecné výpočty. První grafické procesory byly navrženy pouze pro grafické operace. Z tohoto důvodu nešly žádným

způsobem programovat. Situace se změnila s příchodem programovatelného vykreslovacího řetězce. Ten umožňoval některé obecné výpočty grafickému procesoru podsunout formou normální grafické úlohy. Toto řešení umožňovalo obecné výpočty na grafické kartě. Ale pro programování grafické karty musel mít uživatel znalosti nějakého grafického API. O GP-GPU se dá hovořit až po nástupu rozhraní pro obecné výpočty. Těmi jsou například CUDA od společnosti Nvidia nebo OpenCL, o jehož rozvoj se stará skupina Khronos group.



Obrázek 2.2: Rozmístění jednotek na čipu GPU a CPU [13].

2.1 Historie grafických karet

Počátky počítačové grafiky je možné vysledovat až k počátkům výpočetní techniky [19]. Vývoj specializovaného hardwaru pro grafické účely začal v první polovině šedesátých let minulého století. Důvodem pro jeho vývoj bylo využití počítačů pro vědecké výpočty a nutnost prezentovat výsledky těchto výpočtů ve srozumitelné podobě. Například v podobě grafů. Další vývoj počítačové grafiky je spojen s počítačovými hrami, které znamenaly velký trh pro výrobce hardwaru. V začátcích vývoje grafického hardwaru nebyly produkty jednotlivých výrobců mezi sebou navzájem kompatibilní. Každý výrobce měl vlastní řešení grafického hardwaru.

Velký rozmach zažívá počítačová grafika s příchodem osmibitových domácích počítačů a herních automatů. Možnosti těchto strojů byly samozřejmě omezené, ale i přesto dovolovaly vytvářet pokročilé efekty. Zpracování grafických dat v dobách osmibitových počítačů zajišťovaly specializované čipy. V mnoha případech měly na starost také ještě další periferie. Mezi první grafické čipy se řadí čipy ANTIC a GTIA použité v počítačích Atari nebo čip VIC-II ze slavného počítače Commodore C64. Tyto a jim podobné grafické čipy většinou podporovaly dva typy režimů. Jednalo se o textový a grafický režim. Různých grafických čipů vzniklo velké množství. Navzájem se lišily velikostí paměti a podporovanými režimy.

Za první grafické karty lze považovat karty pro počítač IBM PC [21]. Jedná se o kartu MDA pro zobrazování textu a kartu CGA, která umožňovala zobrazovat i grafiku. IBM PC byl první sériově vyráběný počítač umožňující přidávat různé rozšiřující karty. Díky této vlastnosti a faktu, že byl komerčně velice úspěšný, vznikl zcela nový trh s příslušenstvím pro počítače. Mezi takové příslušenství patřily samozřejmě i grafické karty. Tento zcela nový trh byl do několika let obsazen řadou firem, které prodávaly více či méně podařené grafické karty. V průběhu následujících let prošly grafické karty překotným vývojem. Objevila se řada standardů jako například VGA, SVGA a další. S příchodem standardu PCI dochází

k jistému pročištění trhu, protože některé společnosti nebyly schopny novým grafickým kartám konkurovat.

Rok 1996 znamenal příchod 3D grafiky i do osobních počítačů. První verze OpenGL sice vyšla již o 4 roky dříve, ale v té době byla 3D grafika spíše doménou profesionálních a drahých počítačů. Za velkým rozšířením 3D grafiky stojí opět herní průmysl. Jedním z významných nedostatků grafických karet dostupných v té době byla absence široce uznávaného standardu. Každý výrobce měl své vlastní programovací rozhraní. S vydáním DirectX 3 se tato situace začala o něco zlepšovat. Významným hráčem té doby byla společnost 3Dfx a její čipy Voodoo.

Přelomovou kartou z pohledu 3D grafiky byla karta GeForce 256 od společnosti Nvidia, kde byl výpočet transformací a osvětlení přenesen přímo na grafickou kartu [17]. Z pohledu využití grafických karet pro obecné výpočty je přelomový rok 2001 a vydání série GeForce 3. Série GeForce 3 implementovala tehdy nový standart DirectX 8.0. Ten vyžadoval, aby grafická karta obsahovala programovatelné vertex a pixel shadery. Od té doby bylo možné využívat grafické karty pro obecné výpočty, i když toto použití mělo jisté nevýhody.

Další malou revoluci pro obecné výpočty na grafické kartě znamenal rok 2006. Byla totiž představena karta GeForce 8800 GTX. Grafický čip použitý na této kartě byl prvním čipem postaveným na nové technologii CUDA od společnosti Nvidia. Technologie CUDA umožnila plné využití potenciálu grafických karet pro obecné výpočty. Vzhledem k tomu, že se jedná o klíčovou technologii použitou v této diplomové práci, je jí věnována samostatná kapitola, kde je popsána i její historie.

2.2 OpenGL

OpenGL (Open Graphics Library) je multiplatformní API pro vykreslování 2D a 3D grafiky [3]. Toto API je napsáno v jazyce C, ale je možné ho používat v široké řadě jiných programovacích jazyků. Jedná se o otevřený standard spravovaný konsorciem Khronos group. Mezi významné členy konsorcia patří Nvidia, AMD, Intel a další. OpenGL existuje v několika verzích, jako jsou OpenGL pro PC, OpenGL ES pro vestavěné systémy nebo OpenGL SC (Safety Critical) určený pro využití v letectví.

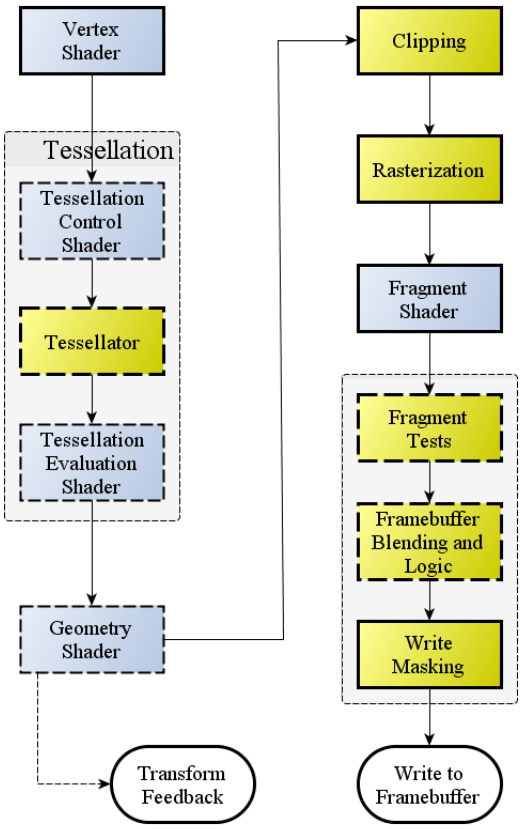
Jak již bylo řečeno, technologie OpenGL je určena pro vykreslování grafiky. Neumožňuje tedy vytvářet okna, spravovat vstupní zařízení a podobně. Tyto vlastnosti by totiž snížily multiplatformnost OpenGL a jsou tedy přenechány na starost jiným knihovnám, mezi které patří například GLUT, SDL a další.

2.2.1 Vykreslovací řetězec

Vykreslovací řetězec neboli rendering pipeline je sekvence kroků, které musí OpenGL provést, když vykresluje nějaký objekt [3]. Vykreslovací řetězec se skládá z několika částí, které jsou na obrázku 2.3, kde jsou modře vyznačeny programovatelné části. V dalším textu je přiblížena funkce pouze programovatelných částí vykreslovacího řetězce.

První částí celého řetězce je vertex shader pro zpracování jednotlivých vstupních vrcholů. Vertex shader zpracovává vždy pouze jeden vstupní vrchol. Jeho výstupem je opět pouze jeden vrchol. Nedochází tedy ke generování žádné další geometrie. V této části řetězce jsou většinou vrcholy transformovány a mohou zde být spočítány další atributy, které poslouží jako vstup následujícím jednotkám.

Za vertex shaderem následuje jednotka pro tessellaci. Ta je volitelná a nemusí být použita během vykreslování. V průběhu tessellace je vstupní primitivum, zvané v terminologii



Obrázek 2.3: Vykreslovací řetězec OpenGL [3].

OpenGL jako patch, rozděleno na menší části. Tessellace se využívá v různých technikách, jako je například změna úrovně detailu a podobně. Jednotka pro tessellaci se skládá ze tří částí, z nichž dvě jsou programovatelné. Tessellation control shader je první částí jednotky pro tessellaci a určuje, jak moc bude vstupní primitivum tessellováno. Tato část je následována částí tessellation primitiv generation, která není programovatelná. Její funkcí je vytváření nových primitiv ze vstupního patche. Způsob vytváření těchto primitiv je ovlivněn výstupem předchozí části, tedy tessellation control shaderem. Poslední částí, která je opět programovatelná, je tessellation evaluation shader. Jedná se o nejdůležitější část, která z výstupů předchozích částí počítá výsledné atributy nově vygenerovaných vrcholů.

Dalším stupněm vykreslovacího řetězce, který nemusí být podobně jako tessellace použit, je geometry shader. Geometry shader umožňuje generování dalších primitiv. Ze vstupního primitiva může vygenerovat více výstupních primitiv, ale také nemusí vygenerovat žádné. Výstup této části může pokračovat dál vykreslovacím řetězcem nebo může být uložen do paměti pro další zpracování.

Poslední programovatelnou částí vykreslovacího řetězce je takzvaný fragment shader. Je umístěn za jednotkou pro rasterizaci. Fragment shader zpracovává výstupy rasterizace zvané fragmenty. Každý fragment má pozici v rámci výstupního okna a několik dalších atributů z předchozích částí. Výstupem fragment shaderu je hloubka fragmentu a barva, která je zapsána do výstupního framebufferu.

Zbylé části vykreslovacího řetězce nejsou programovatelné a mají na starost ořezání primitiv dle pohledového objemu - clipping. Rasterizaci vstupních primitiv zajišťuje jednotka

rasterization. Poslední jednotkou před zápisem na výstup je jednotka obsahující části fragment tests, framebuffer blending and logic a write masking. Určuje jakým způsobem a jestli vůbec, budou data zapsána na výstup.

Mimo vykreslovací řetězec stojí takzvaný compute shader a není tudíž na obrázku 2.3 zobrazen. Jedná se o další programovatelnou jednotku určenou pro obecné výpočty na grafické kartě. V porovnání s předchozími částmi má několik rozdílů. Předchozí části měly pevně definovaný vstup a výstup. V compute shaderu je získání vstupů a zápis výstupů plně v jeho režii. Další rozdíl je v počtu volání compute shaderu. Počet volání programu v předchozích jednotkách vycházel z jejich povahy, takže například vertex shader je volán pro každý vstupní vrchol. U compute shaderu je počet volání zcela v rukou programátora. Maximální počet volání compute shaderu je ovšem omezen. Stejně tak je omezena velikost všech sdílených proměnných. Velikost těchto omezení může být zjištěna během činnosti programu.

2.3 CUDA

CUDA, neboli Compute Unified Device Architecture, je technologie společnosti Nvidia pro obecné výpočty na grafické kartě [23]. Jak již bylo uvedeno v úvodu, byly obecné výpočty na grafické kartě možné už s příchodem programovatelného vykreslovacího řetězce. Technologie CUDA však přinesla zcela nové možnosti v této oblasti a výrazně zjednodušila způsob programování grafických karet. Dále také umožnila využití grafických karet pro širší spektrum algoritmů, které v předchozím způsobu programování nemohly být z důvodů omezení programovacího modelu na GPU přeneseny.

2.3.1 Historie

První pokusy o využití grafické karty pro obecné výpočty je možné najít již v roce 2003 s příchodem programovatelného vykreslovacího řetězce [1]. Pro tyto účely byla využívána grafická rozhraní jako například OpenGL nebo DirectX. Využívání těchto rozhraní pro grafické účely s sebou neslo mnohé nevýhody.

Jako odpověď na tyto problémy představila společnost Nvidia dvě technologie: G80 sjednocenou grafickou a výpočetní architekturu a CUDA softwarovou a hardwarovou architekturu. Technologie CUDA umožnila programování GPU pomocí rozšířené množiny jazyka C. První grafickou kartou postavenou na nové architektuře je karta GeForce 8800, vydaná v roce 2006. Architektura G80 obsahovala několik vylepšení. Velmi důležitým bylo použití unifikovaného procesoru. Dřívější karty totiž využívaly samostatné jednotky pro výpočet vertex shaderu a samostatné jednotky pro výpočet fragment shaderu. V nové architektuře byly tyto jednotky sloučeny a vznikl nový procesor, který umožňoval spuštění vertex, geometry, pixel a výpočetních programů. Další klíčovou novinkou bylo zavedení SIMT (Single-Instruction Multiple-Thread) výpočetního modelu, ve kterém více nezávislých vláken vykonává souběžně stejnou instrukci. Poslední důležitou novinkou představenou architekturou G80 byla sdílená paměť a bariérová synchronizace, umožňující komunikaci mezi vlákny.

V roce 2008 byla představena revize architektury G80. Byla jí druhá generace unifikované architektury - G200. Architektura G200 obsahuje vyšší počet procesorových jader. Registrová sada každého procesoru byla zdvojnásobena a byla přidána podpora pro čísla v plovoucí desetinné čárce s dvojitou přesností. Dále byl vylepšen přístup k paměti.

Následujícím krokem ve vývoji GPU byla architektura Fermi vydaná v roce 2010. Tato architektura v sobě zahrnuje několik důležitých vylepšení vzniklých ze zpětné vazby s uži-

vateli předchozích generací. V nové architektuře byla zvýšena rychlost počítání s čísly v plovoucí desetinné čárce s dvojitou přesností. To umožnilo zrychlit řadu vědeckých výpočtů. Přibylo konfigurovatelné rozdělení paměti mezi sdílenou paměť a L1 cache. Také byla zvýšena rychlost atomických operací a rychlost přepínání kontextu.

Další vylepšení grafických karet přináší architektura Kepler z roku 2012 [4]. Tato architektura přináší například dynamický paralelismus umožňující GPU generovat další práci samo pro sebe, bez zásahu ze strany CPU. Dále je přidána technologie Hyper-Q. Ta umožňuje, aby více CPU jader spouštělo práci na jednom GPU, čímž je snížen čas nečinnosti CPU. V neposlední řadě byla přidána technologie GPUDirect. GPUDirect dovoluje různým GPU, umístěným v jednom počítači nebo na různých serverech propojených sítí, přímou výměnu dat bez zásahu CPU.

Vzhledem k tomu, že právě na této architektuře bude vyvíjen program, jenž je součástí této diplomové práce, je popisu architektury Kepler věnována následují kapitola.

2.3.2 Architektura Kepler

Architektura Kepler byla navržena pro rychlé výpočty s čísly v plovoucí desetinné čárce s dvojitou přesností. Toto je klíčovou vlastností pro mnoho vědeckých výpočtů [4]. Dalšími požadavky na tuto architekturu byly nízká spotřeba energie a nízké množství vyzářeného zbytkového tepla. Plná konfigurace architektury Kepler obsahuje patnáct streaming multiprocesorů (SMX) a šest 64-bitových řadičů paměti. Tato čísla se mohou produkt od produktu mírně lišit.

Nejdůležitější částí, která vykonává samotné výpočty, je streaming multiprocesor. Každý streaming multiprocesor obsahuje 192 CUDA jader s jednoduchou přesností. Každé jádro pak obsahuje plně zřetězenou aritmetickologickou jednotku pro výpočty s plovoucí desetinnou čárkou a pro výpočty s celými čísly. Součástí streaming multiprocesoru je dále šedesát čtyři jednotek s dvojitou přesností a třicet dva jednotek pro speciální funkce starajících se o výpočty funkcí, jako jsou sinus, cosinus atd. Dalších třicet dva jednotek je pro načítání/ukládání dat. Zastoupení jednotlivých jednotek ve streaming multiprocesoru je vyobrazeno na obrázku 2.4.

Další důležitou součástí každého streaming multiprocesoru je warp sheduler. V každém streaming multiprocesoru jsou celkem čtyři jednotky warp sheduler a osm jednotek instruction dispatch. Vlákna jsou v multiprocesoru plánována po skupinách třiceti dvou paralelních vláken, zvaných warp. Proto mohou být v každém streaming multiprocesoru vybrány a spuštěny čtyři warpy současně. Pro každý warp mohou být každý takt vybrány dvě nezávislé instrukce.

Všechny streaming multiprocesory jsou dále vybaveny šestnácti texturními jednotkami. Texturní jednotky jsou výhodné, pokud potřebuje program načítat data podle nějakého vzoru. Příkladem je načítání čtyř okolí bodu při zpracování obrazu.

Streaming multiprocesory také obsahují jak sdílenou paměť/L1 cache, tak cache o velikosti 48 kB pro data určená pouze pro čtení. Celková velikost paměti vyhrazené pro sdílenou paměť/L1 cache je 64 kB. Tuto paměť je možné rozdělit mezi sdílenou paměť a L1 cache následujícím způsobem: buď 48 kB pro sdílenou paměť a 16 kB pro L1 cache nebo naopak. Případně může být paměť rozdělena rovnoměrně, což znamená 32 kB pro sdílenou paměť a 32 kB pro L1 cache.

Obrázek 2.4: Streaming multiprocesor [4].

2.3.3 Nástroje

Pro podporu vývoje CUDA aplikací jsou kromě překladače CUDA programů k dispozici i další programy. Mezi nejzajímavější patří Nvidia Nsight a Nvidia Visual Profiler. Oba programy mají za účel zjednodušit vývoj aplikací a jejich optimalizaci.

Nvidia Nsight je dostupný ve dvou verzích a to ve verzi pro Visual Studio nebo pro Ecllipse. Po nainstalování se zaintegruje do těchto vývojových prostředí a umožňuje debugování kódu určeného pro GPU, včetně debugování shader programů. Program Nsight také podporuje profilování výsledného kódu. Tím je umožněno nalezení případných úzkých hrdel, která snižují výkonnost vyvíjené aplikace.

Nvidia Visual Profiler má podobnou funkci jako Nsight. Dovoluje provést sérii testů za účelem nalezení problematických míst z pohledu výkonnosti a jejich následnou optimalizaci. Jedná se o samostatný program, který se neintegruje do žádného vývojového prostředí.

2.4 Spolupráce OpenGL a CUDA

Důležitou vlastností technologie CUDA je možnost spolupráce s některým ze dvou nejpoužívanějších grafických API, kterými jsou OpenGL a DirectX [9] a [22]. Tato vlastnost je v rámci práce využita a to ve spojitosti s technologií OpenGL. Proto se tato kapitola omezí pouze na tuto technologii, i když použití technologie DirectX je principiálně velice podobné.

Spolupráce OpenGL a CUDA umožňuje, aby tyto dvě technologie sdílely mezi sebou data. Proto je možné vykreslit data spočítaná pomocí CUDA přímo z paměti grafické karty bez jakýchkoliv datových přesunů a bez zásahu ze strany CPU. Další možností využití je postprocesing dat zpracovaných pomocí OpenGL.

Nastavení CUDA na spolupráci s OpenGL je relativně jednoduché. Vyžaduje ale několik kroků, které není nutné v běžné CUDA aplikaci provádět. Nejprve je nutné získat identifikátor CUDA zařízení, které je použito pro spolupráci CUDA a OpenGL. Poté je nutné sdělit CUDA běhovému prostředí, že toto zařízení budeme chtít použít tímto způsobem. Následuje standardní nastavení OpenGL kontextu. Po nastavení CUDA a OpenGL je již možné vygenerovat sdílené paměťové objekty. Takovéto objekty musí být vygenerovány pomocí OpenGL volání. Po uspěšné alokaci paměti pomocí OpenGL následuje registrace této paměti pomocí CUDA jako grafický prostředek. V tento okamžik existují pro jeden paměťový objekt dva ukazatele. Jeden je možné používat pouze v rámci OpenGL volání, druhý zase pouze v rámci CUDA volání. Pro práci s takto alokovanou pamětí je nutné získat adresu v paměti grafické karty. Toho je dosaženo voláním funkce, která z ukazatele vrací adresu. Adresu je potom možné používat jako adresu paměti alokované přímo pomocí CUDA volání. Před jakoukoliv manipulací s touto pamětí je nutné paměť namapovat. Poté je možné provést změny. Následně po ukončení práce s pamětí je nutné ji zase odmapovat. To zajistí, že všechny zápisy do této paměti již proběhly a je možné ji používat v rámci OpenGL.

Kapitola 3

Další použité technologie

Tato kapitola se zabývá popisem pomocných technologií, které jsou použity při implementaci výsledného programu. Nejprve je uveden stručný popis knihovny Qt, která je použita pro práci s grafickým uživatelským rozhraním. Poté je probrána technologie OpenMP umožňující jednoduše paralelizovat provádění kódu na CPU. OpenMP je využita na optimalizaci CPU kódu simulace chování tkanin vzniklého v přípravných fázích implementace.

3.1 Qt

Qt je framework pro multiplatformní vývoj v jazyce C++. Umožňuje však vývoj i v jiných jazycích, jako je například Python. Mezi podporované platformy patří většina dnes široce rozšířených platforem. Jedná se především o Windows a Linux, ale také o mobilní platformy jako je Android, iOS a Windows Phone. Tento framework také umožňuje vývoj i pro vestavěné systémy. Qt je možné využívat pod několika různými licencemi [7]. Vývoj je umožněn pod licencemi LGPL a GPL pro open-source vývoj, dále je umožněn vývoj pro studijní účely a v neposlední řadě komerční vývoj. Framework Qt je velice populární a je využíván různými projekty, mezi které patří KDE, Skype, Google Earth a další [6].

Součástí Qt je i celý ekosystém nástrojů určených pro vývoj s využitím tohoto frameworku. Nejdůležitějším nástrojem je IDE Qt Creator, jehož součástí jsou nástroje určené pro návrh grafického uživatelského rozhraní, jazykovou lokalizaci aplikace, atd. Dalším nástrojem, jenž je součástí ekosystému Qt a je využitý v této diplomové práci, je doplněk pro Visual Studio. Umožňuje pohodlný vývoj Qt aplikace v tomto IDE.

3.2 OpenMP

OpenMP bylo poprvé vydáno v roce 1997. Jedná se o standardní aplikační programovací rozhraní umožňující psaní paralelních programů v jazycích C/C++ a Fortran [14]. Standard má na starost skupina OpenMP Architecture Review Board, jejímiž členy jsou významní výrobci hardwaru a překladačů.

Hlavní výhodou OpenMP je fakt, že každý sériový kód je možné pomocí této technologie jednoduše paralelizovat. OpenMP se skládá ze sady direktiv pro kompilátor a sady knihovních funkcí. Tento přístup je výhodný. Pokud kompilátor neumí zpracovávat OpenMP, tak jsou OpenMP direktivy ignorovány a nedochází k žádným chybám při překladu. Tudíž správně napsaný paralelní program je validním sériovým kódem. [11] Toto rozhraní využívá

architekturu sdílené paměti. Těžištěm celé technologie jsou tedy vlákna, která mezi sebou paměť sdílejí.

Každý OpenMP program začíná s jedním vláknem zvaným hlavní vlákno (master thread). V tomto vlákně probíhá sekvenční vykonávání programu, dokud program nenarazí na paralelní sekci. Poté je vytvořena skupina vláken, která spolupracují na výpočtu daného úkolu. Ukončením paralelní sekce je ukončen také běh těchto vláken. Program pak pokračuje dál ve výpočtu pouze v hlavním vlákně. Pokud program při svém vykonávání narazí na další paralelní sekci, tak se celý výše uvedený cyklus opakuje. Počet paralelních sekcí není ničím omezen.

Kapitola 4

Simulace chování tkanin

Tkaniny a simulace jejich chování jsou nedílnou součástí moderní počítačové grafiky. Využití simulace tkanin je možné nalézt v počítačových hrách, kde se jedná o simulaci v reálném čase. Dále se nachází i ve filmech na virtuálních postavách oblečených v látkových oděvech, případně ve virtuálních prostředích tvořených tkaninou.

Složitost simulace v reálném čase je samozřejmě mnohem větší, protože na získání kvalitního výstupu je k dispozici malé množství času. Oproti tomu situace při použití simulace tkanin ve filmech je jednodušší o fakt, že na vykreslení jednoho snímku je neporovnatelně více času. Z tohoto důvodu lze užít mnohem komplexnější metody, jejichž výstupem je mnohem kvalitnější simulace chování tkanin. Přes dílčí odlišnosti jsou si obě využití simulace tkanin značně podobná. V základu vychází ze stejných metod, ale v grafice počítané v reálném čase je nutné užití různých optimalizací. Tyto optimalizace jsou občas na úkor vizuální kvality a fyzikální přesnosti simulace. Ovšem díky překotnému vývoji grafických karet je možné dosahovat vizuálně velice dobrých výsledků i v reálném čase.

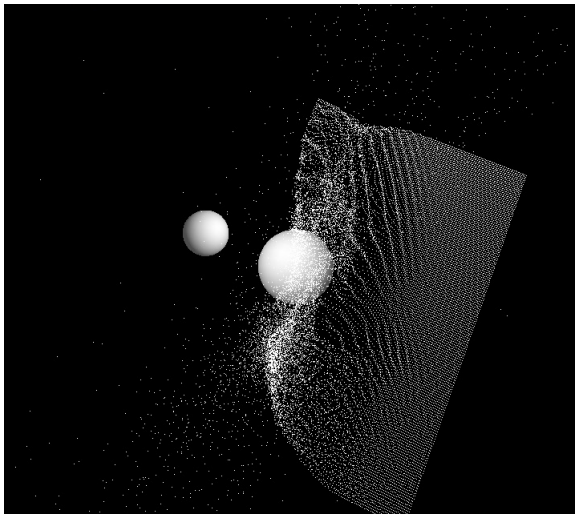
Pro simulaci tkanin lze použít celou řadu metod, jako je například metoda konečných prvků. Tato metoda se ale v praxi příliš nepoužívá. Pro simulaci chování tkanin by vyžadovala speciální komplexní prvek, který by byl schopen reprezentovat ohyb tkaniny [15]. Nejčastěji používanou metodou pro simulaci chování tkanin jsou částicové systémy. Na rozdíl od metody konečných prvků jsou obecně výpočetně méně náročné. Ale na druhou stranu je těžké je správně nastavit a nemusejí vždy konvergovat ke správnému řešení. Nicméně částicové systémy lze kromě simulace tkanin využít i pro simulaci dalších fenoménů, které by se jinými postupy simulovaly těžce. Mezi častá využití částicových systémů patří simulace vlasů, ohně, kouře a různých dalších efektů. Obecnému popisu částicových systémů je věnována následující kapitola.

4.1 Částicové systémy

Částicové systémy jsou ve své základní podobě velice jednoduché, ale i přesto za nimi stojí silný matematický základ. Je možné nalézt mnoho různých variant částicových systémů, které se snaží vylepšit jejich chování. Částicové systémy chápou simulované těleso jako soustavu hmotných bodů, které jsou navzájem spojené vazbami (pružinami) [18]. Tyto body se nazývají částice. Reprezentují hmotná tělesa o malých rozměrech, na která působí z vnější strany různé síly. Každá hmotná částice má v každém simulačním čase i svoji pozici a okamžitou rychlost, která může být i nulová. Na každou částici působí síla, která je výslednicí všech dílčích sil působících na těleso.

Při simulaci tohoto systému je nutné v každém kroku simulace spočítat řadu diferenciálních rovnic. Pro výpočet diferenciálních rovnic je možné použít různé metody numerické integrace. Výběr metody numerické integrace s sebou přináší několik problémů. Nejvýznamnějším problémem je problém stability zvolené integrační metody. Některé metody totiž mohou pro svoji stabilitu vyžadovat poměrně malý integrační krok. To nemusí být vhodné pro použití v grafice počítané v reálném čase. Velikost integračního kroku je také ovlivněna tuhostí pružin, kterými jsou částice spojeny. Čím je pružina tužší, tím menší krok je potřeba pro udržení stability. Nestabilita simulovaného systému vede ve většině případů až k jeho rozpadu, kdy se jednotlivé části nekontrolovatelně rozletí. Příklad nestabilního systému je možné nalézt na obrázku 4.1. Mezi další problémy částicových systémů dle [15] patří:

- Chování tkaniny je ovlivněno tím, jak je vytvořena simulační síť
- Obtížnost nastavení konstant pružin pro dosažení požadovaného výsledku
- Fyzikální nepřesnost



Obrázek 4.1: Nestabilní systém.

Dalším hlediskem při výběru integrační metody je složitost její implementace ovlivňující rychlost výpočtu. Některé integrační metody vyššího řádu mohou mít také problém při startu výpočtu. V tomto případě je nutné několik kroků předem vypočítat pomocí metod nižšího řádu. Mezi metody, které lze použít, patří například Eulerova metoda. Tato metoda je výhodná pro svoji jednoduchou implementaci a je i poměrně rychlá. Na druhou stranu patří mezi ty metody, které vyžadují malý integrační krok, aby se předešlo numerické nestabilitě. Dále je možné použít některou metodu vyššího řádu. Mezi ně patří například metoda Runge-Kutta. Tato metoda nemá problémy se stabilitou, ale je zase po implementační stránce složitější. Problematickým faktem je i to, že na začátku výpočtu je nutné několik kroků předem vypočítat jinou metodou.

Výběr metody pro numerickou integraci značně ovlivňuje použitelnost simulace. Metody numerické integrace se dělí na dvě hlavní části [15], kterými jsou explicitní a implicitní metody numerické integrace. Liší se od sebe navzájem právě stabilitou a složitostí výpočtu.

Jednoduché explicitní metody nemají žádnou záruku své stability. Implicitní metody jsou zase výpočetně náročnější.

4.1.1 Explicitní metody

Nejjednodušší metodou je explicitní Eulerova metoda [15] a [12]. Tato metoda není výpočetně náročná, ale trpí právě problémem nestability při zvolení příliš velkého integračního kroku. V případě využití této metody pro grafiku počítanou v reálném čase je potřeba provést několik simulačních kroků na jeden vykreslený rámec. Novou pozici x' a rychlost v' částice je tedy pomocí Eulerovy integrace možné spočítat podle následujících vzorců:

$$v^{t+1} = v^t + \Delta t f(x^t, v^t)/m \tag{4.1}$$

$$x^{t+1} = x^t + \Delta t v^t \tag{4.2}$$

kde x je pozice částice, v je rychlost částice a t je časový posun. Stabilita metody se dá vylepšit nahrazením v^{t+1} za v^t v rovnici 4.2. Tento trik je možný díky tomu, že v této části výpočtu je nová rychlost už spočítána.

Způsob jak snížit úroveň nestability je použití metod vyšších řádů. Mezi často využívané metody patří metody Runge-Kutta a to především metody druhého a čtvrtého řádu. Metody Runge-Kutta vyhodnocují síly vícekrát během jednoho kroku výpočtu, a tím snižují numerickou nestabilitu. Runge-Kutta druhého řádu je definovaná následujícími vzorci:

$$a_1 = v^t \tag{4.3}$$

$$a_2 = f(x^t, v^t)/m \tag{4.4}$$

$$b_1 = v^t + \frac{\Delta t}{2} a_2 \tag{4.5}$$

$$b_2 = f(x^t + \frac{\Delta t}{2} a_1, v^t + \frac{\Delta t}{2} a_2)/m \tag{4.6}$$

$$x^{t+1} = x^t + \Delta t b_1 \tag{4.7}$$

$$v^{t+1} = v^t + \Delta t b_2 \tag{4.8}$$

Vzhledem k tomu, že síly jsou vyhodnoceny dvakrát během jednoho simulačního kroku, je zvýšena přesnost výpočtu, nicméně vzroste i výpočetní náročnost [15]. Jeden krok pomocí metody Runge-Kutta druhého řádu je časově stejně náročný jako dva kroky Eulerovy metody.

Dalšího zvýšení přesnosti lze dosáhnout použitím metody Runge-Kutta čtvrtého řádu, která patří k nejpoužívanějším metodám v oblasti vědeckých výpočtů. Síly jsou vyhodnocovány čtyřikrát během jednoho simulačního kroku a tím je opět zvýšena přesnost výpočtu. Tato metoda je definována pomocí níže uvedených vzorců:

$$a_1 = v^t \tag{4.9}$$

$$a_2 = f(x^t, v^t)/m \tag{4.10}$$

$$b_1 = v^t + \frac{\Delta t}{2} a_2 \tag{4.11}$$

$$b_2 = f(x^t + \frac{\Delta t}{2} a_1, v^t + \frac{\Delta t}{2} a_2)/m \tag{4.12}$$

$$c_1 = v^t + \frac{\Delta t}{2} b_2 \tag{4.13}$$

$$c_2 = f(x^t + \frac{\Delta t}{2} b_1, v^t + \frac{\Delta t}{2} b_2)/m \tag{4.14}$$

$$d_1 = v^t + \frac{\Delta t}{2} c_2 \tag{4.15}$$

$$d_2 = f(x^t + \Delta t c_1, v^t + \Delta t c_2)/m \tag{4.16}$$

$$x^{t+1} = x^t + \frac{\Delta t}{6}(a_1 + 2b_1 + 2c_1 + d_1) \tag{4.17}$$

$$v^{t+1} = v^t + \frac{\Delta t}{6}(a_2 + 2b_2 + 2c_2 + d_2) \tag{4.18}$$

Metody Runge-Kutta využívaly pro zvýšení přesnosti výpočtu vícenásobného vyhodnocování sil v průběhu jednoho simulačního kroku.

Dalším způsobem jak dosáhnout vyšší přesnosti je použití hodnot vypočítaných v předchozích krocích. Tohoto principu využívá řada metod. Nejpoužívanější metodou pro simulace v reálném čase je verlet metoda. Metoda verlet je výpočetně jednoduchá a využívá hodnoty pozic z předchozího výpočetního kroku. Výpočet pomocí verlet metody je definován následujícími vzorci [18]:

$$x' = 2x - x^* + a * \Delta t^2 \tag{4.19}$$

$$x^* = x \tag{4.20}$$

Při použití této metody není nutné u každé částice uchovávat její rychlost, ale místo toho se uchovává její předchozí pozice x^*. Rychlost je aproximována pomocí vztahu $x - x^*$ v použitém vzorci. Metoda verlet je poměrně stabilní a rychlá, ale není přesná. Pro správnou funkci této metody se v průběhu simulace nesmí měnit velikost časového kroku.

4.1.2 Implicitní metody

Implicitní metody mají jednu zásadní výhodu oproti explicitním metodám. A to je jejich stabilita za jakýchkoliv podmínek. Této vlastnosti lze s výhodou využít v simulacích počítaných v reálném čase, kde je možné simulační krok nastavit podle požadované snímkové frekvence. Na druhou stranu mohou být tyto metody výpočetně více náročné než explicitní metody [15].

Oblíbenou implicitní metodou je implicitní Eulerova metoda. Implicitní Eulerovu metodu, jak je uvedena v následujících vzorcích, je možné z explicitní Eulerovy metody získat drobnými úpravami:

$$v^{t+1} = v^t + \Delta t f(x^{t+1})/m \tag{4.21}$$

$$x^{t+1} = x^t + \Delta t v^{t+1} \tag{4.22}$$

Implicitní integrace vnáší do výpočtu poměrně velké tlumení. Proto není nutné dodávat uměle fyzikální tlumení. Hlavní změnou je nicméně použití pozicí a rychlosti z nového kroku také na pravých stranách rovnic. Nyní již není možné tyto rovnice explicitně vyhodnotit a dohromady tvoří nelineární algebraický systém.

Způsob, jak takový systém řešit pro celou síť reprezentující tkaninu, je vytvořit z pozic, rychlostí a sil vektory následujícím způsobem:

$$x = \left[x_1^T, ..., x_n^T\right]^T \tag{4.23}$$

$$v = \left[v_1^T, ..., v_n^T\right]^T \tag{4.24}$$

$$f(x) = \left[f_1(x_1, ..., x_n)^T, ..., f_n(x_1, ..., x_n)^T\right]^T \tag{4.25}$$

Dále je nutné vytvořit matici $M \in \Re^{3n\times 3n}$, která je diagonální s prvky m_1, m_1, m_1, m_2, m_2, m_2,...,m_n, m_n, m_n podél diagonály. Výsledkem těchto definic je systém:

$$Mv^{t+1} = Mv^t + \Delta t f(x^{t+1}) \tag{4.26}$$

$$x^{t+1} = x + \Delta v^{t+1} \tag{4.27}$$

Dosazením rovnice 4.27 do rovnice 4.26 získáme výsledný systém s neznámou rychlostí v^{t+1}:

$$Mv^{t+1} = Mv^t + \Delta t f(x^t + \Delta t v^{t+1}) \tag{4.28}$$

Takové systémy je možné řešit například pomocí Newton-Raphson solveru. Algoritmus výpočtu pomocí Newton-Raphson solveru začíná odhadem neznámé v^{t+1}, který je pak dále iterativně vylepšován. Nyní jsou rovnice linearizovány a jsou vyřešeny za účelem nalezení lepšího odhadu. Celý cyklus je opakován, dokud se chyba výpočtu nedostane pod určitou hranici. Takové opakování výpočtu s předem nejasným počtem opakování je pro použití v reálném čase nepřípustné. Tudíž linearizace je provedena pouze jednou a aktuální hodnota rychlosti v^t je použita jako odhad rychlosti v^{t+1}. Linearizace sil v x^t vede na následující vzorce:

$$\begin{aligned} Mv^{t+1} =& Mv^t + \Delta t \left[f(x^t) + \frac{\delta}{\delta x} f(x^t) \cdot (\Delta t v^{t+1})\right] \\ =& Mv^t + \Delta t f(x^t) + \Delta t^2 K v^{t+1} \end{aligned} \tag{4.29}$$

kde $K \in \Re^{3n\times 3n}$ je Jakobián f. Přesunutím výrazů získáme standardní systém pro neznámou rychlost:

$$\left[M - \Delta t^2 K\right] v^{t+1} = Mv^t + \Delta t f(x^t) \tag{4.30}$$

$$Av^{t+1} = b \tag{4.31}$$

kde A je $3n\times 3n$ rozměrná matice a b je $3n$ rozměrný vektor. Metodou vhodnou pro řešení takových systémů v simulacích počítaných v reálném čase je metoda sdružených gradientů. Pro výpočet je důležitá hodnota K, ke které každá síla mezi částicemi i a j přidává čtyři 3×3 matice $K_{i,i}, K_{i,j}, K_{i,j}, K_{j,j}$ na pozice $(3_i, 3_i)$,$(3_i, 3_j)$,$(3_j, 3_i)$,$(3_j, 3_j)$. Jednotlivé 3×3 matice jsou definovány následovně:

$$\begin{aligned} K_{i,i} =& \frac{\delta}{\delta x_i} f^S(x_i, x_j) \\ =& k_s \frac{\delta}{\delta x_i} \left((x_j - x_i) - l_0 \frac{x_j - x_i}{|x_j - x_i|}\right) \\ =& k_s \left(-I + \frac{l_0}{l}\left[I - \frac{(x_j - x_i)(x_j - x_i)^T}{l^2}\right]\right) \\ =& - K_{i,j} = K_{j,j} = -K_{j,i} \end{aligned} \tag{4.32}$$

kde I je 3×3 matice identity a $l = |x_j - x_i|$ neboli aktuální délka pružiny. Na závěr je vyhodnoceno b z rovnice 4.31, výsledný systém je pak vyhodnocen pro v^{t+1}. Nakonec mohou být rychlosti aktualizovány pomocí rovnice 4.27.

4.1.3 Síly působící na částice

Na každou částici může působit několik různých sil. Jejich výslednice F je použita pro výpočet zrychlení pomocí Newtonova zákona. Takových sil existuje mnoho, ale podle [18] a [8] patří mezi základní síly globální gravitační síla, síla odporu prostředí, pružná síla, vzájemná síla a Coulombova síla.

Globální gravitační síla působí na celý systém a je definována pomocí vzorce:

$$F_g = mg \tag{4.33}$$

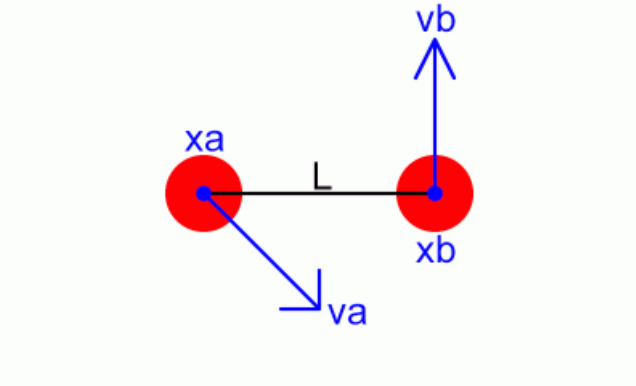
kde g je vektor směřující kolmo dolů a jeho velikost samozřejmě odpovídá gravitačnímu zrychlení a m je hmotnost částice.

Síla odporu prostředí je rovněž silou, která působí na celý systém a je definována vztahem:

$$F_d = -k_d v \tag{4.34}$$

Tato síla působí proti směru pohybu částice a roste lineárně s rychlostí v částice. Koeficient k_d je koeficient tlumení.

Pružná síla je, na rozdíl od ostatních, silou působící lokálně mezi dvěma částicemi, spojenými navzájem vazbou. Vzájemné působení dvou částic je na obrázku 4.2.



Obrázek 4.2: Působení sil [18].

Pružná síla je definována pomocí následujícího vztahu:

$$F_a = -\left[k_s\left(|L| - r\right) + k_d \frac{L'L}{|L|}\right] \frac{L}{|L|} \tag{4.35}$$

kde F_a je výsledná síla, L je vzdálenost částic $x_a - x_b$, L' je vektor rozdílu rychlostí částic $v_a - v_b$, k_s je koeficient pružnosti, k_d je koeficient tlumení a r je délka pružiny spojující částice v klidovém stavu.

Síla vzájemné gravitace je rovněž silou lokální. Velikost a směr je dán níže uvedeným vzorcem:

$$F_{ab} = \frac{G m_a m_b}{|L|^2} \frac{L}{|L|} \tag{4.36}$$

Jednotlivé symboly jsou definovány následovně: F_{ab} je výsledná síla, G je gravitační konstanta, jejíž velikost je $6.672 \times 10^{-11} N m^2 kg^{-2}$, m_a a m_b jsou hmotnosti jednotlivých částic a L je vektor vzdálenosti částic.

Poslední silou z výše uvedeného výčtu je Coulombova síla. Ta se řídí Coulombovým zákonem:

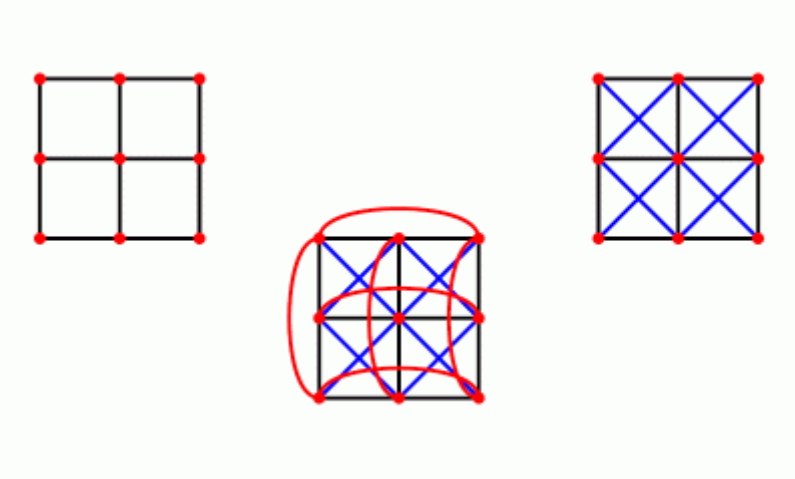
$$F_{ab} = \frac{k q_a q_b}{|L|^2} \frac{L}{|L|} \tag{4.37}$$

kde F_{ab} je výsledná síla, $k = 8.895 \times 10^9 N m^2 C^{-2}$, q_a a q_b jsou elektrické náboje částic a L je vektor reprezentující vzdálenost částic.

Pro simulaci částicového systému mohou být použity i další síly. Naopak zase nemusí být použity všechny síly z výše uvedeného seznamu. Použití jednotlivých sil závisí na tom, jaký systém se pokoušíme pomocí částicových systémů simulovat.

4.1.4 Provázání částic

Částicový systém je reprezentován pomocí množiny částic spojených vzájemně vazbami. Tyto vazby si lze představit jako pružiny, kterými na sebe jednotlivé částice navzájem působí. Jednotlivé částice jsou navzájem propojeny pružnými vazbami, viz obrázek 4.3 levá část. Tento způsob propojení tkaniny ale není dostatečný z pohledu stability. Tkanina tvořená pouze takto spojenými částicemi by nedržela svůj tvar a zhroutila by se do úzké "nudle". Z toho důvodu je nutné přidat další vazby, které uvedenému nežádoucímu chování zabrání. Přidáním diagonálních vazeb lze předejít zborcení tkaniny, viz obrázek 4.3 pravá část. Dalšího vylepšení chování systému lze dosáhnout přidáním vazeb, které spojují částice přes jednu částici, jak je tomu na obrázku 4.3 prostřední část.



Obrázek 4.3: Propojení částic [18].

4.1.5 Kolize tkaniny a její detekce

Detekce a správná reakce na kolize je i v současnosti stále aktivní oblastí výzkumu. Detekce kolizí je výpočetně velice náročná, především v oblasti simulací počítaných v reálném čase. Samotný algoritmus detekce a reakce na kolizi není principiálně náročný, ale musí být použitý na celkem velkou množinu dat. V případě naivního řešení, kdy jsou všechny částice testovány na kolizi, se jedná o celkem zdlouhavý proces, který se může stát úzkým hrdlem celého výpočtu. Pokud má tkanina N částic a objekt má M vrcholů je výsledná složitost detekce kolize $O(MN)$. Složitost výpočtu kolize tkaniny samé se sebou je $O(N2)$. První optimalizací je použití různých stromových struktur a ohraničujících obsahů na zmenšení množiny testovaných dat. Dalším problémem je správná reakce na kolizi. Většina metod dle [16] řeší kolize jednotlivě, což může vést k nekonzistencím při vícenásobné kolizi. Při řešení vícenásobné kolize pomocí série řešení jednotlivých kolizí může dojít k tomu, že řešení jedné kolize je v konfliktu s řešením jiné kolize. Tudíž vícenásobná kolize není nakonec vyřešena správným způsobem. Následující text stručně probere některá možná řešení problému detekce a reakce na kolize.

Principy zpracování kolizí je možné rozdělit podle několika hledisek. Prvním hlediskem je rozdělení na detekci kolize a reakci na vzniklou kolizi. Dále mohou být metody zpracování kolizí rozděleny na zpracování kolizí tkaniny s ostatními objekty a kolize tkaniny samé se sebou. Tyto dva případy se principiálně neliší, ale je možné u nich použít různé optimalizační techniky.

Kolize samotné lze dle [16] rozdělit do dvou skupin:

- Kolize vrcholu s trojúhelníkem - vrchol jednoho tělesa prošel trojúhelníkem druhého tělesa
- Kolize hrany s hranou - hrana jednoho tělesa prošla hranou druhého tělesa

Řešení navržené v [16] pracuje s oběma výše uvedenými typy kolizí. Pro snížení počtu jednotlivých testů využívá stromovou strukturu obalových těles, kdy je tkanina rekurzivně dělena do překrývajících se zón. Takto zbudovaná struktura umožňuje rychlé vyřazení zón, jejichž obalová tělesa se nepřekrývají, a tudíž zde nemůže dojít ke kolizi. Trojúhelníky z překrývajících se zón jsou následně testovány na kolize vrcholu s trojúhelníkem a hrany s hranou. Autoři tohoto řešení navrhují pro kolize tkaniny samé se sebou ještě další optimalizaci. Ta je založena na tom, že pokud daná zóna má nízké zakřivení, pak se zóny v ní obsažené nemohou vzájemně protnout. Zakřivení zóny je vyhodnoceno pomocí normálových vektorů jednotlivých trojúhelníků obsažených v této zóně. S využitím této techniky je možné z testu na kolize vyřadit poměrně velké množství větví ze stromu obalových těles, kde díky jejich nízkému zakřivení nemůže dojít ke kolizi tkaniny samotné se sebou.

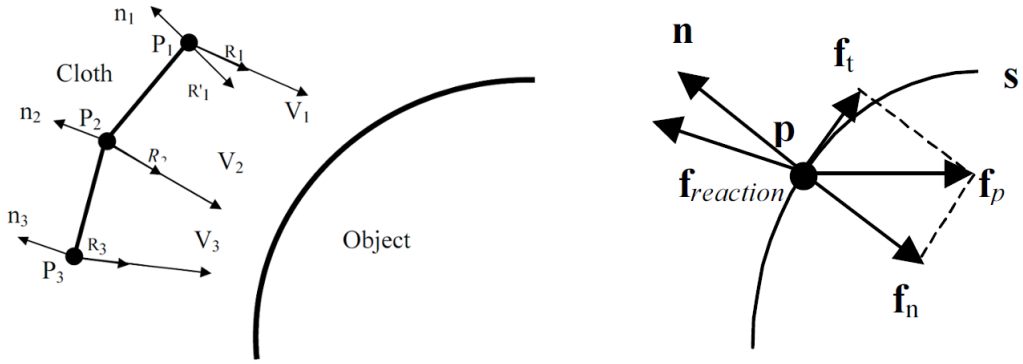
Jiné řešení kolize je možné nalézt v [20]. Algoritmus použitý v tomto článku je založen na využití metody sledování paprsků. Princip metody detekce kolize je vidět na obrázku 4.4 levá část. Pro každý vrchol je zkonstruovaný paprsek (R_n na obrázku 4.4) začínající v tomto vrcholu a směřující ve směru vektoru rychlosti. Paprsku je nastavena určitá velikost. Pokud se takto zkonstruovaný paprsek protne s nějakým objektem, došlo ke kolizi. Jestliže se tento paprsek neprotne s žádným objektem, je zkonstruován druhý paprsek. Druhý paprsek je na obrázku označen jako R'_n. Je vytvořen tak, že opět vychází z testovaného vrcholu a jeho směr je opačný ke směru normálového vektoru. Paprsek je následně opět testován na protnutí s nějakým objektem. V případě že dojde ke kolizi, je spočítána síla $f_{reaction}$, která reaguje na kolizi, viz obrázek 4.4 pravá část. Síla $f_{reaction}$ je spočítána dle následujícího vzorce:

$$f_{reaction} = -C_{fric} f_t - f_n \tag{4.38}$$

kde C_{fric} je koeficient tření a f_t, f_n jsou tangentová a normálová složka síly f_p působící na částici p. Rychlost vrcholu je možné spočítat dle vzorce 4.39, ve kterém C_{refl} je koeficient odrazu a v_t, v_n jsou opět tangentová a normálová složka síly působící na bod p.

$$v_{res} = C_{fric} v_t - C_{refl} v_n \tag{4.39}$$

Dosud diskutované metody řešily kolize až v okamžiku, když nastaly. Algoritmus [10] navrhuje jiné řešení pro kolize látky samé se sebou a to je kolizím předcházet. Kolizím lze předcházet tak, že částice, které jsou u sebe blízko, jsou udržovány v určité vzdálenosti od sebe. Takovým přístupem lze ušetřit hodně výpočetního času. Test na to, jestli jsou dvě částice blízko u sebe, lze totiž provést poměrně jednoduše. Nicméně je vhodné snížit počet takových testů na minimum. Vhodným způsobem jak toho dosáhnout je opět využití nějaké hierarchické struktury obalových těles. Pro zpracování kolizí mezi tkaninou a jinými objekty je v rámci tohoto algoritmu navržen poměrně rychlý princip. Algoritmus umožňuje zpracovávat kolize pouze s koulemi a kvádry s hranami zarovnanými s osami. Metoda je založena na tom, že je jednoduché spočítat vzdálenost bodu k těmto primitivům. Pokud je vzdálenost menší než určitá hodnota, je částice posunuta zpět na povrch primitiva a to ve směru svého normálového vektoru.



Obrázek 4.4: Využití sledování paprsků pro detekci a reakci na kolize [20].

Kapitola 5

Návrh programu

Obsahem této kapitoly je návrh uživatelského rozhraní programu, který bude vytvořen v rámci této diplomové práce. Jedná se o grafické uživatelské rozhraní, které umožňuje interakci se simulovanou tkaninou. Druhá část kapitoly obsahuje návrh testovací scény, na které je ověřena korektnost simulace tkaniny.

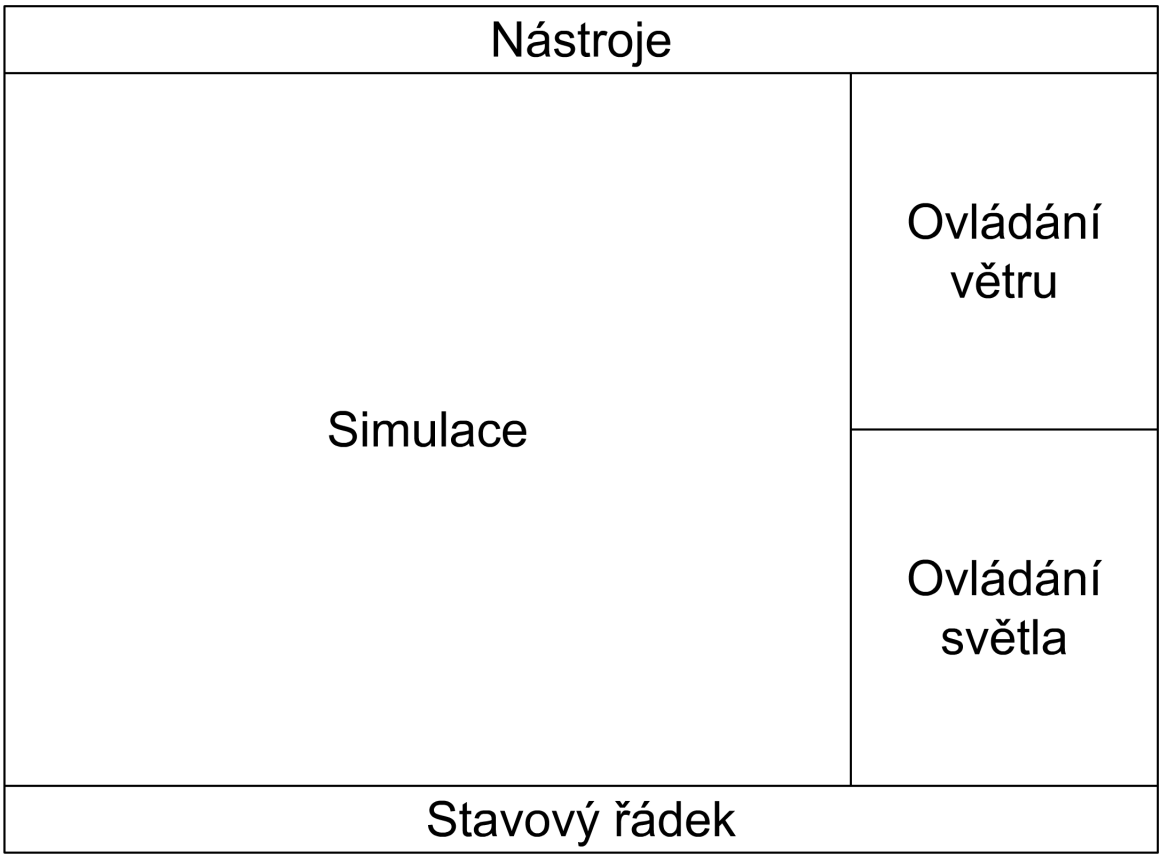
5.1 Návrh uživatelského rozhraní

Okno programu je rozděleno na několik částí. V největší části je zobrazován výstup simulace. Zde je také umožněna interakce se simulovanou tkaninou pomocí různých modifikačních nástrojů. Nad částí pro výstup simulace je umístěna lišta s různými nástroji. Mezi tyto nástroje patří nástroj pro otočení scény, nástroj pro změnu způsobu vykreslování simulované tkaniny. Tkaninu je možné vykreslit pomocí simulačních bodů. Případně lze simulační body spojit pomocí čar. Dále je umožněno tkaninu vykreslit pomocí trojúhelníků, které jsou tvořeny těmito simulačními body. V této liště jsou také umístěna tlačítka pro spuštění a pozastavení simulace.

Výše uvedené nástroje jsou rovněž součástí kontextového menu, které je možné vyvolat klikem pravým tlačítkem myši v oblasti vykreslování simulace. Napravo od hlavní části programu je umístěn panel, ve kterém je možné zapnout a vypnout vítr včetně možnosti měnit jeho sílu a směr. Vítr představuje další sílu působící na tkaninu. Pod panelem pro ovládání větru je panel, který umožňuje měnit pozici světla. Poslední částí navrhovaného uživatelského rozhraní je stavový řádek, kde je zobrazena rychlost simulace ve formátu počtu snímků za sekundu. V pravé části stavového řádku je umístěn posuvník umožňující změnu počtu simulačních bodů. Rozvržení okna výsledného programu je vidět na následujícím obrázku.

5.2 Návrh testovací scény

Testovací scéna je výchozí scénou při spuštění výsledného programu. Tato scéna samozřejmě obsahuje simulovanou tkaninu, která je zavěšena na dvou pevných bodech a visí ze "stropu". Scéna dále obsahuje plochu, která představuje "podlahu". Zde jsou umístěny různé objekty, se kterými simulovaná tkanina na sebe vzájemně působí.



Obrázek 5.1: Návrh rozvržení okna výsledného programu.

Kapitola 6

Implementace

V této části diplomové práce je uveden popis konkrétních metod zvolených pro implementaci simulace chování tkanin v reálném čase. Jedná se o následující implementace: čistě CPU implementace, CPU implementace optimalizovaná pomocí technologie OpenMP, CUDA implementace a optimalizovaná CUDA implementace. Popis společné části všech výše zmíněným implementací, tj. způsob vykreslování simulace, tuto kapitolu uvozuje.

6.1 Vykreslování

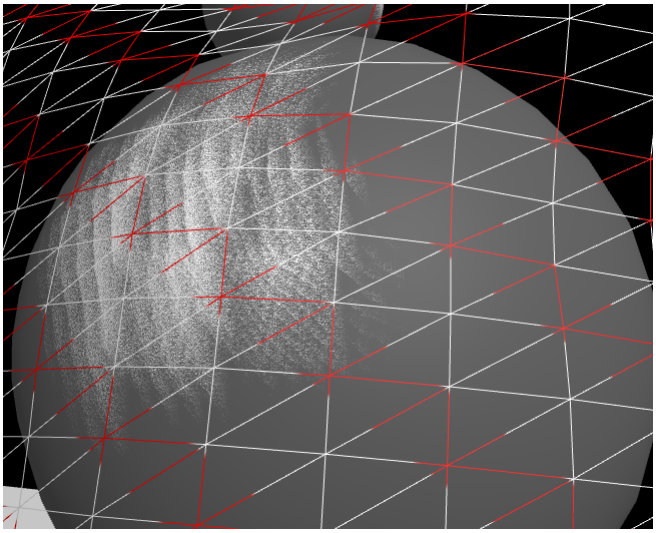
Pro implementaci vykreslování je zvolena technologie OpenGL. Způsob vykreslování je společný všem použitým implementacím simulace. Jedinou drobnou odlišností mezi CPU implementacemi a CUDA implementacemi je uspořádání dat v paměti. V případě CPU implementací jsou všechna data související s jedním vrcholem uložena v jednom bufferu v pořadí: pozice, normálový vektor a texturovací souřadnice. V případě CUDA implementací je pro každou složku vyhrazen samostatný buffer. Tato odlišnost nemá podstatný vliv na rychlost vykreslování, ale značný dopad má na rychlost práce s těmito daty v případě simulace.

Metoda zvolená pro vykreslování se skládá ze dvou fixních a jednoho volitelného vykreslovacího průchodu simulovanou scénou. Fixní vykreslovací průchody mají na starost samotné vykreslení scény. První průchod vypočítá data využitá následně ve druhém průchodu. Tyto dva průchody vykreslí celou scénu i se stíny vrhanými objekty ve scéně. Pro výpočet stínů je nutný samostatný průchod scénou. V našem případě se jedná právě o první povinný průchod. Druhý průchod pak má následně veškeré informace nutné pro vykreslení celé scény. Třetí volitelný průchod má na starost interakci se simulovanou tkaninou. Během tohoto průchodu je tkanina vykreslena do paměti tak, že místo barvy se ukládají indexy jednotlivých vrcholů. Následně je možné tato data vyčíst z paměti a tím určit, na který z vrcholů bylo kliknuto. Detailnějšímu popisu jednotlivých průchodů bude věnován zbytek této podkapitoly.

První průchod vypočítá data nutná pro vykreslení stínů. Pro výpočet stínů existuje mnoho různých metod, jako jsou na příklad stínová tělesa, nicméně v rámci této práce byly implementovány stínové mapy. Základní princip stínových map je poměrně jednoduchý. Celý princip je založen na vykreslení scény z pohledu světla. Při tomto vykreslování není brán v potaz žádný osvětlovací model ani texturování. Jediným vstupem tohoto kroku jsou pozice vrcholů. Výstupem je hloubka vykreslovaného pixelu, která je uložena do stínové mapy. Takto vytvořená stínová mapa je následně použita v druhém průchodu. Aby bylo možné použít výstup tohoto kroku, je nutné výstup zapisovat do textury. Bohužel stínové

mapy trpí několika zásadními problémy [2].

Největším problémem je "samozastiňování", kdy na osvětlených částech vzniká pravidelný vzor se stínem a osvětlenými částmi. Tento problém je možné celkem snadno odstranit pomocí nastavení jistého ofsetu. Použití ofsetu s sebou přináší ovšem také jisté problémy, jak je vidět na obrázku 6.1. Zde zobrazená koule, i když je ve stínu, je místy osvětlená, pokud je tkanina dostatečně blízko jejímu povrchu. Tento problém je způsoben také konečným rozlišením stínové mapy. Pokud dva objekty kolidují, dochází k nepřesnostem ve vyhodnocování hloubky. Proto mohou některé části být osvětleny, i když by neměly. Dalším problémem stínových map v jejich základní podobě je ostrý přechod mezi osvětlenými a zastíněnými částmi. Řešením je výpočet stínu za použití více vzorků ze stínové mapy v kombinaci s náhodným vzorkováním.



Obrázek 6.1: Problém se stínem.

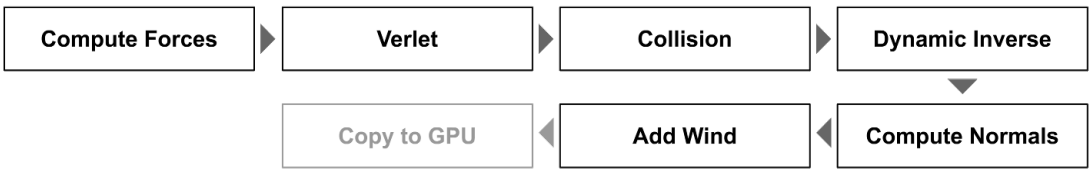
Druhý průchod vykresluje scénu na obrazovku. Využívá výstup prvního průchodu k určení, zda je vykreslovaný pixel osvětlen, nebo je ve stínu. Pokud je ve stínové mapě nějaký objekt blíže ke světlu než aktuálně vykreslovaný prvek, znamená to, že tento prvek je ve stínu. Kromě testu na zastínění je ve druhém průchodu počítán osvětlovací model. Ten bere v potaz zastínění vykreslovaného prvku. Dále jsou na vykreslované prvky nanášeny textury zvyšující realističnost vykreslovaných objektů. V tomto kroku je tedy nutné znát pro každý z vrcholů jeho souřadnice, normálový vektor a texturovací souřadnice.

Třetí volitelný průchod zajišťuje interakci s tkaninou tak, že je možné za simulovanou tkaninu zatáhnout. Tento průchod je volitelný, a tudíž není vždy aktivní. Využívá se pouze, pokud je vybrán nástroj pro interakci s tkaninou. V jiném případě tento průchod nemá smysl a zpomaloval by vykreslování. Výstupem třetího průchodu je tkanina vykreslená z pohledu kamery ale tak, že místo barvy se ukládá index vrcholu každého trojúhelníku. Výstup je zapisován do textury, ze které je možné tato data opět přečíst na pozici, kam bylo kliknuto myší. Tím určíme, na který trojúhelník bylo kliknuto.

6.2 CPU implementace

Čistě CPU implementace je základní implementací, ze které vycházejí všechny následující implementace. Před samotnou simulací je nejprve nutné vytvořit simulační mřížku. Tento krok je s drobnými obměnami také společný všem zde vytvořeným implementacím simulace tkanin. Simulační mřížka je vytvářena programově zadáním čtverce, který reprezentuje vnější rozměry simulované tkaniny. Dále pak zadáním počtu bodů n, který reprezentuje počet bodů v každém rozměru. Tím vznikne trojúhelníková mřížka s rozměry $n \times n$, která je použita jak pro vykreslování, tak pro samotnou simulaci. V průběhu vytváření simulační mřížky jsou buffery naplněny pozicemi vrcholů jednotlivých trojúhelníků, normálovými vektory a texturovacími souřadnicemi. V případě této implementace jsou všechna data uložena v jediném bufferu za sebou tak, že vždy parametry jednoho vrcholu jsou u sebe. Způsob uložení dat v paměti se v jednotlivých implementacích liší. Další součástí simulační mřížky jsou vazby mezi jednotlivými částicemi, reprezentovanými vrcholy trojúhelníkové sítě. Tyto vazby nebo také pružiny udržují tvar tkaniny. Pružiny jsou uloženy v samostatném bufferu a jejich vlastnostmi jsou indexy bodů, které spojují, koeficienty pružnosti a tlumení a délka v klidovém stavu. V této implementaci jsou použity všechny typy pružin uvedených v kapitole o částicových systémech. Na závěr vytváření simulační mřížky je alokována paměť pro uložení sil působících na jednotlivé částice a paměť pro normály jednotlivých trojúhelníků. Simulační mřížka je takto vytvořena při startu programu nebo je přegenerována při každé změně jejích rozměrů.

Pokud je mřížka úspěšně vytvořena, může být použita pro simulaci. Simulace probíhá v několika krocích, které jsou vyobrazeny na obrázku 6.2. Jsou prováděny v pořadí, jaké je možno vidět na obrázku s tím, že krok Add Wind je volitelný a může být přeskočen. Vzhledem k tomu, že se jedná o CPU implementaci, je na obrázku také krok přenosu dat na grafickou kartu. Ten samozřejmě v CUDA implementaci není. Nicméně obrázek ukazuje základní princip simulace, který je využit i v dalších implementacích. Rozdílem je, že některé kroky mohou být sloučeny nebo rozděleny do více kroků. Další text bude věnován právě popisu jednotlivých kroků.



Obrázek 6.2: Simulace.

Compute Forces je krok, kde jsou vypočítány jednotlivé síly působící na každou jednotlivou částici. Výsledné síly jsou reprezentovány jako vektory a jsou součtem všech sil. Konkrétně gravitační síly, rychlosti, na kterou je aplikováno tlumení, a síly pružin spojujících jednotlivé částice. Síly vyvíjené pružinami slouží k tomu, aby se udržovala víceméně stále stejná vzdálenost mezi navzájem spojenými částicemi.

Krok Verlet počítá numerickou integraci celého systému dle vzorců uvedených v části o částicových systémech. V rámci tohoto kroku je také řešena kolize tkaniny s podlahou následujícím způsobem. Pokud by nová pozice částice měla být menší, než je pozice podlahy, je poloha této částice nastavena tak, aby ležela na podlaze.

Další částí simulace je část Collision, jež vyhodnocuje kolize tkaniny s koulemi. Detekce kolize probíhá takto. Pokud jsou souřadnice částice uvnitř testované koule, tak je tato částice

označena za kolidující. Reakce na zjištěnou kolizi následně vypočítá místo na povrchu koule, kam je kolidující částice posunuta.

Následující krok Dynamic Inverse zajišťuje to, že pružiny nejsou moc natažené a tím zvyšuje reálnost celé simulace. Po tomto kroku jsou již známy konečné polohy všech částic a je možné dopočítat další vlastnosti jednotlivých částic přímo závislých na jejich polohách.

Takovou vlastností jsou normálové vektory jednotlivých částic. Normály jsou počítány v kroku jménem Compute Normals. Nejprve je nutné vypočítat normálové vektory jednotlivých trojúhelníků. Takto vypočítané normály jsou uloženy pro případné použití v následujícím kroku. Normálové vektory jednotlivých částic jsou vypočítány jako suma normálových vektorů trojúhelníků, jejichž součástí je zpracovávaná částice.

Posledním výpočetním krokem je krok Add Wind. Pokud je zapnuto působení větru, je právě v tomto kroku vypočítána síla, která působí na jednotlivé částice. Síla větru je stanovena jako skalární součin normálového vektoru trojúhelníku, jehož součástí je zpracovávaná částice a vektoru reprezentujícího vítr. Tímto způsobem vypočítaná síla je pak připočítána k ostatním silám.

Na závěr je nutné nově vypočítané pozice částic a jejich normálové vektory přenést na grafickou kartu, aby mohly být vykresleny. Datový přenos mezi CPU, kde probíhá simulace, a grafickou kartou po každém simulačním kroku zvyšuje výpočetní čas. Z tohoto důvodu jsou mnohem lepší řešení, kde i výpočet probíhá na grafické kartě.

6.2.1 Optimalizace

Simulace chování tkanin implementovaná v předchozí kapitole používala pro svůj běh pouze jedno vlákno. Vzhledem k tomu, že simulace tkanin je vysoce paralelní problém, není takovýto přístup vhodný. V současnosti, kdy většina běžně dostupných procesorů má více jader a umožňuje běh více vláken, je vhodné problémy, jakými je i simulace tkanin, paralelizovat. Metod, jak nějaký problém paralelizovat, je mnoho. Některé jsou složitější, jiné jednodušší. Technologie OpenMP, použitá v této práci, byla zvolena pro svoji jednoduchost. Umožňuje totiž paralelizovat libovolný sériový kód.

Optimalizovaná verze vychází z předchozí CPU implementace a na procesu simulace samém se nic nemění. Simulace tedy probíhá stále tak, jak je zobrazena na obrázku 6.2. Každý krok simulace zpracovává data v cyklu. Počet opakování cyklu se mění krok od kroku. Nicméně data zpacovávaná v jednotlivých cyklech jsou na sobě nezávislá. Zde lze právě s výhodou použít technologii OpenMP. Jednotlivé cykly lze paralelizovat přidáním direktiv pro překladač před každý cyklus. Přidání direktiv je, kromě nastavení OpenMP, jediný nutný krok pro paralelizaci kódu.

Zrychlení provádění kódu, získané touto úpravou, závisí na použitém překladači a samozřejmě na možnostech procesoru, na kterém je kód spuštěn. Nicméně výkonnostní nárůst je značný v porovnání s jednoduchostí úprav sériového kódu.

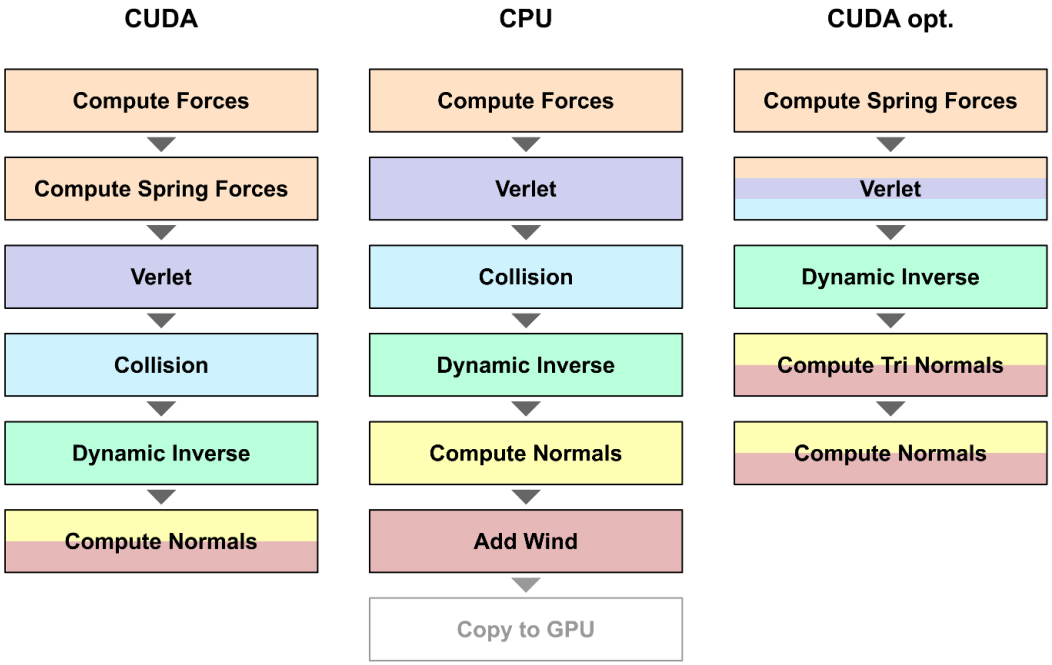
6.3 CUDA implementace

Implementace diskutovaná v rámci této kapitoly vychází ze základů položených v průběhu implementace CPU verzí. Nejprve je probrána verze, která vznikla přímým přepisem CPU implementace do technologie CUDA. Následně byla CUDA implementace optimalizována tak, aby co nejvíce využila možnosti dané technologie. Popisu optimalizace je věnována samostatná podkapitola.

V průběhu přepisu CPU kódu do technologie CUDA bylo nutné některé funkce rozdělit do více funkcí, jiné naopak sloučit. Konečné rozdělení funkčnosti mezi jednotlivé funkce neboli kernely v terminologii CUDA je možné nalézt na obrázku 6.3. V prostřední části tohoto obrázku je znázorněna posloupnost kroků v případě CPU implementace. Jednotlivé funkce jsou barevně rozlišeny. V levé části obrázku je zobrazena posloupnost kroku simulace přepsané do CUDA kódu. Kroky jsou opět barevně označeny a reprezentují jednotlivé kernely. Barva určuje, jaká funkčnost je v jednotlivých kernelech vykonávána vzhledem k CPU implementaci. Takže kernely Compute forces a Compute Spring Forces vykonávají stejnou funkčnost jako funkce Compute Forces v CPU implementaci. Naopak kernel Compute Normals z CUDA implementace vykonává stejnou činnost jako funkce Compute Normals a Add Wind. Ostatní funkce byly převedeny jedna k jedné na kernely.

Důvodem pro rozdělení funkce Compute Forces do dvou kernelů je fakt, že v rámci funkce Compute Forces jsou prováděny dva cykly, které zpracovávají různě velké množiny dat. První cyklus prochází všechny částice a počítá síly na ně působící. Druhý cyklus jde přes všechny pružiny. Přidává k jednotlivým částicím síly, kterými na ně právě tyto pružiny působí. Vzhledem k tomu, že CUDA konfigurace jednotlivých kernelů je ovlivněna množstvím dat, které zpracovávají, bylo nutné tyto dva cykly rozdělit do samostatných kernelů.

Sloučení funkcí Compute Normals a Add Wind bylo zapříčiněno podobností ve výpočtu normál a síly větru působícího na jednotlivé částice. Obě funkce potřebují pro výpočet znát normály jednotlivých trojúhelníků. V CPU implementaci byly normály trojúhelníků spočítány ve funkci Compute Normals, kde byly také využity pro výpočet normál jednotlivých částic. Normály trojúhelníků byly ale také uloženy a následně použity ve funkci Add Wind. Vzhledem k výše uvedeným skutečnostem byly tyto dvě funkce sloučeny do jednoho kernelu. Proto již není nutné uchovávat všechny normály trojúhelníků pro další užití.



Obrázek 6.3: Simulace.

6.3.1 Optimalizace

CUDA implementace vzniklá přímým přepisem CPU kódu není z pohledu výkonu příliš dobrá. Taková implementace je zatížena některými specifickými problémy, které souvisí jak s technologií CUDA samotnou, tak i s paralelním zpracováním dat. Problémem spojeným s paralelním zpracováním je například nutnost využití atomických operací u kernelu Compute Spring Forces. Kernel zpracovává data pružin, která jsou reprezentována sadou konstant jednotlivých pružin. Pružiny jsou ale hlavně reprezentovány pomocí dvojic částic, které spojují. K těmto částicím je nutné připočítat sílu působící pružiny. V tomto okamžiku je nutné použít atomický součet, protože částice je většinou spojena více pružinami. Z důvodu nutnosti použití atomických operací je kernel Compute Spring Forces ponechán samostatně, jak je vidět na obrázku 6.3 pravá část. Dalším důvodem, proč nebyl tento kernel sloučen s žádným jiným, je rozdílná velikost zpracovávaných dat. Na druhou stranu kernel Compute Forces nevyžaduje použití atomických operací, a byl proto sloučen s kernelem Verlet. V kernelu Verlet jsou tedy vypočítány všechny síly působící na částici a výsledná síla je rovnou použita pro integraci. Tím oproti neoptimalizované verzi odpadá jedna paměťová operace s globální pamětí. Síly pružin vypočítané v předchozím kroku vstupují do tohoto kernelu a jsou započítány jako další síla.

Funkce Collision, jak je vidět na obrázku 6.3, je také sloučena do kernelu Verlet. Tímto sloučením opět odpadly některé paměťové operace s globální pamětí. Nová pozice částice, zjištěná numerickou integrací, je zapsána do výstupního bufferu pouze tehdy, pokud nekoliduje s žádným objektem. V případě kolize je kolize vyřešena. Až tato nová pozice částice je zapsána na výstup.

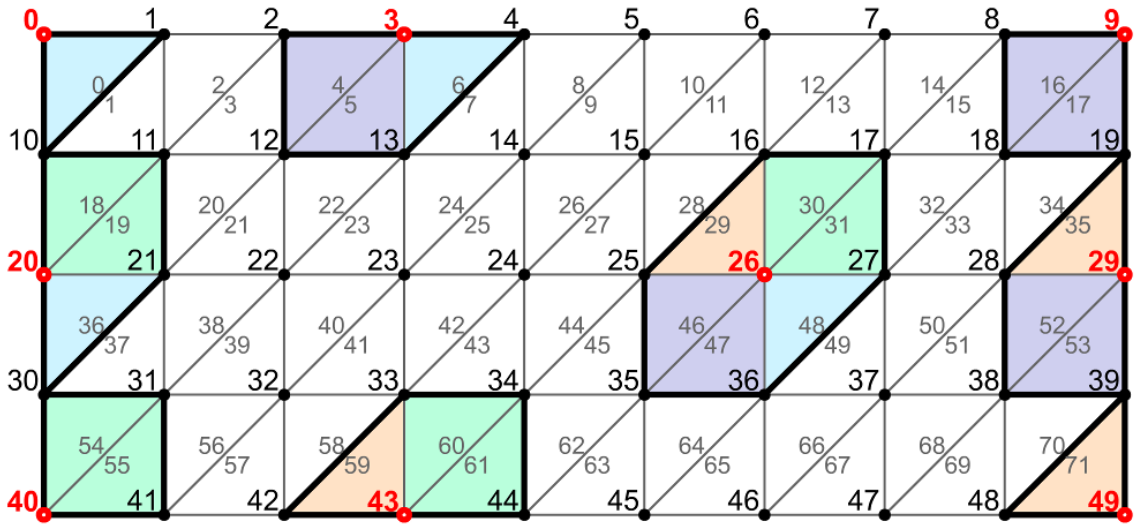
Další kernel Dynamic Inverse by mohl být sloučen s kernelem Compute Spring Forces, protože zpracovává stejně velké sady dat. Bohužel kernel Dynamic Inverse musí být až za kernelem Verlet. Rovněž ho není možné sloučit s žádným následujícím kernelem z důvodu odlišně velké sady dat.

Poslední dva kernely spolu úzce souvisí. Jmenovitě se jedná o kernely Compute Tri Normals a Compute Normals. Pro výpočet normál jednotlivých částic a síly větru je nutné znát normály jednotlivých trojúhelníků. V neoptimalizované verzi byly normály trojúhelníků vypočítány v kernelu Compute Forces a následně použity pro výpočet normál částic a síly větru. Takovýto přístup byl výhodný, protože nevyžadoval ukládání normál trojúhelníků. Nicméně opět vyžadoval použití atomických operací. Kernel byl totiž spouštěn pro každý trojúhelník. Jeho normála byla připočtena k normálám částic, které byly součástí daného trojúhelníku. Zde právě bylo opět potřeba využít atomické operace. Optimalizovaná verze problém řeší tak, že normály trojúhelníků jsou předem vypočítány v kernelu Compute Tri Normals. Normály z tohoto kernelu jsou pak použity v kernelu Compute Normals. Způsob výpočtu jednotlivých normál částic z předem vypočítaných normál trojúhelníků je na obrázku 6.4. Kernel Compute Normals je spuštěn pro každou jednotlivou částici. Výsledná normála částice je vypočítána jako suma normál okolních trojúhelníků. V případě této implementace bylo nutné vyřešit okrajové podmínky. Normála částice s indexem nula se například rovná přímo normále trojúhelníku nula. Podobně je tomu i u ostatních částic ležících v rozích simulované tkaniny. Vyřešením těchto čtyř okrajových podmínek je vyřešen celý algoritmus výpočtu normál. Normálu částice číslo tři je totiž možné vypočítat jako kombinaci podmínek, které řeší částici číslo nula a částici číslo děvet. Podobně je tomu i u ostatních částic, viz. obrázek 6.4.

Další optimalizace je využití textur pro čtení některých dat. Čtení dat pomocí textur má opět výkonnostní benefity. Jako vždy ale s výhodami přichází i nevýhody. Textury je

možné použít pouze pro čtení dat. Nicméně v simulaci tkanin jsou data, která se pouze čtou. Taková data jsou data jednotlivých pružin. Všechny parametry pružin jsou vygenerovány při startu programu a v rámci simulace se pouze čtou. Další nevýhodou textur je poněkud nestandardní zacházení s texturami z pohledu programátora. Textury jsou chápány jako globální objekty. Je nutné je před každým použitím připojit a následně zase odpojit. Připojování a odpojování textur má jisté dopady na výkon. Výhody použití textur ale většinu nevýhod převáží.

Možností jak získat nějaký výkon navíc je použití konstantní paměti. Konstantní paměť má opět nějaká svá omezení. Podobně jako textury je ji možné použít pouze pro čtení. Navíc má omezenou velikost. Ovšem v simulaci chování tkanin se najdou data, která mohou těžit z výhod konstantní paměti. Parametry koulí, se kterými tkanina koliduje, je možné s výhodou přenést do tohoto typu paměti.



Obrázek 6.4: Výpočet normálových vektorů.

Kapitola 7

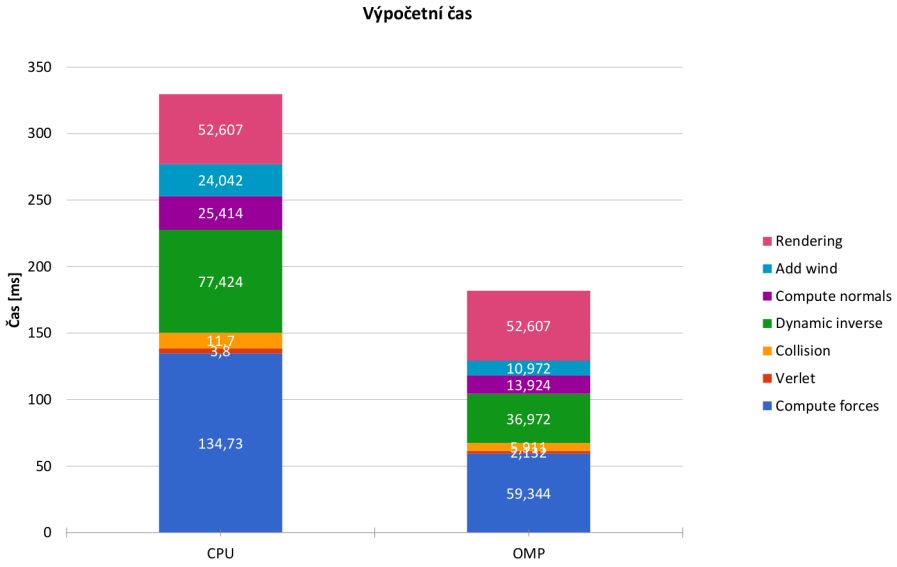
Výkonnostní porovnání jednotlivých implementací

V této části práce byly jednotlivé implementace změřeny a porovnány z pohledu jejich výpočetní náročnosti. Měření byla provedena na dvou počítačích. Konkrétně se jednalo o notebook osazený procesorem Intel Core i5-4200U s frekvencí 1.60GHz, RAM pamětí o kapacitě 4 GB a grafickou kartou Nvidia GeForce GT 730M. Druhým testovacím počítačem byl stolní počítač s následující konfigurací - procesor Intel Core i5-3350P s taktem 3.10GHz, RAM paměť s 8 GB a grafickou kartou Nvidia GeForce GT 640.

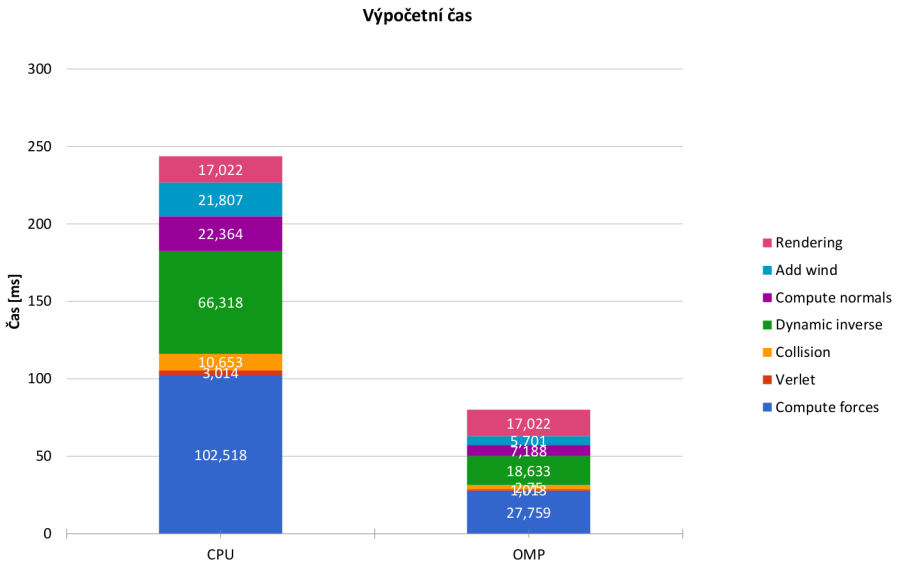
Měření probíhala na obou počítačích stejným způsobem. Notebook byl po celou dobu měření připojen do elektrické sítě, aby nebyl výkon hardwaru snížen. Na žádném z testovacích počítačů v době testu nebyl spuštěn žádný jiný uživatelský program. Jednotlivá měření probíhala tak, že vždy bylo provedeno tisíc iterací simulace pro každou měřenou sekci kódu. Velikost simulované tkaniny byla nastavena na 500×500 částic. Čas potřebný na provedení dané sekce kódu byl následně zapsán do výstupního souboru. Tímto způsobem bylo získáno tisíc výsledků pro každou sekci, což snížilo nepřesnosti vzniklé chybami měření. Výsledky těchto měření jsou v následujících podkapitolách uvedeny ve formě agregovaných dat a prezentovány pomocí grafů. Kompletní tabulky všech naměřených dat je možné nalézt na CD, které je k této práci přiloženo.

7.1 CPU implementace

Tato podkapitola je věnována popisu měření CPU implementací. Měření probíhala po jednotlivých funkcích, jež jsou zodpovědné za simulaci. Dále byl změřen i čas potřebný na vykreslení jednoho snímku. Aktivní byly všechny tři vykreslovací průchody a data byla před vykreslením přenášena na grafickou kartu. Výsledky měření je možné nalézt na obrázku 7.1 v případě notebooku a na obrázku 7.2 pro stolní počítač. Z grafů je patrné, že OpenMP implementace je až dvakrát rychlejší než CPU implementace v případě notebooku a až čtyřikrát rychlejší v případě stolního počítače. Výsledky těchto měření potrzují domněnku, že OpenMP implementace je rychlejší než čistě CPU implementace. Jedinou částí, kterou nebylo možné pomocí OpenMP urychlit, je část vykreslování. Jedná se totiž o čistě OpenGL volání, jež nelze pomocí OpenMP paralelizovat. Velký podíl času si vyžádalo také přenášení simulačních dat na grafickou kartu.



Obrázek 7.1: Porovnání CPU a OpenMP implementace (měřeno na notebooku).

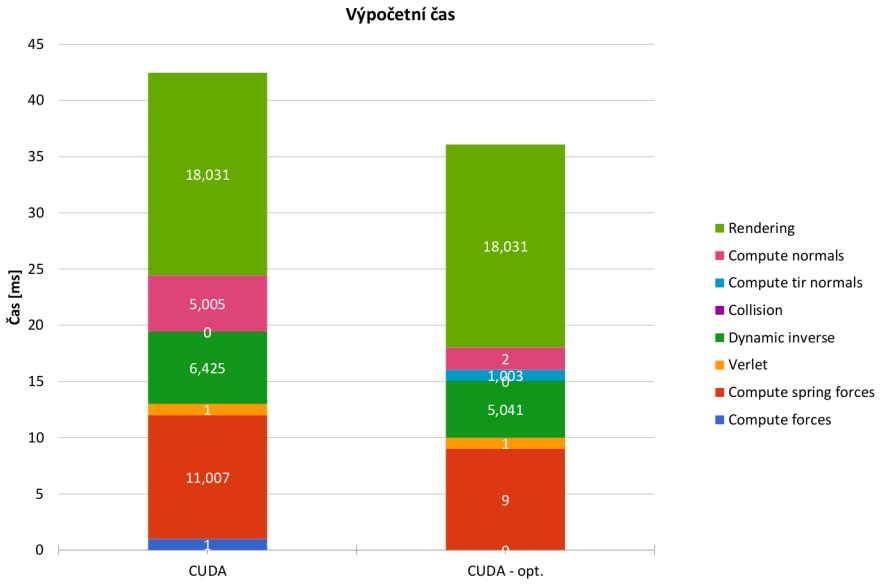


Obrázek 7.2: Porovnání CPU a OpenMP implementace (měřeno na stolním počítači).

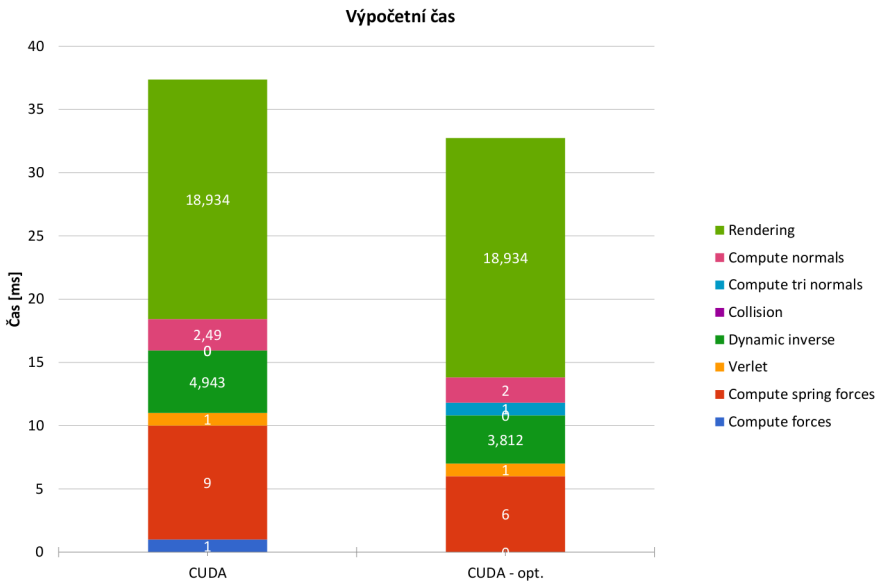
7.2 CUDA implementace

Měření výpočetní náročnosti CUDA implementací probíhalo podobným způsobem jako měření CPU implementací. Byl tedy změřen výpočetní čas potřebný k provedení jednotlivých kernelů. Také byl změřen čas potřebný k vykreslení jednoho snímku. Opět byly aktivní všechny tři vykreslovací průchody. V tomto případě ale nebylo nutné simulační data přenášet na grafickou kartu k vykreslení. Výsledky jednotlivých měření je možné vidět na obrázku

7.3 pro notebook. Na obrázku 7.4 jsou prezentována data získaná na stolním počítači. Porovnáním těchto grafů zjistíme, že již nedošlo k tak výraznému snížení výpočetního času. Nicméně optimalizovaná CUDA implementace je stále schopna vykreslit o čtyři snímky za sekundu víc než implementace bez jakýkoliv optimalizací.



Obrázek 7.3: Porovnání CUDA implementací (měřeno na notebooku).

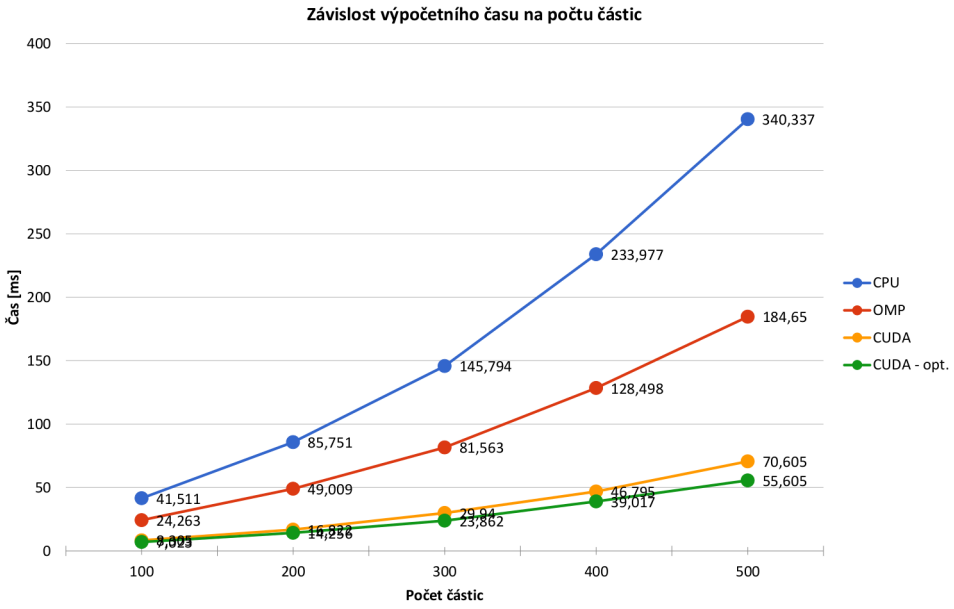


Obrázek 7.4: Porovnání CUDA implementací (měřeno na stolním počítači).

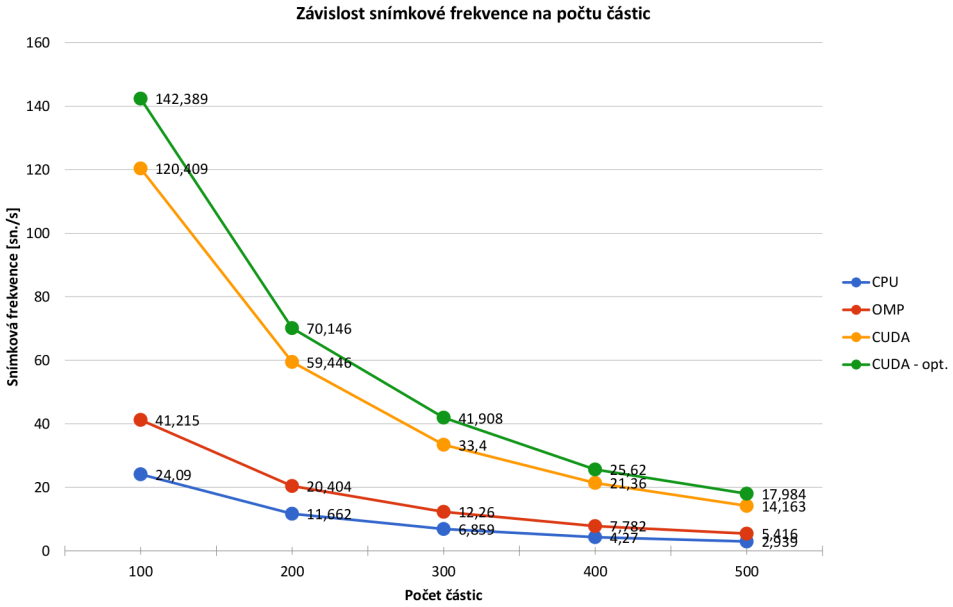
7.3 Zhodnocení

Měření prezentovaná v této kapitole probíhala trochu odlišně od předchozích měření. Místo měření jednotlivých funkcí byla měřena kompletní simulace včetně jejího vykreslení. Největší změnou bylo ale nastavení velikosti simulované tkaniny. Byla provedena sada měření s tím, že velikost tkaniny byla nastavena postupně na 100×100, 200×200, 300×300, 400×400 a 500×500 částic. Tyto sady měření byly provedeny pro všechny čtyři implementace. Na následujících obrázcích 7.5 a 7.7, jsou získaná data prezentována jako závislost výpočetního času na počtu částic. Na obrázcích 7.6 a 7.8 jsou data prezentována jako závislost snímkové frekvence na počtu částic.

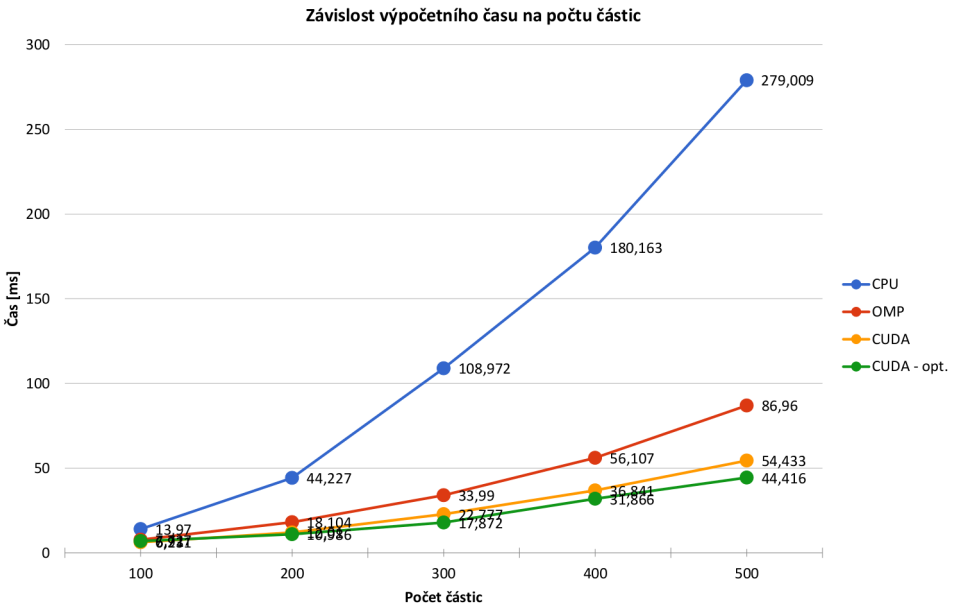
Z obrázků je patrné, že nejhorších výsledků dosahuje čistě CPU implementace. Důvodem takto špatných výsledků je to, že celý algoritmus je prováděn v jednom vlákně. Se zvyšujícím se počtem vláken se zvyšuje i rychlost vykonávání algoritmu. OpenMP implementace, kde například na stolním počítači výpočet probíhal ve čtyřech vláknech, dosahuje mnohem lepších výsledků. Velký skok ve výkonu je ale mezi CPU a CUDA implementacemi. CUDA implementace bez jakýkoliv optimalizací dosahuje výrazně lepších výsledků. To je právě dáno tím, že CUDA implementace běží v desítkách paralelních vláknech. Další výkon byl získán optimalizací CUDA kódu. Tato implementace již nezískává výpočetní výkon pomocí více vláken. V průběhu optimalizace byla totiž vylepšena práce s pamětí. Paměťové operace jsou úzkým hrdlem většiny programů. Stejně je tomu i u simulace chování tkanin.



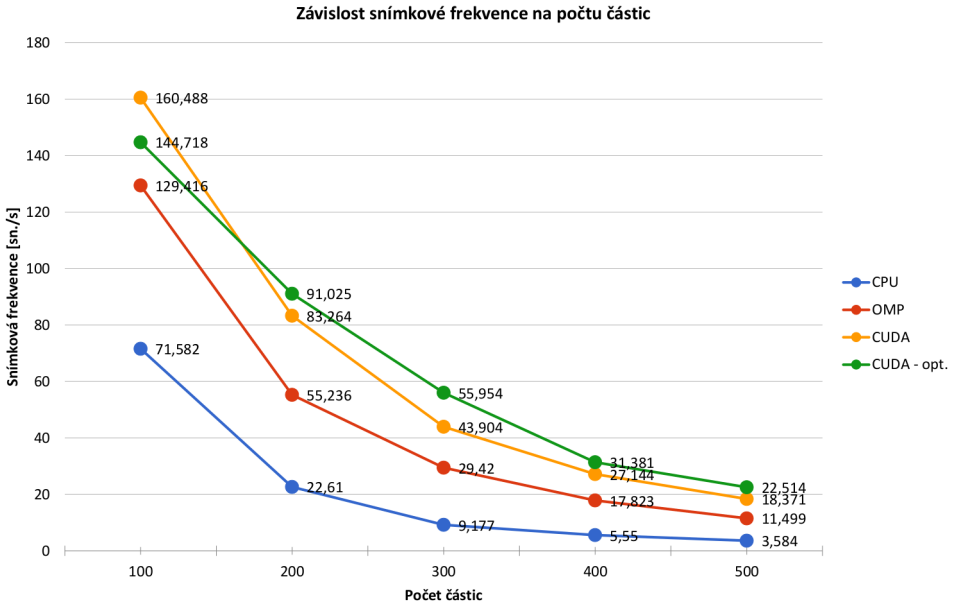
Obrázek 7.5: Závislost výpočetního času na počtu částic (měřeno na notebooku).



Obrázek 7.6: Závislost snímkové frekvence na počtu částic (měřeno na notebooku).



Obrázek 7.7: Závislost výpočetního času na počtu částic (měřeno na stolním počítači).



Obrázek 7.8: Závislost snímkové frekvence na počtu částic (měřeno na stolním počítači).

Kapitola 8

Závěr

Tato práce se zabývá simulací chování tkanin. Jejím výstupem je program prezentující možnosti simulace chování tkanin počítané v reálném čase. Program umožňuje interakci se simulovanou tkaninou pomocí několika nástrojů. Se scénou, obsahující simulovanou tkaninu, lze otáčet, přibližovat ji i oddalovat. Dále je možné měnit velikost simulační mřížky a způsob jejího vykreslení. Program také obsahuje nástroje pro ovládání větru, který na tkaninu působí. V neposlední řadě je možné za simulovanou tkaninu zatáhnout.

V průběhu práce byly vytvořeny čtyři různé implementace téhož algoritmu simulace chování tkanin. Jmenovitě se jedná o čistě CPU implementaci, OpenMP implementaci, CUDA implementaci a optimalizovanou CUDA implementaci. Nad každou z těchto implementací byla provedena série měření. Měření proběhla na dvou testovacích počítačích. Konkrétně na notebooku a na stolním počítači. Na základě naměřených dat byly mezi sebou jednotlivé implementace porovnány z hlediska své výpočetní náročnosti.

Z naměřených dat vyplývá, že čistě CPU implementace je nejpomalejší. To je dáno tím, že celý výpočet probíhá v jednom vláknu. O něco lepších výsledků dosahuje OpenMP implementace. Kód OpenMP implementace je stále prováděn na CPU, ale již ve více vláknech. Největšího výkonnostního nárůstu bylo dosaženo až v CUDA implementaci. Výpočet v případě CUDA implementace již plně probíhá na grafické kartě. Paralelizmus na grafické kartě je na zcela jiné úrovni v porovnání s CPU. To je hlavním důvodem tak velkého výkonnostního nárůstu. Poslední měřenou implementací byla optimalizovaná CUDA implementace. Zde již nárůst výkonu oproti CUDA implementaci nebyl tak značný. Výsledky těchto měření dokazují předpoklad, že čistě CPU implementace bude nejpomalejší a optimalizovaná CUDA implementace nejrychlejší. Tento rozdíl je dán právě počtem vláken, ve kterých je program prováděn. CUDA implementace je tedy nejvhodnější pro použití v grafice počítané v reálném čase.

Rozšířením této práce by mohlo být zvýšení výkonu optimalizací algoritmu simulace. Jedním z možných vylepšení by bylo použití lepšího algoritmu pro detekci a řešení kolizí. Takovým algoritmem by mohl být algoritmus používající stromovou hierarchii obalových těles ke snížení počtu testovaných dat. Zvýšení výkonu optimalizací algoritmu se samozřejmě netýká pouze CUDA implementace. Takto by šel zvýšit výkon i CPU implementace. Ovšem i takto optimalizovaný CPU kód by stále nebyl vhodný pro seriózní využití v grafice počítané v reálném čase. Po vizuální stránce by bylo možné výsledek simulace vylepšit například použitím techniky bumb mapping. Tato technika umožňuje vytvořit iluzi nerovností povrchu beze změny geometrie daného tělesa.

Literatura

[1] NVIDIA Next Generation CUDA Compute Architecture: Fermi. online, 2009 [cit. 2015-01-03].
URL http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf

[2] Shadow mapping. online, 2012 [cit. 2016-04-01].
URL http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-16-shadow-mapping

[3] OpenGL wiki. online, 2014 [cit. 2015-01-02].
URL https://www.opengl.org/wiki/Main_Page

[4] NVIDIA Next Generation CUDA Compute Architecture: Kepler GK110 210. online, 2014 [cit. 2015-01-03].
URL http://international.download.nvidia.com/pdf/kepler/NVIDIA-Kepler-GK110-GK210-Architecture-Whitepaper.pdf

[5] CUDA C Programming Guide. online, 2014 [cit. 2015-04-01].
URL
http://docs.nvidia.com/cuda/cuda-c-programming-guide/#axzz3OAHVDTbb

[6] Introduction to Qt5 toolkit. online, 2015 [cit. 2016-04-01].
URL http://zetcode.com/gui/qt5/introduction/

[7] Qt. online, 2016 [cit. 2016-04-01].
URL http://www.qt.io

[8] Babic, K.: Cloth Modeling. online, 1999-04-28 [cit. 2015-01-06].
URL http://davis.wpi.edu/~matt/courses/cloth/

[9] Farber, R.: *CUDA Application design and development*. Morgan Kaufmann, 2011, ISBN 978-0-12-388426-8.

[10] Fuhrmann, A., Groß, C., Luckas, V.: Interactive animation of cloth including self collision detection. online, 2012 [cit. 2016-04-01].
URL http://cg.web.th-koeln.de/wp-content/uploads/2012/09/InteractiveClothAnimation.pdf

[11] Hager, G.: *Introduction to High Performance Computing for Scientists and Engineers*. CRC Press, 2011, ISBN 978-1-4398-1192-4.

[12] Jakobsen, T.: Advanced Character Physics. online, 2006 [cit. 2015-02-01].
URL http://web.archive.org/web/20070610223835/http://www.teknikus.dk/tj/gdc2001.htm

[13] Jun: What is the difference between GPU and CPU? online, 2013-05-06 [cit. 2014-12-19].
URL
http://allegroviva.com/gpu-computing/difference-between-gpu-and-cpu/

[14] Kiessling, A.: An Introduction to Parallel Programming with OpenMP. online, 2009 [cit. 2016-04-01].
URL http://www.roe.ac.uk/ifa/postgrad/pedagogy/2009_kiessling.pdf

[15] Müller, M., Stam, J., James, D., Thürey, N.: Real Time Physics. online, 2008 [cit. 2016-04-01].
URL http://matthias-mueller-fischer.ch/realtimephysics/coursenotes.pdf

[16] Provot, X.: Collision and self-collision handling in cloth model dedicated to design garments. online, [cit. 2016-04-01].
URL http://graphics.stanford.edu/courses/cs468-02-winter/Papers/Collisions_vetements.pdf

[17] Sanders, J., Kandrot, E.: *CUDA by example: An introduction to general-purpose GPU programming*. Addison-Wesley, 2011, ISBN 978-0-13-138768-3.

[18] Tišnovský, P.: Modelování a vizualizace elastických těles pomocí systému vázaných částic. online, 2003-02-06 [cit. 2015-01-06].
URL http://www.elektrorevue.cz/clanky/03006/index.htm

[19] Tišnovský, P.: Seriál Grafické karty a grafické akcelerátory. online, 2005-03-02 [cit. 2014-12-21].
URL http://www.root.cz/serialy/graficke-karty-a-graficke-akceleratory/?pi=2

[20] Tzvetomir, V.: Collision detection for cloth simulation using ray-tracing on the GPU. online, 2012 [cit. 2016-04-01].
URL https://www.researchgate.net/publication/258121755_COLLISION_DETECTION_FOR_CLOTH_SIMULATION_USING_RAY-TRACING_ON_THE_GPU

[21] Vlask: 20 let vývoje grafických karet. online, 2012-06-28 [cit. 2015-02-01].
URL http://www.cnews.cz/clanky/20-let-vyvoje-grafickych-karet-od-cga-az-po-konec-3dfx-dil-i

[22] Wilt, N.: *The CUDA handbook: A comprehensive guide to GPU programming*. Addison-Wesley, 2013, ISBN 978-0-321-80946-9.

[23] Zaorálek, L.: Úvod do technologie CUDA. online, 2009-07-20 [cit. 2015-01-03].
URL http://www.root.cz/clanky/uvod-do-technologie-cuda/

Přílohy

Seznam příloh

Příloha A

Manual

A.1 Zprovoznění programu

Program vytvořený v rámci této diplomové práce využívá technologii CUDA. Tudíž je nutné, aby počítač disponoval CUDA kompatibilním hardwarem. Pro správnou funkci programu je nutné mít nainstalovány následující programy:

- Visual Studio - dostupné na www.visualstudio.com
- Qt - dotupné na www.qt.io/download
- Visual Studio Addin - dostupný na www.qt.io/download-open-source/#section-2
- Nvidia CUDA Toolkit - dostupný na developer.nvidia.com/cuda-toolkit

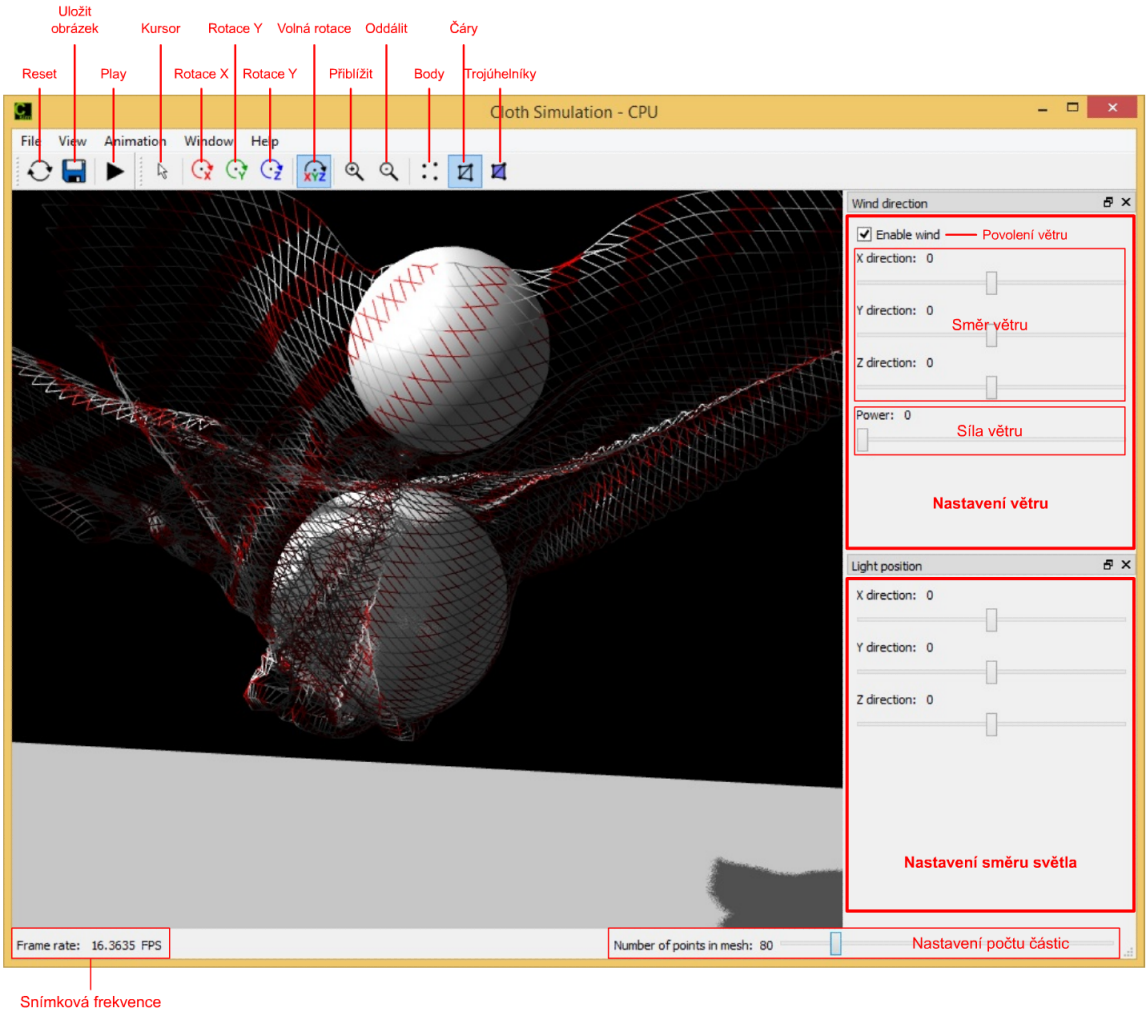
Po nainstalování všech zmíněných programů je možné otevřít Qt projekt pomocí Qt doplňku (Visual Studio Add-in) přímo ve Visual Studiu. Otevření projektu přes doplněk nastaví všechny potřebné Qt závislosti. Dále je nutné nastavit CUDA závislosti pomocí nástrojů Visual Studia. Tyto závislosti nejsou obsaženy v Qt projektu. Jakmile jsou všechny závislosti správně nastaveny, je možné program překládat obvyklým způsobem.

A.2 Ovládání programu

Uživatelské rozhraní je možné vidět na obrázku A.1, kde jsou také jednotlivé ovládací prvky popsány. Uživatelské rozhraní také obsahuje menu, jehož jednotlivé položky jsou popsány níže. Popis obsahuje jak název položky, který je použitý v uživatelském rozhraní, tak i český ekvivalent. Pokud má nástroj přiřazenu klávesovou zkratku, je uvedena v závorce za popisem. Uživatelské rozhraní dále obsahuje kontextové menu, které je možné vyvolat klikem pravým tlačítkem myši do oblasti simulace. Kontextové menu obsahuje nástroje, které jsou uvedeny v položkách Zobrazení (View) a Animace (Animation) v níže uvedeném seznamu.

- File - Soubor
  - Reset - Reset (Ctrl + R)
  - Save - Uložit obrázek (Ctrl + S)
  - Exit - Konec programu (Ctrl + Q)

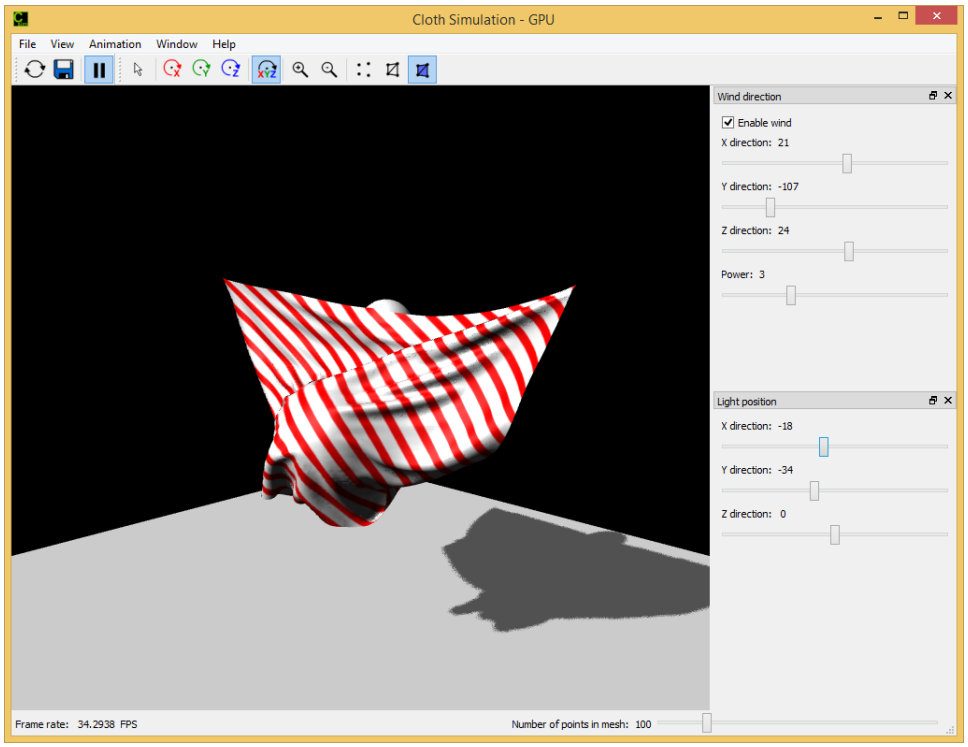
- View - Zobrazení
  - Cursor - Kursor (C)
  - Rotate X - Rotace kolem osy X (X)
  - Rotate Y - Rotace kolem osy Y (Y)
  - Rotate Z - Rotace kolem osy Z (Z)
  - Rotate free - Volná rotace kolem všech os (F)
  - Zoom in - Přiblížit (I)
  - Zoom out - Oddálit (O)
  - Show points - Zobrazit pouze body (P)
  - Show lines - Zobrazit čáry (L)
  - Show triangles - Zobrazit trojúhelníky (T)
- Animation - Animace
  - Play/Pause - Pustit/Pozastavit (mezerník)
- Window - Okno
  - Animation - Panel nástrojů animace
  - Tools - Panel nástrojů
  - Wind controller - Panel ovládání větru
  - Light controller - Panel ovládání světla
  - Status bar - Stavový řádek
- Help - Nápověda
  - About - O programu
  - About Qt - O frameworku Qt



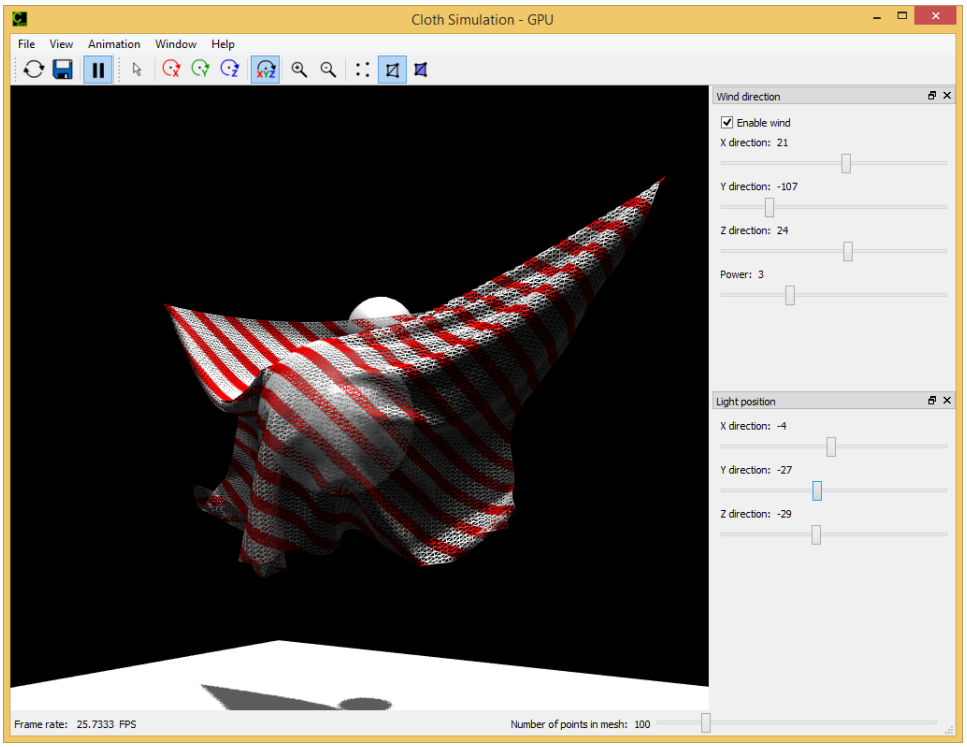
Obrázek A.1: Ovládání programu.

Příloha B

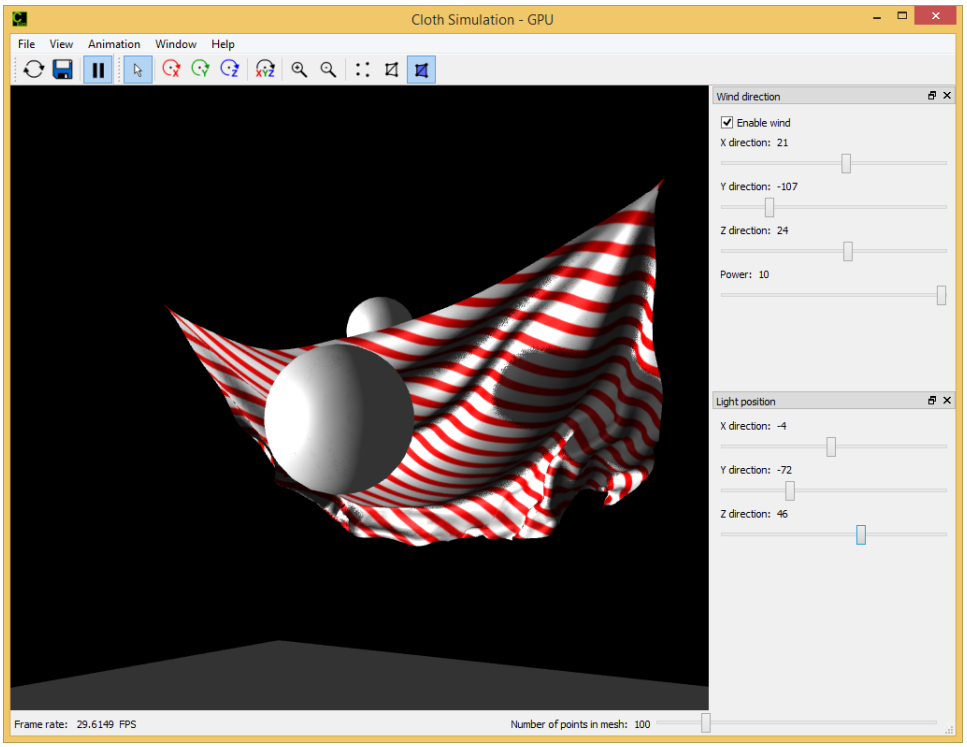
Obrázky



Obrázek B.1: Simulace chování tkanin.



Obrázek B.2: Simulace chování tkanin.



Obrázek B.3: Simulace chování tkanin.

Příloha C

Obsah CD

Kořenový adresář CD, přiloženého k této diplomové práci, obsahuje následující složky:

- Sources - zdrojové kódy
  - ClothQtCPU - CPU implementace
  - ClothQtOMP - OpenMP implementace
  - ClothQtCUNO - CUDA implementace bez optimalizací
  - ClothQtGPU - CUDA implementace s optimalizacemi
- Text - text diplomové práce
- Measurements - naměřené hodnoty
  - NB - hodnoty naměřené na notebooku
  - PC - hodnoty naměřené na stolním počítači