Zobrazování dýmu simulovaného pomocí částicového systému

Petr Mohelník xmohel01@stud.fit.vutbr.cz,
Tomáš Růžička xruzic42@stud.fit.vutbr.cz

30. prosince 2015

1 Úvod

Tato práce se zaměřuje na realistickou simulaci a zobrazení dýmu pomocí grafické karty. Využití by mělo být v aplikacích pracujících v reálném čase, např. hrách. V počítačových hrách může být dým využit u explozí, mlhy, ohně apod.

2 Teorie

V této kapitole je popsána teorie použitá při výpočtu a zobrazení dýmu.

2.1 Simulace

Přístupy pro simulaci kapalin a plynů se dají rozdělit do dvou kategorií. Jeden je *Eulerův přístup*, kde je prostor rozdělen na fixní 2D nebo 3D mřížku. Každá buňka v mřížce obsahuje informace o kapalině nebo plynu na dané neměnné pozici. Tyto informace mohou být tlak, hustota, teplota, viskozita aj. Oproti tomu *Lagrangeův přístup* využívá částice s proměnlivou pozicí jako nositele informace. Není vázán na fixní mřížku a může se libovolně rozpínat v prostoru. Na druhou stranu může být výpočetně náročnější kvůli nutnosti vyhledávat okolní částice. Tyto přístupy se někdy kombinují.

V tomto projektu jsme zvolili *Lagrangeovu metodu* [1] založenou na *Smoothed Particle Hydrodynamics* (*SPH*), která pro simulaci využívá částice. Řeší nestlačitelnost, symetrii sil, konzervaci hybnosti. *SPH* je interpolační metoda. Každá částice má prostorovou vzdálenost h určující, které okolní

částice na ní mají vliv. V *SPH* se fyzikální hodnota na pozici $\mathbf{r}$ určí jako vážená suma fyzikálních hodnot ϕ_j sousedních částic j:

$$\phi(\mathbf{r}) = \sum_j m_j \frac{\phi_j}{\rho_j} W(\mathbf{r} - \mathbf{r}_j, h) \tag{1}$$

kde m_i je hmotnost částice. Hmotnost je konstantní po celou dobu simulace a shodná pro všechny částice a $W(\mathbf{r}, h)$ je symetrická vyhlazovací funkce. Hustota ρ_i se vypočítá:

$$\rho_i = \sum_j m_j W(\mathbf{r}_i - \mathbf{r}_j, h) \tag{2}$$

kde $\mathbf{r}_i$ je pozice částice i. Akcelerace $\mathbf{a}_i$ částice i se určí následovně:

$$\mathbf{a}_i = \frac{\mathbf{f}_i}{\rho_i} \tag{3}$$

kde $\mathbf{f}_i$ se spočítá jako $\mathbf{f}_i = \mathbf{f}_i^{viscosity} + \mathbf{f}_i^{pressure} + \mathbf{f}_i^{external}$.

Tlaková síla $\mathbf{f}_i^{pressure}$ je ze vztahu 1 určena:

$$\mathbf{f}_i^{pressure} = -\sum_j m_j \frac{p_j}{\rho_j} \nabla W(\mathbf{r}_i - \mathbf{r}_j, h) \tag{4}$$

Bohužel tato síla není symetrická, jak může být vidět při interakci pouze dvou částic. Gradient je nula uprostřed, proto částice i využívá pouze tlak částice j pro výpočet síly a naopak. Tlak v lokacích dvou různých částic není shodný. Tlaková síla je symetrizována následovně (mohou být i jiné tvary rovnice pro symetrizaci):

$$\mathbf{f}_i^{pressure} = -\sum_j m_j \frac{p_i + p_j}{2\rho_j} \nabla W(\mathbf{r}_i - \mathbf{r}_j, h) \tag{5}$$

Tlak p_i se určí pomocí modifikované rovnice ideálního plynu:

$$p_i = k(\rho - \rho_0) \tag{6}$$

kde ρ_0 je klidová hustota a k je tuhost plynu.

Síla viskozity $\mathbf{f}_i^{viscosity}$ je z rovnice 1 určena:

$$\mathbf{f}_i^{viscosity} = \mu \sum_j m_j \frac{\mathbf{v}_j}{\rho_j} \nabla^2 W(\mathbf{r}_i - \mathbf{r}_j, h) \tag{7}$$

kde μ je viskozita kapaliny nebo plynu a $\mathbf{v}_i$ je rychlost částice i. Tato síla je také nesymetrická. Je symetrizována následovně:

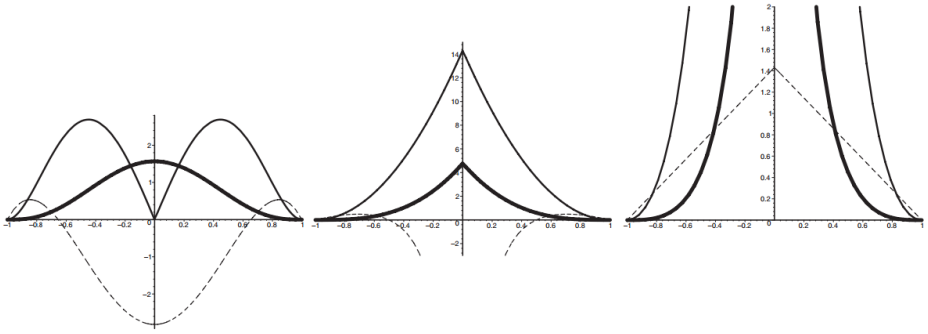
$$\mathbf{f}_i^{viscosity} = \mu \sum_j m_j \frac{\mathbf{v}_j - \mathbf{v}_i}{\rho_j} \nabla^2 W(\mathbf{r}_i - \mathbf{r}_j, h) \tag{8}$$

Pro kapalinu by bylo vhodné počítat povrchové napětí. To my pro dým nepotřebujeme. Určujeme vztlakovou sílu [2], která je způsobena šířením teplot. My modelujeme izotermální plyn proto ji vypočítáme jako:

$$\mathbf{f}_i^{buoyancy} = b(\rho_i - \rho_0)\mathbf{g} \tag{9}$$

kde $b > 0$ je koeficient vztlaku. Pokud bude hustota menší než klidová, částice budou tlačeny vzhůru.

Vyhlazovací funkce $W(\mathbf{r}, h)$ velmi ovlivňují rychlost, stabilitu a přesnost *SPH* metod. Používáme tři různé kernely, pro dosažení co nejlepších výsledků simulace, (obr. 1). Jeden pro výpočet hustoty, druhý pro tlakovou sílu a třetí pro sílu viskozity.



Obrázek 1: Použité kernely. Zleva doprava použity pro výpočet hustoty, tlakovou sílu a sílu viskozity. Tlusté čáry jsou kernely, tenké gradienty a šrafované Laplaciány.

Pro integraci částic v čase používáme Eulerovo schéma. Zde se prvně aktualizuje pozice rychlost $\mathbf{v}$:

$$\mathbf{v}_{t+\Delta t} = \mathbf{v}_t + \Delta t \mathbf{a}_t \tag{10}$$

Poté se aktualizuje pozice $\mathbf{r}$:

$$\mathbf{r}_{t+\Delta t} = \mathbf{r}_t + \Delta t \mathbf{v}_{t+\Delta t} \tag{11}$$

Následně se určují kolize. Při kolizi s prostředím je částice odražena směrem od překážky.

2.2 Řazení

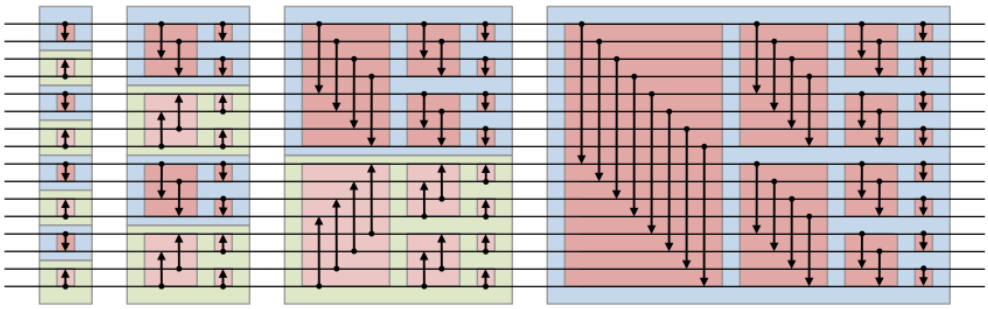
Pro správné zobrazování částic je potřeba řazení. Pro implementaci jsme zvolili *Bitonic sort* [3], protože je paralelní a datově nezávislý a tudíž vhodný pro implementaci na GPU. Provádí $O(n \log(n^2))$ porovnání.

Algoritmus řadí *bitonické sekvence*. Bitonická sekvence je sekvence, kde $x_0 \leq \ldots \leq x_k \geq \ldots \geq x_{n-1}$ pro nějaké $k, 0 \leq k \leq n$, nebo cyklické posunutí této sekvence.

Bitonic sort (viz obrázek 2) provádí $\log n$ iterací, kde výstupem každého průchodu je pole tvořené střídavě rostoucími a klesajícími posloupnostmi o velikosti 2^i, kde $1 \leq i \leq \log n$ je číslo iterace. Toho se dosáhne dalšími i iteracemi v rámci i-té iterace. Každá tato vnitřní iterace provádí porovnání mezi prvky s poloviční vzdáleností než ta předchozí (viz obrázek 2). Nakonec vznikne jedna rostoucí nebo klesající posloupnost.

Takto funguje bitonic sort pro posloupnosti délky mocniny 2. Triviální řešení pro libovolné délky posloupnosti by bylo na konci uvažovat prvky maximální (resp. minimální) hodnoty. Protože bitonic sort řadí prvky v sekvencích o střídajících se směrech (rostoucí nebo klesající), tyto prvky by byly přesouvány a musely by tedy fyzicky existovat.

Nicméně v bitonic sortu není podstatné zda jsou sekvence řazeny jako rostoucí/klesající nebo naopak. Proto může být řazení upraveno tak, aby v každé iteraci i byla na konci rostoucí (resp. klesající) sekvence a tím hodnoty nebudou nikdy přesouvány a tudíž nemusí existovat.

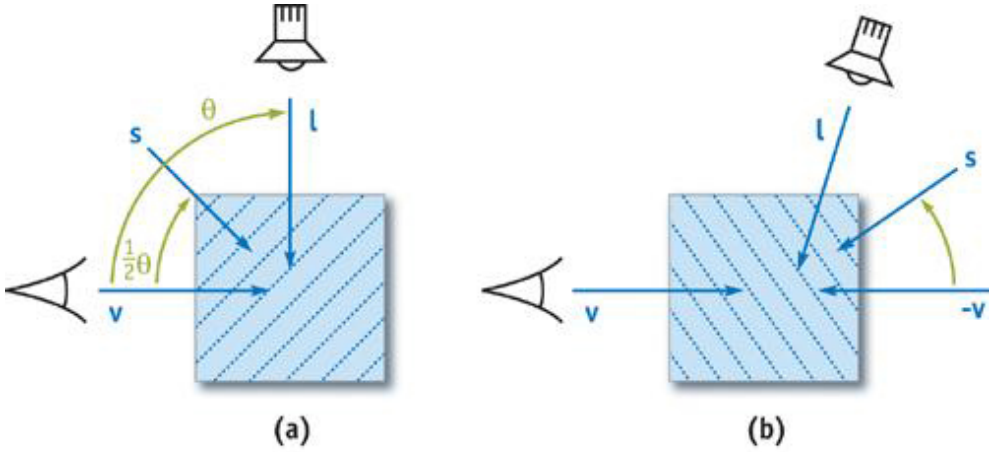


Obrázek 2: Sorting network Bitonic sortu. Převzato z Wikipedie.

2.3 Vykreslení

Při kroku vykreslování jsme vycházeli z [4], kde byla představena technika seřazení pomocí tzv. *half vektoru* (HV). Tento vektor je otočen mezi směrem pohledu kamery a pozice světla (obr. 3). Hlavní výhodou je, že k vykreslení

stačí pouze jedna hloubková mapa. Dým je pak vykreslen v postupných řezech, které jsou kolmé ke směru *HV*. A díky tomu lze akumulovat zastínění pomocí *blendingu* do tzv. *light bufferu* (*LB*) a také vypočítat *hloubkovou mapu*.



Obrázek 3: Ukázka dvou případů výsledku *half vektoru*, který je umístěn mezi vektor pohledu kamery a směru světla. Převzato z [4].

Nicméně při kreslení je důležité, aby byl dým kreslen správným směrem. Pokud je směr světla a pozorovatele podobný (úhel mezi nimi je menší než 90°) potom je dým kreslen zepředu dozadu. Pokud světlo leží naproti kameře, potom jsou částice kresleny zezadu dopředu.

Samotné řazení řezů pracuje na principu projekce pozice částice na *HV*. Tuto projekci lze jednoduše spočítat pomocí skalárního součinu.

Při stínování je pak využit výsledek *light bufferu* spolu se *stínovou mapou*.

3 Popis řešení

Pro implementaci jsme použili *OpenGL* ve verzi `4.3`. K simulaci dýmu byly využity *compute shadery* spolu s *shader storage* buffery. V části stínovány zajišťovaly *framebuffery* záznam do textur.

3.1 Simulace

Pro simulaci je zavedeno několik shader storage buffer objectů (SSBO), všechny mají velikost maximálního počtu částic v systému. Pro vytvoření základní

struktury částicového systému jsme se inspirovali ve slajdech z přednášky [5].

Základní buffer je `ParticlePool`, který slouží pro uložení částic. Každá částice má uložených 12 floatů: pozici, rychlost, sílu, hustotu, tlak a čas života.

Další je `DeadList`, který obsahuje indexy do `ParticlePool` s částicemi, které jsou volné k použití. K tomu se používá atomický čítač, určující pozici od které vpravo jsou volné indexy.

`SortList` slouží k seřazení částic. Pro každou částici obsahuje klíč a index do `ParticlePool`. Využívá atomický čítač určující počet částic, které se budou zobrazovat, tedy řadit.

U simulace je nutné procházet okolní částice. Pokud nechceme provádět n^2 porovnání, musíme to nějak řešit. Používáme uniformní mřížku založenou na řazení, jak je nastíněno v [6]. Tato mřížka omezuje pohyb částic pouze do prostoru pokrytého mřízkou. Pro neomezený pohyb částic se dají použít hashovací metody, ale ty mohou být pomalejší, protože do sousedních oblasti mapují částice, které sousední nejsou. Tato mřížka se vytváří každý snímek znovu. Pro každou částici se určí do jaké buňky spadá. Index této buňky se použije jako klíč pro řazení. Po seřazení částic jsou zjištěny a uloženy počáteční indexy pro každou buňku. Při procházení okolních částic se prochází $3 \times 3 \times 3 = 27$ okolních buněk pro každou částici. Podporuje neomezené množství částic v buňce a díky seřazení urychluje přístupy do paměti pro okolní částice. K tomu jsou zavedeny další dva SSBO. `GridList` obsahuje index do `ParticlePool` a index buňky jako klíč pro řazení. Využívá atomický čítač určující počet částic aktuálně v oblasti mřížky. `StartIndexList` obsahuje pro každou buňku její počáteční index v `GridList`.

V programu je použito množství compute shaderu. Popíšeme si je v pořadí v jakém jsou volány.

`emit_particle` generuje malé množství částic v časových intervalech. Vytáhne si z `DeadListu` částici a nastaví ji pozici, rychlost a čas života.

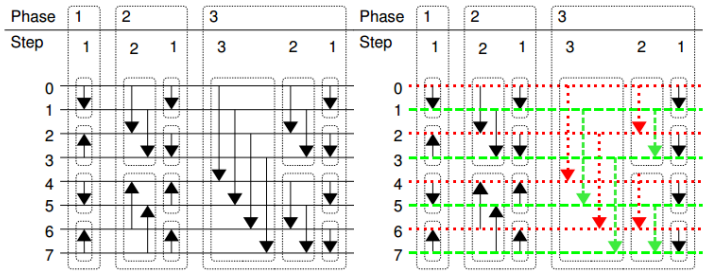
`grid_particle_divide` pro každou částici určí index buňky do které spadá, pokud do žádné nespadá, částice je vrácena do `DeadListu`. Dále jsou částice seřazeny (řazení bude popsáno dále) a v compute shaderu `grid_particle_find_start` jsou vyhledány počáteční indexy buněk a částice z `ParticlePool` jsou zkopírovány do `ParticlePoolSorted` v seřazeném pořadí. To by mělo urychlit přístup do paměti při procházení okolních částic, díky blízkosti částic z jedné buňky v paměti. Velikost buněk je shodná s průměrem h vyhlazovacího kernelu. Tím je zajištěno, že všechny částice spadající do jeho

vlivu budou v některé z 27 okolních buněk.

Nyní přichází na řadu samotná simulace. `simulate_particle_density` prochází pro každou částici 27 buněk mřížky a v každé všechny částice. S těmi potom vypočítá hustotu za využití vyhlazovací funkce podle rovnice 2. Kvůli velkému množství přístupů do globální paměti má tento krok velký vliv na celkovou rychlost. `simulate_particle_pressure` vypočítá tlak podle rovnice 6 v závislosti na hustotě spočítané v předchozím kroku. Následuje `simulate_particle_force`, který počítá vnitří sílu působící na částici. Prochází okolní částice obdobně jako při výpočtu hustoty. Využívá ale jiné dva vyhlazovací kernely a počítá rovnice 5 a 8. Simulaci ukončuje `simulate_particle`. Ten pro každou částici dekremetuje čas života a pokud je čas záporný, vrátí ji do `DeadListu`. Poté se určí vztlaková síla z rovnice 9 a určí se síla jednoduchého větru, který jsme zavedli pro rozhýbání částic. Následuje integrace v čase podle 10 a 11. Nakonec se zkontrolují kolize s koulemi a krychlí, ve které je dým uzavřen. Při kolizi jsou částice odraženy. Částice z `ParticlePoolSorted` jsou zkopírovány zpět do `ParticlePool`.

Následně jsou částice řazeny pro vykreslení, viz kapitola 3.2. Pro řazení jsou použity celkem tři compute shadery. `sort_particle_local` je první v pořadí. Tento shader slouží pro urychlení řazení využitím lokální paměti. V rámci každé pracovní skupiny jsou hodnoty klíč a index nakopírovány do lokální paměti a poté je prováděno řazení dokud velikost seřazených podsekvencí není velikost pracovní skupiny. Pro naší velikost skupiny 512 se pro každé vlákno bude provádět 44 porovnání dvojic. Následuje `sort_particle`, který využívá globální paměti a registrů, neobsahuje žádné cykly, ty jsou na straně CPU. V rámci jednoho volání provede porovnání odpovídající 1, 2, 3 nebo 4 vnitřním cyklům. Na obrázku 4 je vidět příklad porovnání, která se dají provést v rámci jednoho vlákna. Hodnoty se kterými se bude pracovat se na začátku uloží do registrů, poté se provedou všechna porovnání a nakonec se uloží zpět do globální paměti. Tento shader se volá, dokud vzdálenost porovnávaných hodnot je větší než velikost pracovní skupiny. Poté se volá `sort_particle_local_inner` který dokončí vnitřní smyčku využitím lokální paměti i registrů. Funguje obdobně jako sort_particle, ale obsahuje cyklus a hodnoty jsou na začátku uloženy z globální do lokální paměti. Celkem je řazení v rámci jednoho snímku provedeno 3x.

Klidovou hustotu ρ_0 uvažujeme $0.59(\frac{kg}{m^3})$ a hmotnost m jedné částice $5 \times 10^{-5}(kg)$. Radius $h = 0.06(m)$ je určen, aby do něj spadalo asi 12 částic, dále $k = 4(J)$, $\mu = 0.01(Pa \cdot s)$ a $b = 5$. Tyto hodnoty jsou převzaty z [2], kde jsou určeny pomocí fyzikálních vztahů pro objem, hustotu, hmotnost aj. Hodnota



Obrázek 4: Všechna porovnání červeně je možné provést v rámci jednoho vlákna, stejně tak ty zeleně. Převzato z [3].

k není fyzikálně přesná, protože by simulace dobře nefungovala s tak velikou hodnotou. To jak dobře tato hodnota bude fungovat je závislé na množství částic v simulaci.

3.2 Vykreslení

Při vykreslování byla použita metoda seřazení řezů dýmu pomocí *half vektoru*, nicméně nepodařil se nám správně nastavit režim míchání barev. Z toho důvodu byl dým po simulaci řazen dvakrát, a to pro pohled z pozice světla a z pozice kamery.

Při vykreslování z pozice kamery byly nejprve vykreslovány neprůhledné objekty scény do *hloubkové mapy*. Poté byl vykreslován dým do *light bufferu* a také do *hloubkové mapy*. *Light buffer* zde představuje klasickou texturu, do které je ukládán světelný příspěvek, který představují částice dýmu.

Dalším krokem při vykreslování je vytvoření efektu pro paprsky slunce (*sun shafts*). Nejprve je potřeba vykreslit do *framebufferu* scénu z pozice kamery, přičemž scéna je specificky obarvená. Objekt slunce má bílou barvu, pozadí tmavě modrou a ostatní objekty včetně dýmu je vykreslen s černou barvou. Pro další postup je pak vypočtena projekce pozice světla na projekční plátno. Pomocí této pozice je v dalším kroku vykreslen pouze čtverec s texturou výsledku z předchozího kroku. Při tomto kreslení je výstup rozmazán ve směru od pozice světla – tedy ve shaderu pro každý pixel ve směru ke světlu. Tento výsledek je nakonec zkombinován s klasickým kreslením scény pomocí *aditivního míchání* (obr. 5).

Při standardním vykreslování scény z pozice kamery jsou nejprve kresleny neprůhledné objekty. Pro ně jsou využity *stínové mapy* a *akumulační buffer* (*light*) z předchozích kroků. *Stínové mapy* (*hloubkové*) jsou klasicky použity s *shadow samplerem* a s projektivním dotazováním se na zastínění. Dále je

Obrázek 5: Postup vytvoření efektu paprsků slunce. Vlevo: vykreslení scény ve speciálních barvách. Uprostřed: rozmazání textury. Vpravo: Additivně smíchané s výstupem.

využit *light buffer* stejným způsobem pro určení průsvitnosti dýmu. Tento postup je dál kombinován se *stínovou mapou* dýmu, pro oříznutí zastínění pro bližší předměty ke světlu.

Při stínování částic dýmu je také využity *stínové mapy* předmětů a dýmu stejným způsobem. Pro určení průsvitnosti v dýmu pro částice, které neleží přímo na světle, ale jsou zastíněny jinými částicemi se využívá kombinace *stínové mapy* dýmu a *light bufferu*. Tato stínová mapa je však použita jako *hloubková mapa*, pomocí které, spolu s hloubkou fragmentu je určen rozdíl hloubky. Ten je využit k výpočtu míry zastínění.

4 Vyhodnocení

Simulace vyžaduje poměrně malé kroky Δt. Ve výsledné aplikaci jsme zvolili $\Delta t = 0.005s$ jako maximum, které se použije pokud čas mezi snímky je větší. Simulace ještě docela funguje pro $\Delta t = 0.01s$. U větších kroků se simulace rozkmitá, částice budou vystřelovány velkou silou a osamostatňovány. Osamostatnění částice je velký problém, který by ideálně neměl nikdy nastat. Je-li částice samostatná hustota ze vztahu 2 je nulová a protože se touto hustotou v dalších výpočtech dělí, nastává problém. V naší aplikaci se v takovém případě ošetřuje aby nedošlo k dělení nulou a v dalším kroku jsou takové částice zabity. Další problémy mohou nastat např. při emitaci částic. Vygenerují-li se částice příliš blízko, jsou od sebe odpuzovány příliš velkou silou a simulace se rozpadne. Jsou-li naopak příliš daleko, přitahují se k sobě a tvoří malé shluky částic.

Simulace tedy neběží v reálném čase pokud není dostatek fps. To ale je běžné např. ve hrách, že fyzikální simulace jsou pomalejší.

Ačkoliv se SPH používá převážně pro simulaci tekutin, podařilo se nám dosáhnout zajímavých výsledků i pro dým. Ale existují i nějaké hry, které používají SPH pro simulaci kouře (viz [7]). Tedy i naše simulace by mohla být použitá třeba v nějaké hře, když by se vhodně nastavily parametry a

rozložily počáteční částice.

Obr. 6 7 8 9.

5 Závěr

Tady by mělo být stručně napsané jak to funguje.

Reference

[1] M. Müller, D. Charypar, and M. Gross, "Particle-based fluid simulation for interactive applications," in *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ser. SCA '03. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2003, pp. 154–159. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.121.844&rep=rep1&type=pdf

[2] M. Kelager, "Lagrangian Fluid Dynamics Using Smoothed Particle Hydrodynamics," Jan. 2006. [Online]. Available: http://image.diku.dk/projects/media/kelager.06.pdf

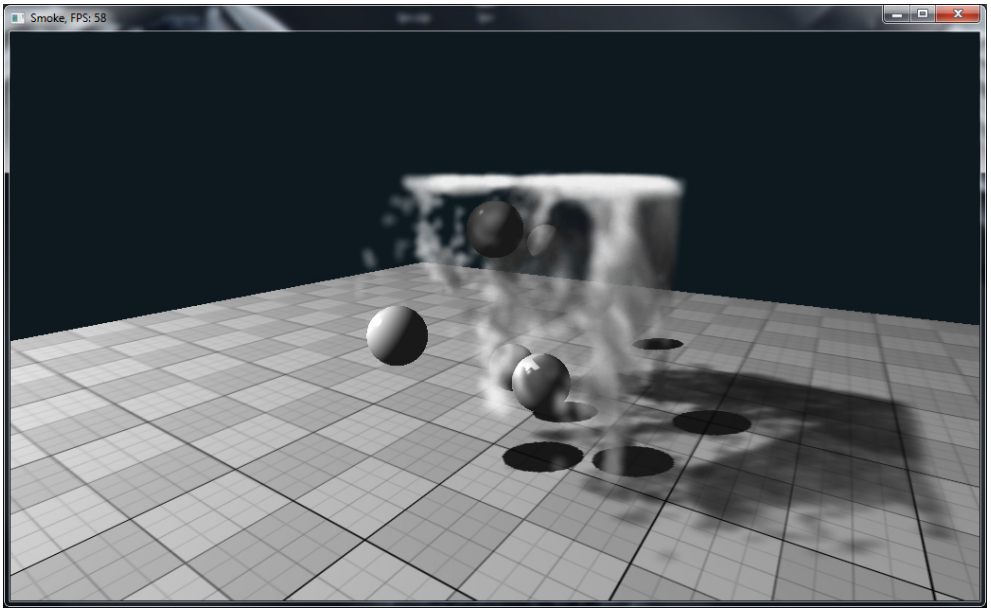
[3] H. Peters, O. Schulz-Hildebrandt, and N. Luttenberger, "Fast in-place sorting with cuda based on bitonic sort," in *Proceedings of the 8th International Conference on Parallel Processing and Applied Mathematics: Part I*, ser. PPAM'09. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 403–410. [Online]. Available: https://comsys.informatik.uni-kiel.de/wp-content/uploads/2012/02/09-Peters-Fast˙in-place˙sorting.pdf

[4] S. Green, "Volumetric particle shadows," NVIDIA, Tech. Rep., 2012. [Online]. Available: http://developer.download.nvidia.com/assets/cuda/files/smokeParticles.pdf

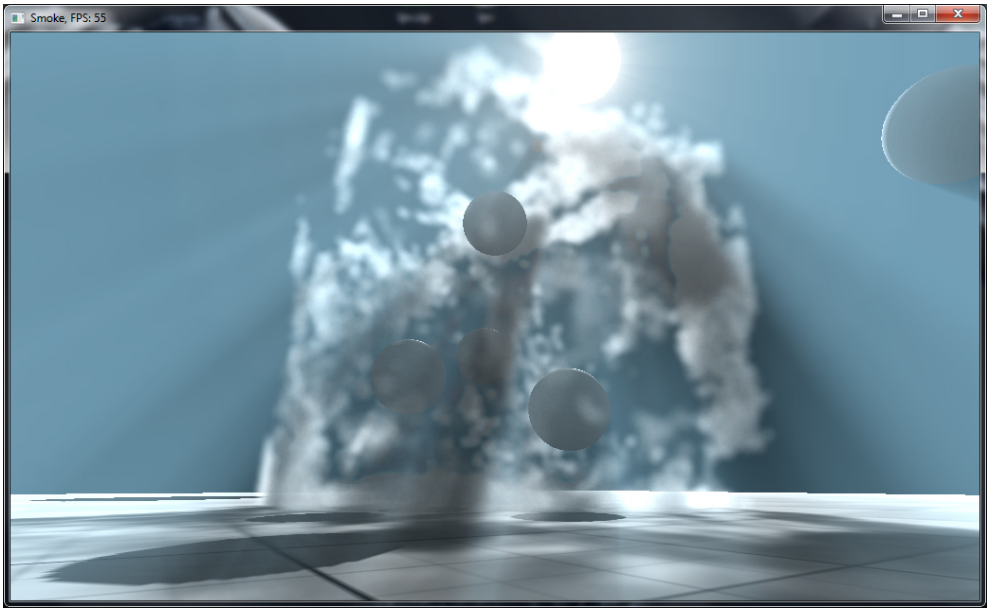
[5] G. Thomas, "Compute-Based GPU Particle Systems," 2014. [Online]. Available: http://twvideo01.ubm-us.net/o1/vault/GDC2014/Presentations/Gareth˙Thomas˙Compute-based˙GPU˙Particle.pdf

[6] S. Green, "Particle-based Fluid Simulation," 2008. [Online]. Available: http://developer.download.nvidia.com/presentations/2008/GDC/GDC08˙ParticleFluids.pdf

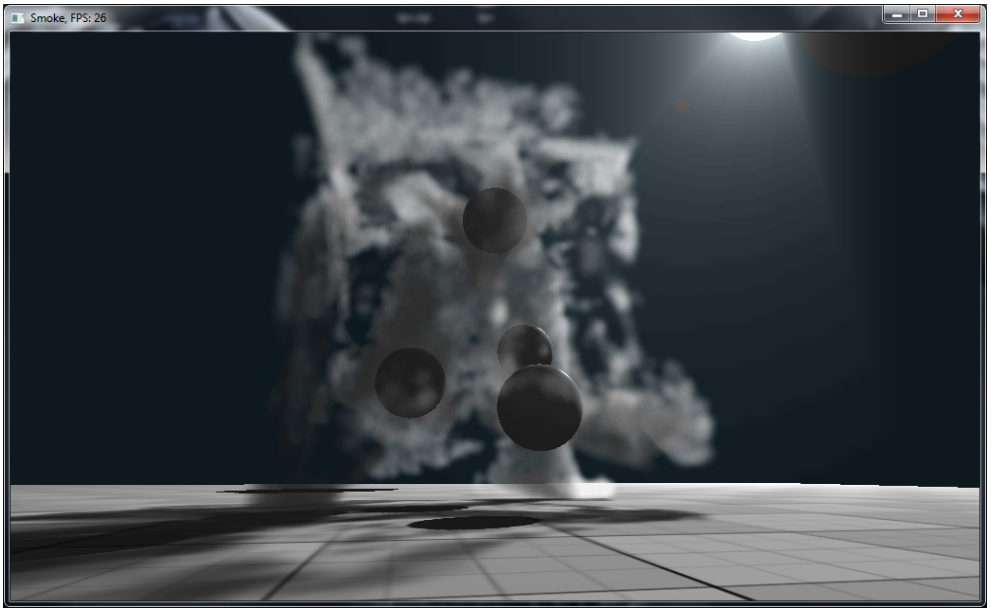
[7] "GPU PhysX in Metro: Last Light." [Online]. Available: http://physxinfo.com/news/11443/gpu-physx-in-metro-last-light/



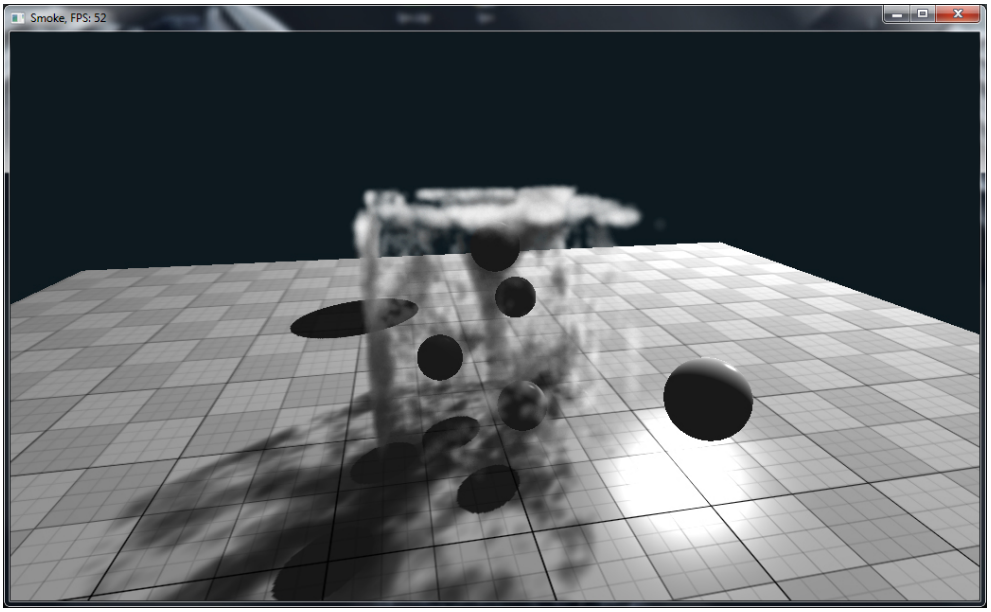
Obrázek 6: Výsledná aplikace.



Obrázek 7: Výsledná aplikace.



Obrázek 8: Výsledná aplikace.



Obrázek 9: Výsledná aplikace.