Semestralni projekt z predmetu A4M39GPU

Tema: Simulace satu (tkaniny)

Vypracoval: Michael Smutny (smutnmic@fel.cvut.cz)

---

1. Struktura dokumentu

Prace je rozdelena do 7 kapitol. Kapitola 1 (stana 1) popisuje strukturu dokumentu. Kapitola 2 (strana 1) obsahuje uvod, tedy co je tematem prace, proc je reseny problem zajimavy z hlediska pouzitelnosti, a nastinuje vytycene cile prace. V kapitole 3 (strana 1) je proveden teoreticky rozbor prace. Jsou zde vysvetleny metody, pomoci nichz se dany problem resi. Kapitola 4 (strana 4) se zabyva sekvencnim resenim na CPU. Je zde uveden popis trid a jejich metod. Kapitola 5 (strana 5) resi paralelni reseni na GPU. Nejprve je zde uveden postup, jak ulohu transformovat, aby se dala pocitat paralelne. Pote jsou reseny zpusoby ulozeni dat na GPU a jejich presuny mezi CPU a GPU. Na zaver kapitoly jsou nastineny i optimalizace. V kapitole 6 (strana 9) jsou uvedeny vysledky ve forme tabulky a grafu. Porovnavany jsou jak rozdily v nastaveni jednotlivych parametru, tak (coz je hlavni) zrychleni GPU verze oproti CPU verzi. Kapitola 7 (strana 11) obsahuje zaver a zhodnoceni. V priloze A (strana 13) je popsane ovladani aplikace a v priloze B (strana 14) jsou obrazky aplikace.

---

2. Uvod

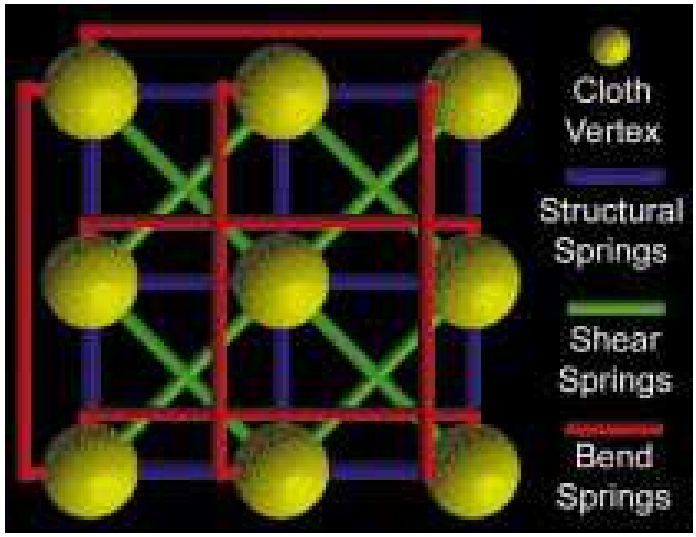
Prace se zabyva simulaci tkanin pomoci casticoveho systemu. Simulace tkanin (satu) je problem, ktery nalezne sve uplatneni ve filmovem, hernim a modelarskem prumyslu. Jedna z nejpouzivanejsich metod pro simulaci satu, je pruzinovy model (Mass spring model), ktery je pouzit i zde. Cilem prace je implementovat tuto metodu na CPU a pote navrhnout reseni, ktere bude paralelizovatelne, a ulohu implementovat I na GPU, a to pomoci jazyka CUDA. Vysledkem GPU by melo byt znatelne zrychleni oproti CPU verzi. Prace se dale zabyva moznostmi optimalizace implementace na GPU. Soucasti prace je i mereni na ruznych scenach.

---

3. Teoreticky rozbor

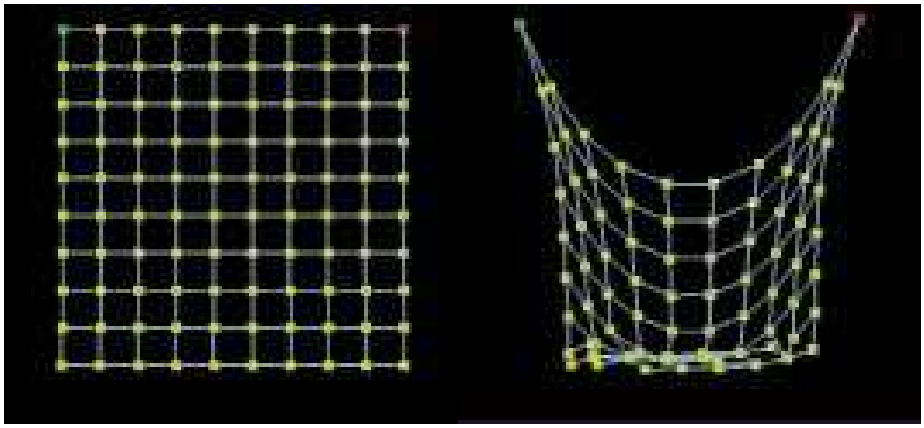
Mass spring model

Vychazel jsem ze zdroju [1], [2] a [3]. Tkanina je reprezentovana pomoci castic. Pouzil jsem ideu Mass-Spring modelu [1]. Kazda castice ma svou hmotnost (hmotnost je stejna pro vsechny castice ve scene) a vzajemne propojene castice jsou mezi sebou spojeny pruzinami. Propojeni (pruziny) jsou tri typu, jak ukazuje obrazek 1.



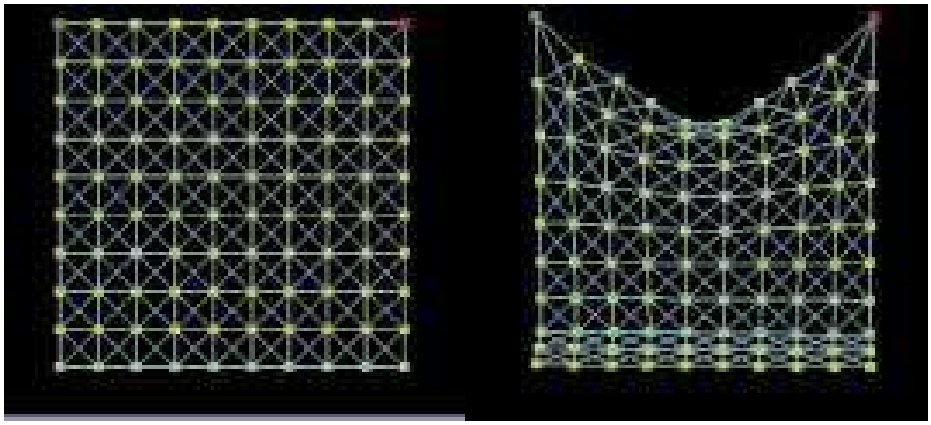
Obrazek 1: Pruzinova struktura tkaniny. (prevzato z [1])

Strukturni pruziny drzi pohromade sousedni castice a jsou to tedy zakladni pilire, bez kterych by se system rozpadl. Nicmene jen strukturni pruziny nestaci, vysledek nevypada moc realisticky. Viz obrazek 2.



Obrazek 2: Strukturni pruziny v pocatecni poloze (vlevo) a při pusobeni gravitace (vpravo). (prevzato z [1])

Z toho duvodu autor clanku pridava dalsi typ pruzin, a to tzv Shear Springs, tedy pruziny, pomoci nich tkanina lepe drzi svuj tvar. Tyto pruziny spojuji “uhlopricne sousedy”. S temito typy pruzin se da jiz vystacit (obrazek 3).



Obrazek 3: Strukturni pruziny + Shear springs v pocatecni poloze (vlevo) a při pusobeni gravitace (vpravo). (prevzato z [1])

Autor vsak pridava dalsi typ pruzin, tzv Bend Springs, ktere spojuji castice ve vzdalenosti 2 (tzn "ob souseda"). Vysledek vypada jeste trosku lepe. Hlavnim duvodem pridani techto pruzin vsak je to, ze bez nich se pri kolizi nechova tkanina zcela korektne a zpusobi to, ze castice utvori takovy "bezstrukturni chomac".

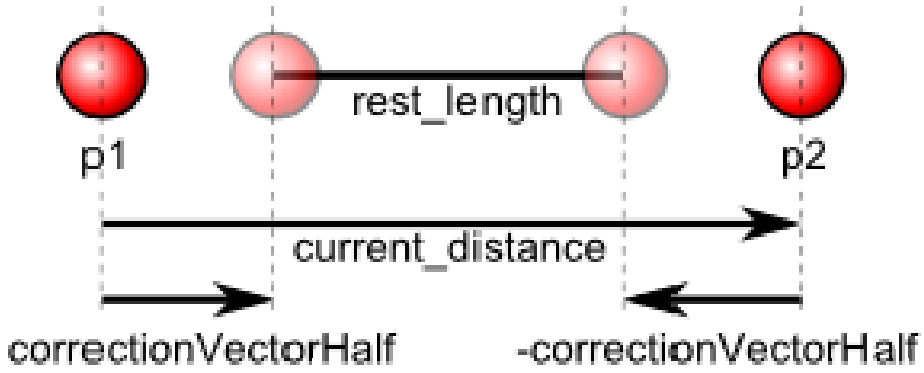
Verletova integracni metoda

Nyni, kdyz je struktura vytvorena, je potreba nejakym zpusobem vypocitat aktualni pozice castic. Z Newtonova 2. zakona dostaneme zrychleni, ktere je rovno druhe casove derivaci pozice. Cili pro stanoveni pozice ze zrychleni je potreba integrovat. Ve zdroji [2] pouzili tzv Verletovu integracni metodu [3] kvuli lepsi stabilite nez ma Eulerova metoda a i v tomto reseni je pouzita stejna metoda.

```
Vec3 temp = pos;
pos = pos + (pos-old_pos) + acceleration*TIME_STEPSIZE2
old_pos = temp;
```

Do implementace je zarazen i parametr odpor prostredi. Po jeho aplikaci se vypocitany vektor posunuti castice o dany koeficient utlumi.

Samotny vypocet nove pozice vsech castic ovsem nestaci. Vysledna pozice je jeste upravena, a to v zavislosti na vsech svych pruzinach. Je mozne urcit maximalni a minimalni delku pruziny, kterou kdyz pruzina prekroci, je delka pruziny nastavena na svou mez, nicmene lepsi je vzdy castice posunout do takove polohy, ze jejich vzdalenost bude stejna jako originalni delka pruziny (delka bez pusobeni sil – viz obrazek 4). Jelikoz kazda castice je soucasti az dvanacti pruzin (krome castic na hranici), tak v kazdem kroku "korekce vzdalenosti" se k sobe castice priblizi na vzdalenost blizsi k puvodni delce pruziny, nez tomu bylo pred timto korekcnim krokem. Cim vice techto korekcnich kroku provedeme, tim stabilnejsi bude reseni a tim mene budou castice "poskakovat".



Obrazek 4: Korekce vzdalenosti. (prevzato z [2])

---

4. CPU reseni

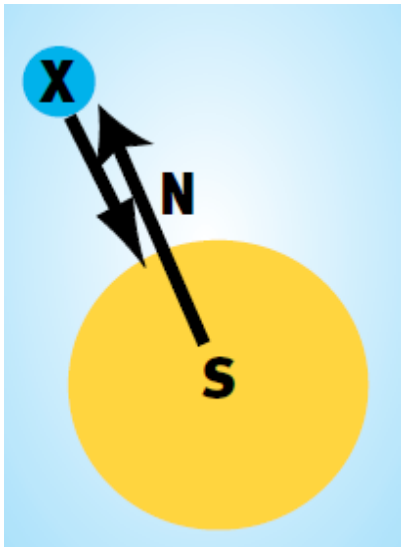
CPU verze 1

Pri seriovem reseni jsem pouzil 3 hlavni tridy, a to tridu Spring, Cloth a Particle.

Trida **Particle** predstavuje jednu castici a jejimi clenskymi daty jsou aktualni pozice, stara pozice, zrychleni, hmotnost a take informace, zda se jedna o pohyblivou castici. Nektere castice museji byt ukotveny ve scene a tudiz byt nepohyblive. Trida obsahuje take metodu pro vypocet aktualni pozice castice, tedy vyse zminenou metodu integrace.

Trida **Spring** je velice jednoducha a predstavuje pruzinu, na jejichz obou koncich je castice. Trida tedy obsahuje ukazatele na dve koncove castice a take metodu pro korekci vzdalenosti. Kvuli tomu je take nutne uchovavat v ni informaci o puvodni vzdelenosti (vzdalenosti v klidovem stavu).

Trida **Cloth** predstavuje tkaninu a jejimi clenskymi daty tedy nutne museji byt vsechny castice (std::vector) a pruziny (taktez std::vector). Trida obsahuje metody pro vykresleni satu jako trojuhelniku a i jako quadu. V konstruktoru se vyrobi vsechny castice a z prislusnych dvojic castic se vytvori pruziny. Castice v hornich rozich se museji fixovat. Trida dale obsahuje metodu addForce(), ktera vsem casticim prida silu, ktera na ne pusobi. Nejdulezitejsi metodou ovsem je metoda NextState(), ktera spocita aktualni pozice vsech castic, a to ve dvou krocich. Nejdrive je potreba spocitat integracni metodou nove pozice a pote korekcni metodou castice “vratit do rozumnych pozic”. Tato korekce se obvykle provadi vice nez jednou. Posledni metodou je metoda computeSphereCollision(), ktera zjisti, zda se nejaka castice nenachazi uprostred koule. Pokud ano, posune ji na hranici koule ve smeru normaly. Viz obrazek 5.



Obrazek 5: Posun castic ve smeru normaly na povrchu koule. (prevzato z [1])

Dalsi tridou, ktera se v projektu nachazi, je trida Camera. Jeji metoda Look() se stara o nastaveni kamery. Kod pro tuto tridu jsem prevzal ze sve prace z GSY.

CPU verze 2

Pri prechodu k GPU jsem pomerne znacne zmenil kod a zrusil jsem tridy Particle a Spring. Proto jsem se rozhodl, ze udelam druhou CPU verzi, ktera bude vychazet z GPU verze. Jen s tim rozdilem, ze misto kernelu budou “for-cykly”. Zdokumentovana (doxygen) je prave tato druha verze, nicmene nazornejsi a lepe strukturovana ta verze prvni.

---

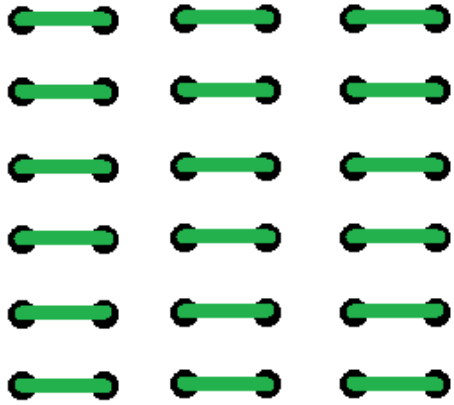
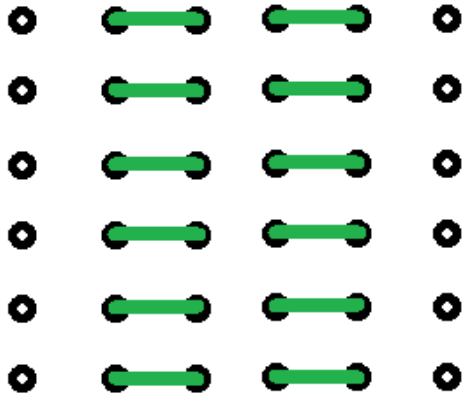
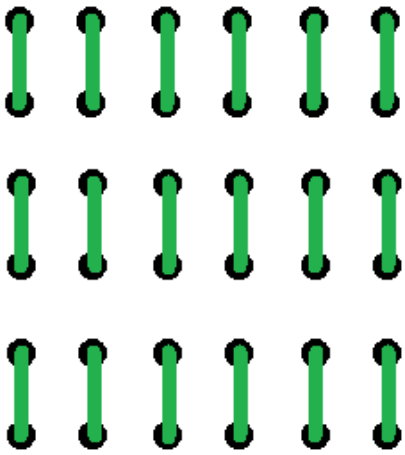
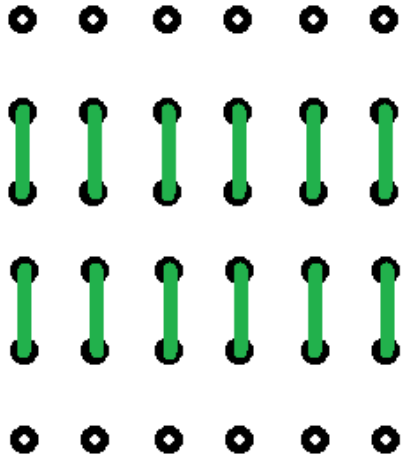
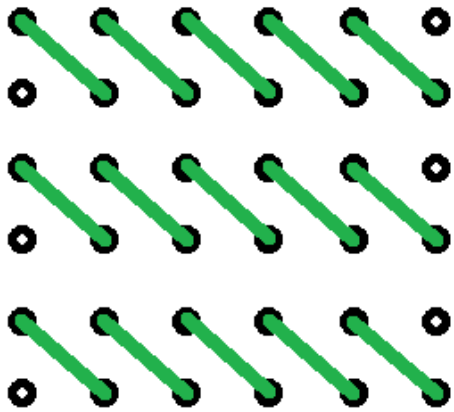
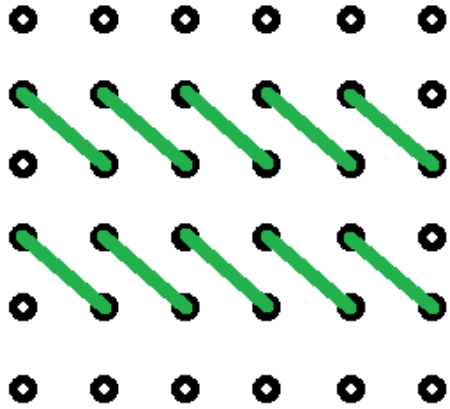
5. GPU reseni

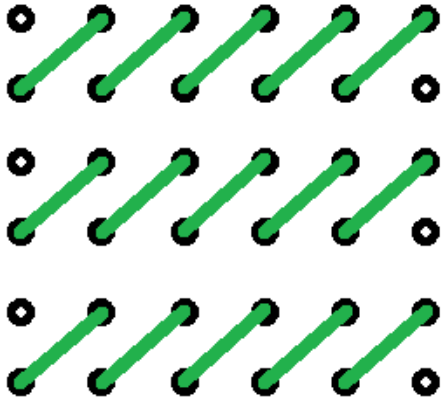
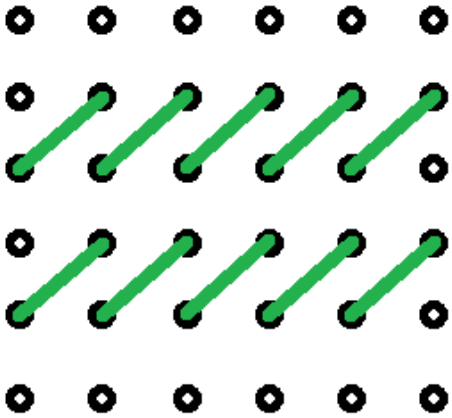
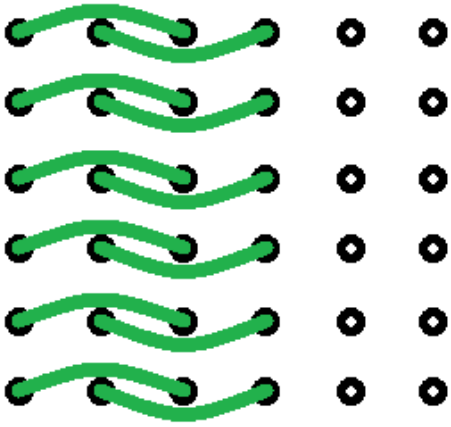
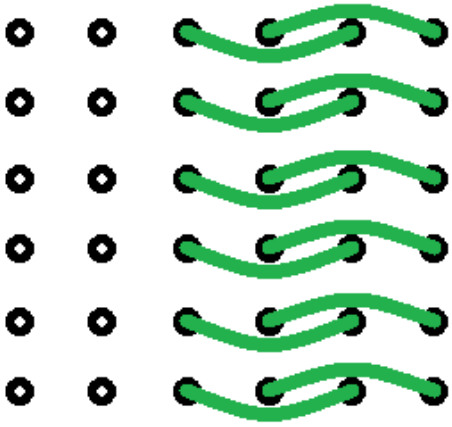
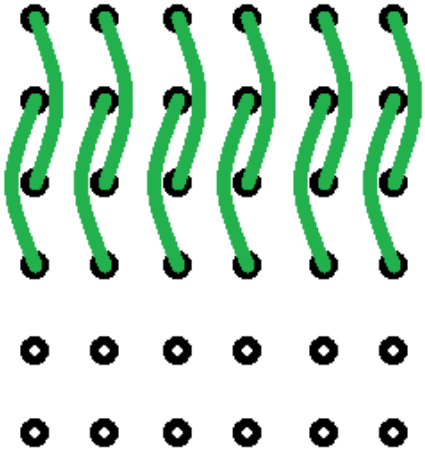
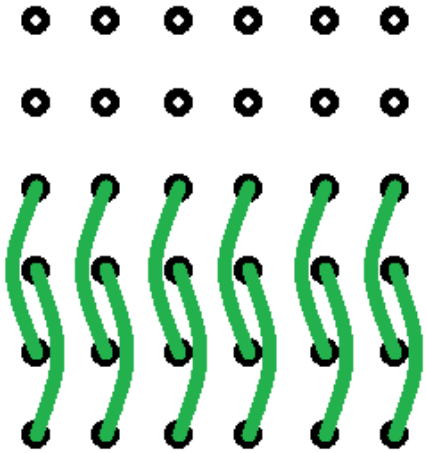
Nejprve bylo potreba zbavit se sloziteho konceptu, kdy trida Cloth obsahovala vektor castic a take vektor pruzin, ktere v sobe mely ukazatele na castice. Pro CUDA reseni bylo potreba pracovat pouze s casticemi, lepe jeste pouze s jejich pozicemi (ci jinymi atributy). Odstranil jsem tedy tridy Particle a Spring a misto nich trida Cloth nyni obsahuje pouze pole atributu castic. Jmenovite je to pole aktualnich pozic castic, pole starych pozic castic, pole zrychleni castic, pole hmotnosti castic a pole boolean informaci, zda je castice fixni. Nad temito poli se daji provadet potrebne operace, ktere jsou shodne pro kazdou castici (->priprava pro CUDA reseni). Abych nahradil i vektor pruzin, rozhodl jsem se, ze kernel funkce si sama urci, ktere dve castice budou predstavovat pruzinu.

Problem prvni – vylucne sekvencni iteracni reseni korekce vzdalenosti

To, jakym zpusobem probiha korekce vzdalenosti castic, tak v teto podobe nejde paralelizovat. V kazdem kroku se pracuje s pozici castice, ktera byla vypoctena v kroku predeslem. Je potreba to uchopit troche z jineho pohledu. Kazda castice je soucasti dvanacti ruznych pruzin. Kazda pruzina koriguje vzdalenosti castic v korekcnim kroku prave jednou. Je jasne, ze s jednou castici se v korekcnim kroku pracuje dvanactkrat, cili zcela paralelizovat tento krok nejde. Pri vhodnem rozdeleni

je mozne v jednom okamziku nad polem castic pracovat jako s pruzinami, jejichz castice se neprekryvaji. Takovychto konfiguraci je dvanact.

1) sousede v X-ove souradnici 1 	2) sousede v X-ove souradnici  2
3) sousede v Y-ove souradnici 1 	4) sousede v X-ove souradnici 2 
5) sousede uhlopricne zleva doprava 1 	6) sousede uhlopricne zleva doprava 2 

7) sousede uhlopricne zprava doleva 1 	8) sousede uhlopricne zprava doleva 2 
9) ob-sousede v X-ove souradnici 1 	10) ob-sousede v X-ove souradnici 2 
11) ob-sousede v Y-ove souradnici 1 	12) ob-sousede v X-ove souradnici 2 

Problem druhy – vsechny vypocty presunout na GPU

Aby se zabranilo zbytecnemu presouvani dat mezi GPU a CPU, tak se zezacatku inicializuji data na CPU, zkopiruji se na GPU a zde jiz probihaji vsechny vypocty. To znamena, ze krome paralelizace vypoctu pozice (integrace) a korekce vzdalenosti, jsem musel na GPU presunout take vypocty pro pridani sily vsem casticim (metoda addForceGPU()) a take vypocet kolize s kouli (computeSphereCollisionGPU()). Jedinym presunem dat tak zustava po kazdem celkovem vypoctu presun dat z GPU na CPU, aby mohla byt tkanina zobrazena na vystupni zarizeni (do spoluprace CUDA<->OpenGL jsem se nepoustel).

Problem treti – rozmery mrizky

Rozhodl jsem se, ze pocet vlaken v bloku bude 16x16. Zaroven museji byt rozmery tkaniny (pocet castic na dimenzi) nasobky 32. Pokud uzivatel zada jine cislo, je toto cislo zaokrouhleno na nejblizsi nizsi cislo 32k (kde k=1,2,…). Rozmery mrizky jsou potom dvoje, a sice:

A) (num_particles_x/32, num_particles_y/16)

B) (num_particles_x/16, num_particles_y/32)

Prvni pripad je pro konfigurace 1,2,9 a 10, druhy pripad pro ostatni konfigurace. Nazornejsi je to na predeslych 12ti obrazcich.

Jednotlive kernely

Kernely pro vypocet pozice (integrace), pro pridani sily a pro vypocet kolize s kouli jsou velice jednoduche. Kernel je zavolan pro kazdou castici a pozadovana velicina je vypoctena nezavisle na ostatnich casticich. Problematictejsi je vypocet korekce vzdalenosti. Jak jiz bylo zmineno, jedna se o 12 sekvencnich kroku (12 po sobe jdoucich volani RUZNYCH kernelu). Kazdy kernel dela toto: vypocita dva indexy do pole pozic castic a zavola funkci CorrectPair(), ktera upravi vzdalenosti. Je zde 12 kernelu pro 12 konfiguraci, kazdy kernel vypocita jine indexy.

Optimalizace

Vysledne GPU reseni je zhruba 10x rychlejsi nez CPU reseni (viz vysledky v dalsi kapitole). Zajimal jsem se take o moznost zrychleni. Uvazoval jsem tri moznosti, maximalizovat sdruzeny pristup, pouzit texturovani pamet a pouzit sdilenou pamet. Maximalizovat sdruzeny pristup bylo prakticky nemozne, protoze kazdy kernel precetl dve PREDEM URCENE hodnoty z globalni pameti. Sdruzeny pristup do pameti se tedy odvijel od navrhu konfiguraci. Ke sdruzenemu pristupu nedochazi v konfiguracich 1,2,9 a 10. V ostatnich konfiguracich k nemu dochazi (nekde castecne). Pouziti texturovaci pameti jsem zavrhl, protoze kazda hodnota z globalni pameti byla prectena (je jedno, jakym vlaknem) JEN JEDNOU, tudiz zde by CACHEovani (jedna z vyhod texturovaci pameti) nepomohlo. Pouziti sdilene pameti se jako perspektivni z toho duvodu, ze by cteni z globalni pameti mohlo byt vyhradne sdruzenym pristupem (stejne tak zapis). Jelikoz v ramci kernelu dochazi pouze k jednomu cteni z globalni pameti, tak by se tato vyhoda promenila spise v nevyhodu (dalsi zbytecna rezie). Proto jsem vyuzil rozmeru mrizky jednotlivych volani kernelu a misto do 12 skupin jsem je rozdelil do dvou skupin. Prvni skupinu tvori konfigurace 1,2,9 a 10 a maji rozmery mrizky (num_particles_x/32,

num_particles_y/16). Druha skupina jsou ty ostatni konfigurace. Idea je tedy mit pouze dva kernely, nacist do sdilene pameti potrebny kus globalni pameti, nad kterym bude provaden vypocet a jakoby provest nekolik puvodnich kernelu v jednom kernelu naraz. Samozrejme je potreba nacist do sdilene pameti vice dat, ze je potreba (data "za hranici" bloku), takze nakonec blok prvniho typu bude mit 18x16 vlaken a blok druheho typu 18x18 vlaken (obsahuje i uhlopricne pruziny).

Pozn.: Podarilo se mi implementovat pouze kernel prvniho typu, u kernelu druheho typu jsem nebyl schopen spravne spocitat indexy pri jednotlive konfigurace, coz vedlo k nekorektnimu a nestabilnimu reseni. V aplikaci je moznost prepinani mezi resenim pomoci globalni a sdilene pameti. Pokud je nastaven vypocet pomoci sdilene pameti, tak kernel prvniho typu dela vypocet konfiguraci 1,2,9 a 10, zatimco pro ostatni konfigurace jsou pouzity prislusne kernely z reseni pomoci globalni pameti. I tak je ale patrne zrychleni oproti verzi s globalni pameti.

---

6. Vysledky

Mereni probehlo na PC s konfiguraci: Core2Duo (2 x 2,9GHz), 2GB RAM, GF GTX 260 (898MB)

V aplikaci se daji nastavit ruzne parametry, ovsem ty, ktere ovlivnuji rychlost vypoctu, jsou dva. Pocet castic a pocet iteraci (pocet kroku korekce vzdalenosti). V tabulkach je zaznamenan cas vypoctu 500 snimku (vysledny objekt nebyl zobrazovan, aby zobrazovani nezkreslovalo mereni). Do tabulky je take vynesena hodnota zrychleni GPU oproti CPU verzi.

Pocet castic	32x32		500 snimku	CPU	GPU global	GPU shared
Pocet iteraci	2		cas [ms]	407	248	281
			ratio	1,0	1,6	1,4

Pocet castic	64x64		500 snimku	CPU	GPU global	GPU shared
Pocet iteraci	2		cas [ms]	1702	265	251
			ratio	1,0	6,4	6,8

Pocet castic	128x128		500 snimku	CPU	GPU global	GPU shared
Pocet iteraci	2		cas [ms]	7748	438	409
			ratio	1,0	17,7	18,9

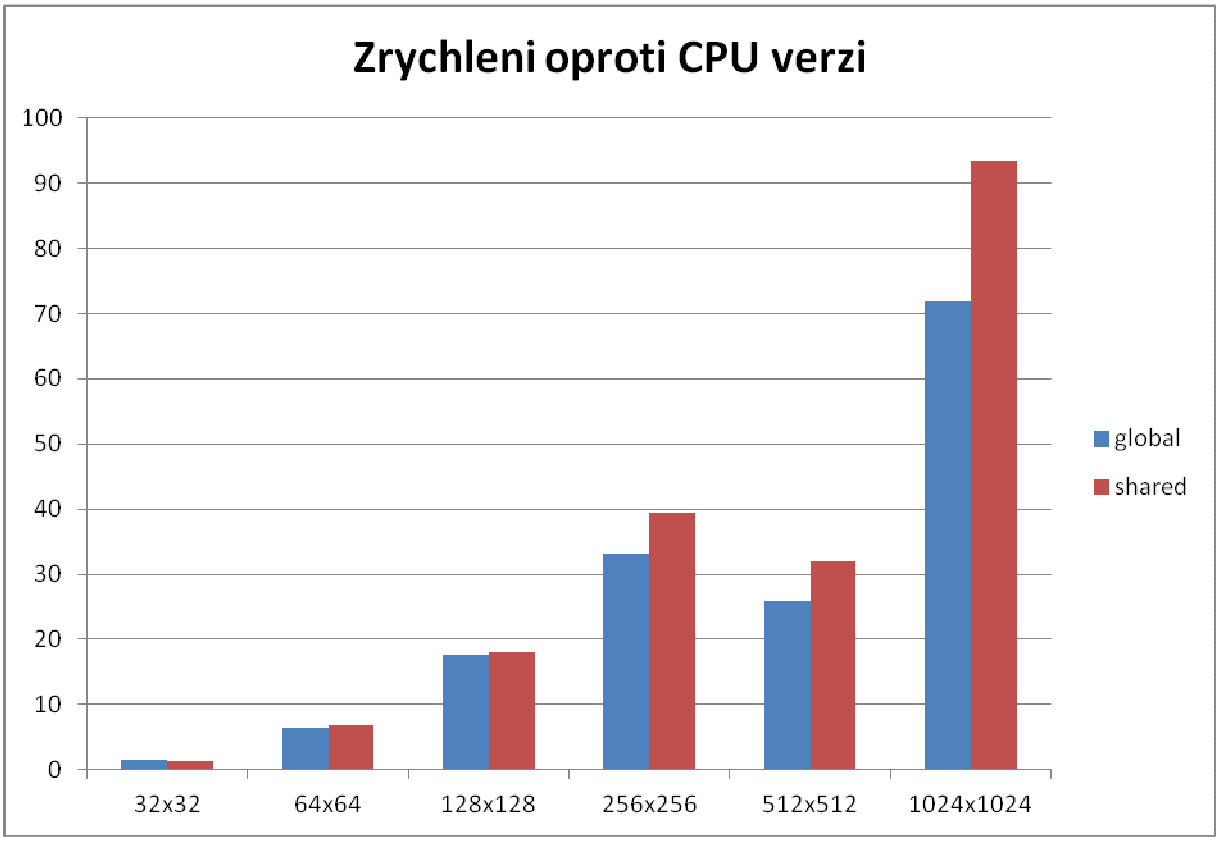
Pocet castic	256x256		500 snimku	CPU	GPU global	GPU shared
Pocet iteraci	2		cas [ms]	43110	1299	1093
			ratio	1,0	33,2	39,4

Pocet castic	512x512		500 snimku	CPU	GPU global	GPU shared
Pocet iteraci	2		cas [ms]	122000	4717	3809
			ratio	1,0	25,9	32,0

Pocet castic	1024x1024		500 snimku	CPU	GPU global	GPU shared
Pocet iteraci	2		cas [ms]	1375000	19137	14706
			ratio	1,0	71,9	93,5

Pocet castic	128x128		500 snimku	CPU	GPU global	GPU Sharp
Pocet iteraci	20		cas [ms]	70999	4404	3782
			ratio	1,0	16,1	18,8

Zrychleni						
Pocet castic	32x32	64x64	128x128	256x256	512x512	1024x1024
GPU global	1,6	6,4	17,7	33,2	25,9	71,9
GPU shared	1,4	6,8	18	39,4	32	93,5



Diskuse

Da se predpokladat, ze pocet iteraci na celkovem pomeru zrychleni (casu) nebude mit moc vliv. To se take potvrdilo (viz priklad pro 128x128 castic). Dale je dobre si povsimnout, ze zrychleni se zvetsuje se vzrustajicim poctem castic (az na zajimavy pripad 512x512). To take odpovida predpokladum. Kyzeneho zrychleni oproti zakladni GPU verzi se dockalo take reseni za pouziti sdilene pameti (v radu jednotek az desitky procent).

7. Zaver

Implementoval jsem metodu [1], [2] pro simulaci satu na GPU pomoci jazyka CUDA. Zrychleni oproti CPU verzi se pohybuje v radu desitek. Zobrazovani je provedenou pouze velice jednoduse (wireframe nebo trojuhelniky/quady bez normal). Vice o program viz priloha A.

Zdroje

[1] Lander J., Devil in the Blue Faceted Dress: Real-time Cloth Animation, Game Developer, 1999, stranka 17-22, http://www.darwin3d.com/gamedev/articles/col0599.pdf

[2] [webova stranka]Mosegaards Cloth Simulation Coding Tutorial
http://cg.alexandra.dk/2009/06/02/mosegaards-cloth-simulation-coding-tutorial/

[3] [webova stranka] Verletova integracni metoda
http://en.wikipedia.org/wiki/Verlet_integration

Priloha A – Cloth Simulation aplikace

Ovladani

W,S,A,D	-	pohyb kamery
Mys	-	rozhlizeni
E	-	wireframe on/off
M	-	pouziti sdilene/globalni pameti
Q	-	kreslit trojuhelniky/quady
T/G	-	pocet iteraci korekcniho kroku +/-
Y/H	-	polomer koule +/-
U/J	-	rychlost koule +/-
I/K	-	odpor prostredi +/-
PG_UP	-	fullscreen
PG_DOWN	-	window
ESC	-	konec

Spusteni

ClothSim_*.exe <cfg_file> (* CPU/GPU)

Config File (mezera nutna u rovnitka a mezi FLOATy)

width = INT	//sirka
height = INT	//vyska
num_x = INT	//pocet castic na sirku
num_y = INT	//pocet castic na vysku
draw_cloth = BOOL	//zda se maji nebo nemaji kreslit saty (pro mereni se nekresli)
force = FLOAT FLOAT FLOAT	//sila;muze jich byt vice; gravitace je v programu automaticky

Priloha B - Obrazky