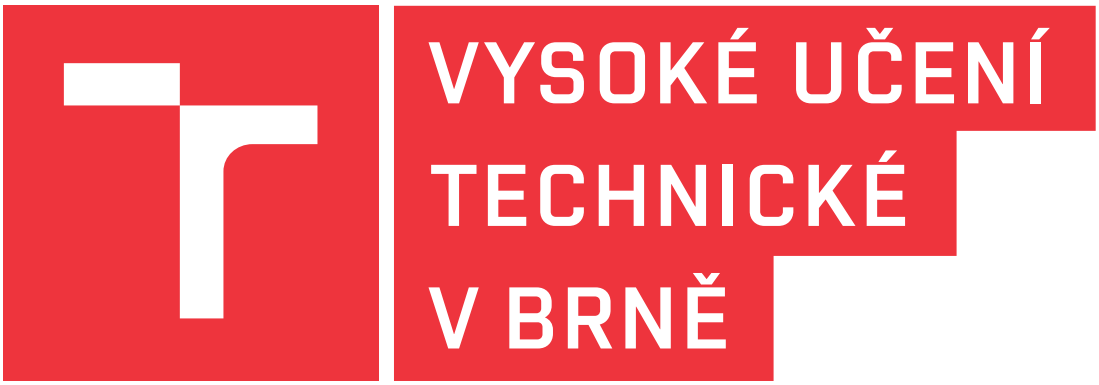


Grafické a multimediální procesory

Aplikace simulace vody pomocí celulárních automatů na GPU

11. ledna 2020

Autoři: Petr Flajšingr, xflajs00@stud.fit.vutbr.cz
Igor Frank, xfrank12@stud.fit.vutbr.cz
Fakulta Informačních Technologií
Vysoké Učení Technické v Brně

Úvod

Tato práce se zabývá návrhem a implementací algoritmů pro simulaci vody pomocí celulárních automatů na GPU s využitím jazyka C++ a knihovny OpenGL. První implementovaná metoda spadá do kategorie typických celulárních automatů. Druhá metoda se snaží být fyzikálně přesnější s využitím směrového vektoru pro určení směru pohybu vody.

Implementace

V obou metodách využíváme dvou bufferů – jeden pro čtení hodnot, které jsou dostupné z minulého kroku (případně inicializace) simulace. Druhý buffer slouží k zápisu nových hodnot. Po každém kroku simulace jsou tyto buffery prohozeny. Obě metody jsou implementovány za pomocí compute shaderů[1].

Obě metody používají Von Neumannovo sousedství. Simulace podporují čtyři typy buněk:

- Solid – reprezentuje buňku, která nemůže obsahovat kapalinu.
- Fluid – buňka, která může obsahovat kapalinu.
- Sink – buňka, která maže veškerou kapalinu, která do ní přiteče.
- Source – buňka, která vždy obsahuje maximální množství kapaliny.

První metoda

Jedná se o dvou-průchodovou metodu, kdy v prvním průchodu vypočteme změnu obsahu tekutiny ve vertikálním směru. V tomto průchodu se snažíme co nejvíce kapaliny přesunout z aktuální buňky do buňky o jedno níže. Pomocí prvního průchodu tak jednoduše simulujeme gravitaci. V druhém průchodu se pokoušíme veškerou kapalinu která se již nedokázala vměstnat do spodní buňky rovnoměrně přesunout do buněk okolních. Jako okolí jsme si zvolili jednoduché Von Neumanovo 4-okolí. Celý algoritmus je následující:

1. Vypočítej množství přítoku podle rovnice 1.
2. Vypočítej množství odtoku podle rovnice 1.
3. Omez hodnoty podle množství tekutiny v současné buňce a podle volného místa v buňce spodní.
4. Ulož změnu do zapisovacího bufferu.
5. Spočítej kolik sousedních buněk má nižší hladinu tekutiny.
6. Pro všechny horizontální sousední buňky proveď následující:
7. Vypočítej přítok ze sousední buňky podle rovnice 2.
8. Vypočítej odtok do sousední buňky podle rovnice 2.

[1] https://www.khronos.org/opengl/wiki/Compute_Shader

9. Přepočítej obsah aktuální buňky podle vypočtených hodnot.

$$\text{f(srcVol, dstVol)} = \begin{cases} srcVol + dstVol & \text{if } srcVol + dstVol < maxCellVol \\ \frac{maxCellVol^2 + (srcVol + dstVol) * maxCompression}{maxCellVol + maxCompression} & \text{if } srcVol + dstVol < 2 * maxCellVol + maxCompression \\ \frac{(srcVol + dstVol)}{2} & \text{jinak} \end{cases} \quad (1)$$

$$f(srcVol, dstVol, lowerCnt) = \begin{cases} \frac{srcVol - dstV}{lowerCnt} & \text{if } srcVol < dstVol \\ 0 & \text{jinak} \end{cases} \quad (2)$$

I když je teoreticky možné reprezentovat směr toku kapaliny, není jednoduše možné reprezentovat rychlost toku kapaliny. Dalším nedostatkem této metody je nemožnost efektivně měnit vlastnosti kapaliny. Prezentovaná kapalina tedy nese efekt vysoké viskozity a spíše než vodu připomíná med, lepidlo, či podobnou kapalinu.

Druhá metoda

Pro tuto metodu jsme zvolili alternativní přístup, kdy každá buňka obsahuje informaci o směru a rychlosti pohybu kapaliny. Celý algoritmus je rozdělen do tří částí.

První část algoritmu je následující:

1. Přidej vliv globálních sil (gravitace)
2. Pro všechny sousední buňky proveď následující:
   (a) Pokud je sousední buňka plná či se jedná o překážku pokračuj, jinak ukonči pro tuto buňku
   (b) Odraž vektor rychlosti od buňky
   (c) Přidej malý odraz ostatními směry
   (d) Akumuluj tyto hodnoty
3. Pokud byl proveden odraz vynuluj současnou rychlost buňky (v další části bude využito vektorů odrazu

Další část se zabývá přepočtem a úpravou vektorů tak, aby nedošlo k duplikování/ztrátě kapaliny:

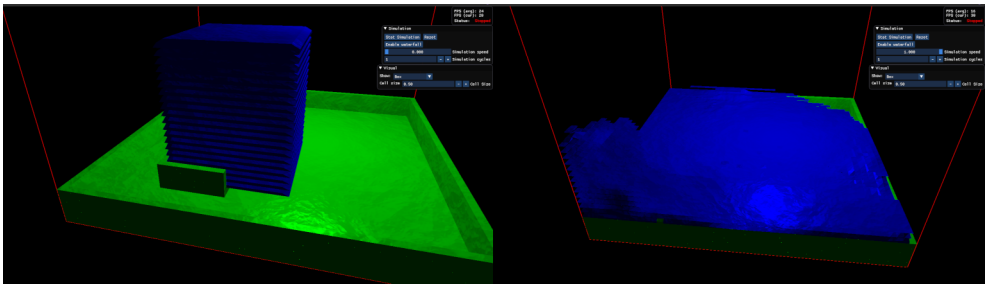
1. Vypočítej vektory pro odtok na základě vektorů odrazu a rychlosti (výsledkem jsou 2 tří-složkové vektory, každá složka pro jeden směr)
2. Uprav vektory podle obsahu buňky tak, aby nemohlo vytéct více tekutiny, než kolik obsahuje
3. Vymaskuj ty komponenty vektorů, které směřují do buněk od kterých se odrážíme
4. Odečti odtékající tekutinu od současné buňky

Poslední fáze se zabývá předáním kapaliny do okolních buněk:

1. Pro všechny sousední buňky:
2. Přičti množství přiteklé kapaliny do dané buňky
3. Přičti směrový vektor přiteklé kapaliny tak, aby se její pohyb projevil v dalších krocích

Důvodem vytvoření této metody byla snaha o pokrytí výrazného nedostatku metody předchozí – voda se nemůže odrážet, ani reálně nikam téct. Tento problém je v případě této metody vyřešen. Bohužel ale obsahuje plno dalších nedostatků. Metoda tedy není plně funkční, ale pro některé případy se chová lépe.

Na obrázku níže je vidět počáteční konfigurace simulace a její stav po několika krocích. V levé části snímku v průběhu simulace je vidět "vlna", která byla vytvořena odrazem od stěny.



Obrázek 1: Chování metody – vlevo počáteční konfigurace

Použité technologie

Aplikace je implementována v jazyce c++20 a GLSL[2]. Aplikace využívá knihovny geGL[3] pro binding OpenGL a pomocné třídy. Dále je zde využito knihovny sdl2cpp[4] pro snadnou práci s SDL2. Jako build systém je využit CMake[5]

Další využité knihovny třetích stran jsou: {fmt}[6], glm[7], imgui[8], tinyobjloader[9].

Pro vývoj jsme využili IDE CLion[10] a služby GitHub[11] pro hostování zdrojových kódů.

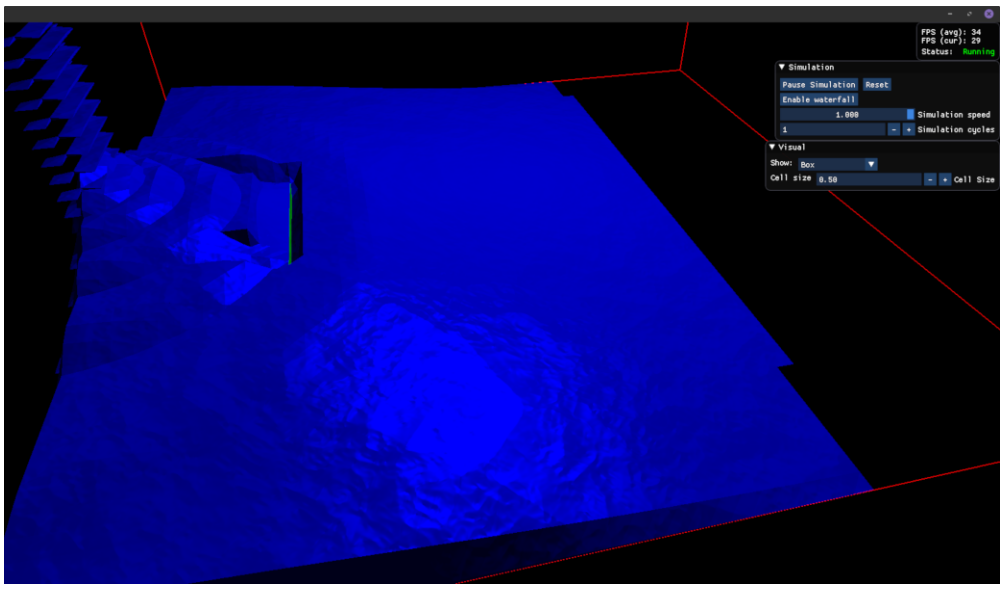
[2] https://www.khronos.org/opengl/wiki/OpenGL_Shading_Language
[3] https://github.com/dormon/geGL
[4] https://github.com/dormon/SDL2CPP
[5] https://cmake.org/
[6] https://github.com/fmtlib/fmt
[7] https://glm.g-truc.net/0.9.9/index.html
[8] https://github.com/ocornut/imgui
[9] https://github.com/syoyo/tinyobjloader
[10] https://www.jetbrains.com/clion/
[11] https://www.github.com

Vykreslování

Pro vykreslení je použito poměrně primitivní metody. Za předpokladu, že každou buňku mřížky vykreslujeme jako krychli (resp. obdelník), můžeme využít znalost zaplnění okolních buněk a tím přizpůsobit výšku hladiny vůči sousedním buňkám. Algoritmus je následující:

1. Vertex shader předá geometry shaderu informace o stavu buňky a zároveň vypočítá pozice buněk pomocí MVP matic.
2. Geometry shader zkontroluje, zda-li buňka obsahuje nějakou kapalinu. Pokud ano, pokračujeme dále.
3. Pro každou hranu buňky kolmou k rovině Y spočítá zaplnění sousedů, zprůměruje a vytvoří tím výšku zaplnění v dané hraně.
4. Na základě vypočtených hodnot jsou vystreamovány vertexy reprezentující vodu v buňce.
5. Ve fragment shaderu dochází k aplikaci osvětlení na fragmenty a také jednoduchému normal mapping založeném na fraktálovém šumu.

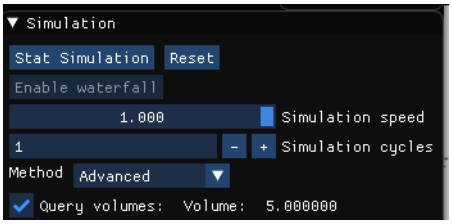
   Výsledek vykreslování je na obrázku 2.



Obrázek 2: Ukázka vykreslené vody

Ovládání aplikace

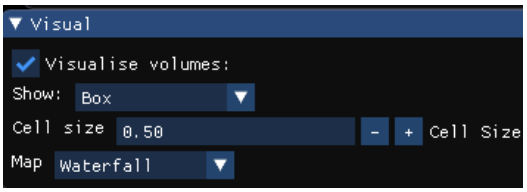
V demonstrační aplikaci je několik prvků, které umožňují změnit její chování. První částí je simulation:



Obrázek 3: UI pro ovládání simulace

Umožňuje zapnout/pozastavit simulaci, případně ji restartovat. Také umožňuje měnit rychlost simulace, její typ a počet cyklů simulace v jednom snímku. Zobrazuje celkové množství kapaliny v simulovaném prostoru.

Dalším prvkem je ovládání vykreslování. Umožňuje nastavit zobrazení okrajů buněk či prostoru vymezeného simulací. Také umožňuje přepnout do módu, kdy každá buňka je vykreslena barvou odpovídající jejímu zaplnění – méně zaplněné buňky se blíží více k červené než k modré barvě, buňky zaplněné nad klidovou maximální kapacitu jsou vykresleny zeleně. Poslední modifikovatelný parametr je velikost buněk v prostoru. Také umožňuje výběr jedné z předpřipravených demonstračních map.



Obrázek 4: UI pro ovládání vykreslování

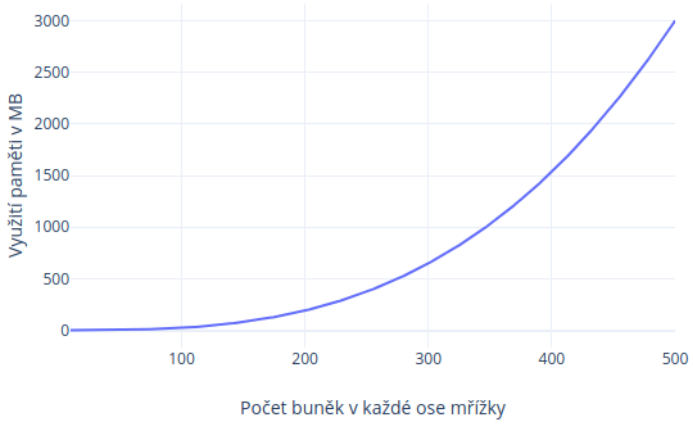
Časová a paměťová náročnost

Vzhledem k tomu, že veškerá data použita k simulaci jsou uložena v paměti grafické karty, se budeme zaobírat pouze využitím VRAM.

Všechna měření času byla prováděna 100 000-krát a následně zprůměrována. Měření bylo provedeno pro prázdnou mřížku, zaplněnou z 50% a plnou. Testováno na NVidia GTX 660 Ti.

První metoda

Každá buňka obsahuje následující data: obsah tekutiny (float), bitové pole pro příznaky (integer), obsah tekutiny v mezikroku (float). Celkově tedy každá buňka zabírá 12 bytů VRAM paměti.Vzhledem k tomu, že potřebujeme pro výpočet 2 kopie buňky (1 ze současného kroku, druhá pro budoucí), je paměťová náročnost zdvojnásobena, tedy na každou buňku spadá 24 bytů.



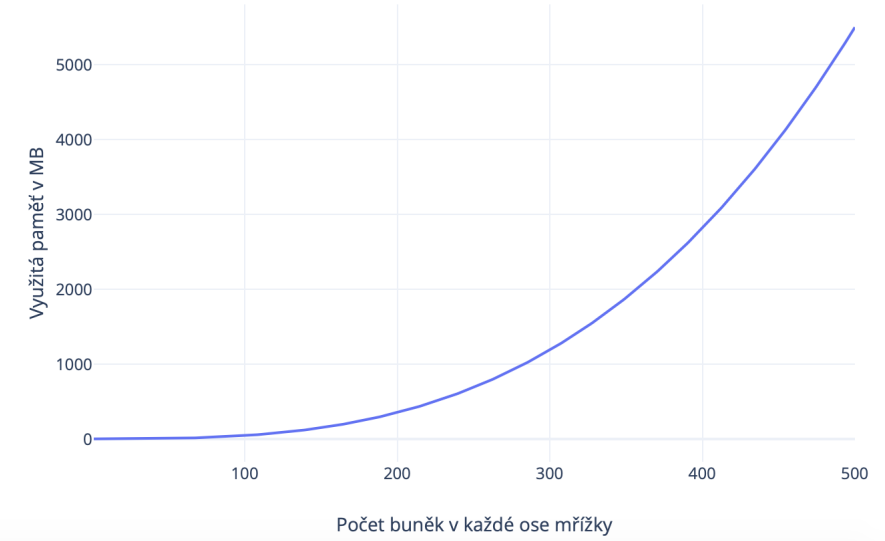
Obrázek 5: Závislost využité paměti na velikosti mřížky buněk

Časová náročnost je samozřejmě závislá na velikosti počítané mřížky a také na momentálním stavu zaplnění. V tabulce níže najdete naměřené hodnoty společně s informacemi o velikosti a zaplnění mřížky.

Mřížka	Buňky (tisíce)	Čas - prázdný grid (μs)	Čas - 50% grid (μs)	Čas - plný grid (μs)
10	1	0.69	0.96	0.94
20	8	0.93	1.12	1.08
50	125	58	73	70
100	1000	1014	1385	1257
200	8000	3120	4203	3830

Druhá metoda

Pro tuto metodu potřebujeme v každé buňce informaci o obsahu kapaliny (float) a rychlosti a směru pohybu (3 float). Tyto hodnoty jsou použity 2x, podobně jako pro metodu předchozí. Dále pro mezi-výpočty používáme 2x vektor pohybu (6 float), příznaky (integer). Celková potřebná paměť pro jednu buňku je tedy 44 bytů.



Obrázek 6: Závislost využité paměti na velikosti mřížky buněk

Časová náročnost, podobně jako v předchozím případě, je opět závislá na velikosti mřížky a stavu zaplnění. V tabulce níže jsou uvedeny hodnoty s informacemi o zaplnění mřížky.

Mřížka	Buňky (tisíce)	Čas - prázdný grid (μs)	Čas - 50% grid (μs)	Čas - plný grid (μs)
10	1	15	90	87
20	8	50	278	282
50	125	343	1830	1943
100	1000	2909	12321	13823
200	8000	15877	83231	95262

Závěr

V programu jsme implementovali dvě metody. U první metody jsme se soustředili především na jednoduchost a rychlost. Metoda však není dostatečně fyzikálně přesná a tekutina působí silně viskózně.

U druhé metody jsme se proto soustředili na lepší reprezentaci fyzikálních vlastností vody. V této metodě se pokoušíme simulovat především odrazy a vnitřní tlak. Metoda v určitých případech opravdu působí na oko mnohem "realističtěji", nicméně má několik nedostatků jako vyšší výpočetní náročnost, či vysoký neklid hladiny.

Celkově jsme si na tomto projektu vyzkoušeli především práci s compute shadery, paralelní myšlení a výhody a nevýhody programování na GPU. Celkově by se program dal dále rozšířit o načítání terénu ze souboru, vylepšení pokročilejší metody, případně zlepšení vizuální stránky například aplikováním Marching cubes na povrch tekutiny.

Literatura

[1] Jakub MEDVECKÝ-HERETIK. Simulace vody v herním prostředí [online]. Diplomová práce, Masarykova univerzita, Fakulta informatiky, Brno, 2018 [cit. 2019-12-30].