PIC/FLIP fluid sim

Filip Toman, Dominik Chaloupka

B221/B4M39MMA

11.01.2023

Úvod

Simulace tekutin je v počítačové grafice populární téma. Reálný svět zachycovaný počítačovou grafikou neobsahuje jen jednoduché objekty z tvrdých materiálů, které je jednoduché vykreslovat, ale i pohyblivější fyzikální prvky. Jedním z těchto prvků jsou právě tekutiny. Nejde zde ale jen o počítačovou grafiku. Simulace tekutin má dalekosáhlejší využití. Lze díky ní přesněji plánovat aktivní opatření proti povodním, simulovat zaplavení nějakých prostor a podle toho modelovat evakuační cesty či předpovídat dopady hurikánů.

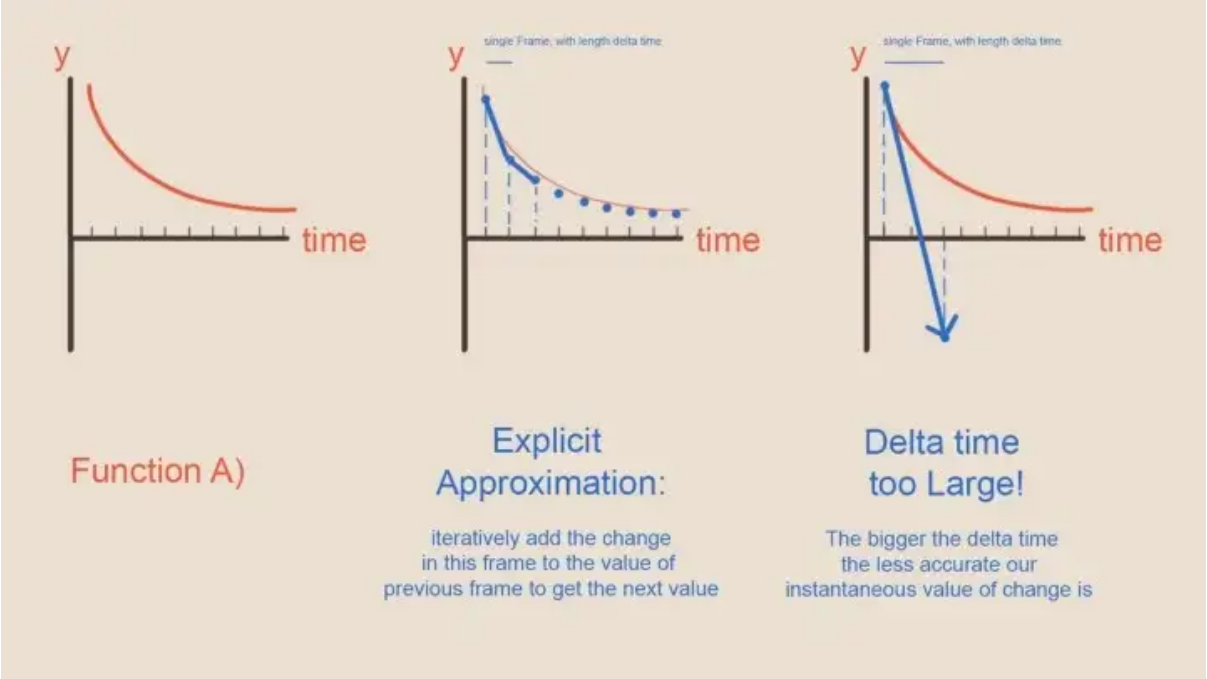
Analýza

Pro začátek je nutné zmínit některé základní vlastnosti kapalin, které budeme simulovat. První vlastností je difúze. Díky Brownově pohybu částic se kapaliny s různými vlastnostmi postupně promíchají do rovnovážného stavu bez jakéhokoliv působení vnějších sil. Tento jev je velmi dobře pozorovatelný například při vložení sáčku čaje do horké vody.
Druhou vlastností je viskozita - ta charakterizuje míru odporu kapaliny k její deformaci. Velký rozdíl poznáme, když budeme prstem chvíli míchat vodu ve sklenici a poté se pokusíme zamíchat med v jiné sklenici. Voda je oproti medu méně viskózní a tak pohyb prstem bude bez většího odporu, kdežto ve sklenici medu se nám to nemusí podařit.
Třetí vlastností je advekce - popisuje přenos částic kapaliny jako celku. Znovu zmíním sklenici s vodou, kterou prstem zamícháme. Ačkoliv jsme se prstem přímo dotknuli jen malého množství částic, jejich proudění přenáší síly i na okolní částice a postupně se rozproudí kapalina v celé sklenici.

Základní východisko všech simulací tekutin jsou Navierovy-Stokesovy rovnice popisující proudění nestlačitelné Newtonské tekutiny. Newtonská tekutina je model látky řídící se Newtonovým zákonem viskozity. Tento model je vhodný pro většinu tekutin, avšak existují i newtonské látky, pro které zákon viskozity neplatí. Navier-Stokesova rovnice je soustava nelineárních parciálních diferenciálních rovnic 2. řádu. Zde se přístup k těmto simulacím rozděluje. Jednou možností je simulovat vlastnosti a chování celkové masy kapaliny, kde není důležitý její vzhled, ale její chování. Většina aplikací v rámci počítačové grafiky se ale vydává druhým směrem, pro který je důležitý vzhled, ale není nutné simulovat chování každé částice. Je zde tedy kladen daleko větší důraz na real-time simulaci.

K řešením těchto simulací je poté možné přistoupit třemi způsoby. Simulací pohybu jednotlivých částic a jejich působení na okolní částice, simulací pomocí mřížky (vektorového vícerozměrného pole) a kombinací těchto dvou přístupů. Mřížkový přístup s sebou přináší rozměrové omezení, protože tekutinu simulujeme pouze v rámci rozměrů dané mřížky. Na každý bod mřížky poté aplikujeme výpočty zmíněných tří vlastností.
Důležité je také zmínit nestlačitelnost tekutiny a zákon zachování hmotnosti. V naší simulované soustavě nemůže bez vnějšího zásahu žádná kapalina vznikat ani zanikat, ani není jakkoliv stlačitelná. Při naivní implementaci může a velmi pravděpodobně dojde k tomu, že v některých místech kapalina zaniká a v jiných místech zaniká, aniž by k tomu byla určena. Řekneme, že kapalina konverguje, respektive diverguje k těmto místům. Reálná kapalina se ale takto nechová a je důležité toto chování správně simulovat. Pro jednoduché přiblížení situace si představme dva proudy kapaliny směřující přímo proti sobě. V neopravené simulaci v místě styku těchto dvou proudů bude kapalina zanikat. V reálném světě si ale proudění kapaliny pouze vybere jiný směr. Vzniká tím v kapalině víření.

Nestlačitelnost je možné vyřešit procesem zvaným projekce. Jedná se o rozdělení pole rychlostí proudění kapaliny na dvě složky. Kapalina, která je nestlačitelná, nemůže obsahovat žádné oblasti, ve kterých je vyšší či nižší tlak, ačkoliv v naší simulaci k tomu zatím dochází. Je třeba tedy identifikovat místa, ve kterých kapalina konverguje a diverguje. Zde nám pomůže zákon zachování hmotnosti. Pokud do dané buňky vtéká více kapaliny, než z ní vytéká, kapalina zde konverguje. Pokud naopak více vytéká, než vtéká, kapalina zde diverguje. Naším cílem je zajistit, aby pro každé místo platilo, že vtéká a vytéká stejné množství kapaliny. Získáme tedy dvě pole, jedno pole bez divergence a druhé pole bez působení hydrostatického tlaku. Výsledek pro naší buňku zahrnuje neznámé z okolních buněk, máme tedy mřížku x*y*z polí s x*y*z neznámými. Zde ze simulace stává aproximace. V této fázi se využívá iteračních řešení soustavy lineárních rovnic. V každé iteraci řešení se výsledek přibližuje reálnému výsledku, avšak přesného čísla nikdy nedosáhne. Přesnost simulace tak závisí na počtu iterací aproximace.
Dalším problémem je možná nestabilita simulace. V případě, že simulační krok by přeskočil vzdálenost jedné buňky v mřížce (kapalinu bychom tedy přesouvali z buňky x do buňky x+2), simulace začne oscilovat. Jelikož se jedná o aproximaci chování kapaliny, vždy bude obsahovat nějakou nepřesnost. S rostoucím časovým mezi výpočty ale roste i nepřesnost simulace. Proto je vhodné zvolit buď pevný časový krok pro simulaci nebo časový krok shora omezit. Viz. obr. 1.



Obr. 1 - Nepřesnost simulace vlivem vysokého časového kroku mezi výpočty

Nestabilita simulace vzniká také v procesu difúze kapaliny. V již zmíněném příkladu čaje dochází při difúzi nakonec k postupnému promíchání čaje ve vodě, až ve výsledku dojde k celkové rovnováze a v každé části vody je stejné množství čaje. Pro náš výpočet ale využijeme opak difúze - budeme postupovat zpět v čase. Pokud na konci difúze je ve vodě všude stejné množství čaje, na začátku simulace bylo v jednom bodě tolik čaje, kolik nyní je v celém systému. V případě naší simulace tedy budeme využívat pro výpočet aktuálního stavu stav difúze v následujícím stavu. Tímto se nám podaří odstranit nepřesnost vlivem

vysokého časového kroku mezi výpočty. Zde také využijeme iterativní výpočet soustavy lineárních rovnic.
Posledním krokem je tedy výpočet advekce. Vypočítáme tedy, kolik kapaliny z minulého stavu bylo přeneseno do jiné pozice v aktuálním stavu.

Simulace se tedy liší v závislosti na:

- použitém přístupu (mřížka / částice / obojí)
- způsob iterativního řešení soustavy lineárních rovnic

Implementace

Naše implementace využívá metody PIC/FLIP. Tato metoda kombinuje přístup mřížkových a částicových algoritmů k tomu, aby dosáhla dobře vypadajících výsledků a zároveň byla vysoce škálovatelná a paralelizovatelná. Tuto metodu je tak například možné simulovat i za použití compute shaderů na GPU (ačkoliv naše konkrétní implementace využívá pouze CPU).

Naším hlavním zdrojem informací o samotném interním fungováním metody PIC/FLIP se nám staly zejména zdrojové kódy cizích implementací, které se nám podařilo dohledat na GitHubu. Zde jsme se pak byli schopni seznámit se základními kroky této metody. Abychom se však ujistili, že algoritmu opravdu rozumíme, tak jsme se nejprve rozhodli pro implementaci referenčního 2D řešení v jazyce JavaScriptu (https://flipfluid.netlify.app/). Tato implementace se později stala velmi dobrým vzorem pro přepis do C++.

Hlavním principem PIC/FLIP je, že ačkoliv v základu reprezentuje tekutinu ve formě částic, tak na začátku každého snímku přenese jejich rychlosti do mřížky, kde pak probíhá většina simulace. To výrazně snižuje celkový počet přístupů do paměti a škálovatelnost celé simulace. Na konci snímku jsou pak rychlosti přeneseny z mřížky zpět do samotných částic a na základě toho proběhne jejich advekce.

Základní kroky simulace jsou:

1. Program iteruje přes veškeré částice a jejich rychlosti přenese do připravené mřížky. Na konci tohoto kroku tak rychlost uvnitř každé buňky reprezentuje průměrnou rychlost v jejím okolí.
2. Na buňky mřížky jsou aplikovány externí síly. V našem případě se jedná o gravitaci, která je přičtena ke všem buňkám.
3. Rychlosti uvnitř mřížky jsou upraveny tak, aby byly brány v potaz kolize s ohraničením celé simulace nebo jinými objekty uvnitř simulace.
4. Pro každou buňku simulace je vypočítána její divergence. Tato část je klíčová pro zachování konstantního objemu celé kapaliny.
5. Za pomocí Jacobiho iterativní metody se vypočítá divergentní část tlaku uvnitř tekutiny. Počet iterací lze upravit přes hodnotu UPROPERTY na actorovi AFluidSim.
6. Gradient tlaku je odečten od rychlostí uvnitř mřížky. Tím skončíme s rychlostmi, jejichž tlak má nulovou divergenci a zbývá nám tak pouze curl.
7. Nové rychlosti jsou přeneseny ze mřížky zpět do částic. V tomto kroku se objasňuje samotný název metody PIC/FLIP, jelikož nová rychlost částice se vypočítá jako lineární interpolace mezi PIC komponentou a FLIP komponentou rychlosti. Zatímco PIC komponenta je pouze aktuální rychlost v mřížce na stejné pozici jako kam tato

částice původně svou rychlost zapsala na začátku snímku, tak výpočet FLIP komponenty je o něco složitější. Zde se nejprve vypočítá rozdíl mezi aktuální rychlostí v mřížce a původní rychlostí v mřížce, takovou jaká byla na počátku snímku před aplikováním jakýchkoliv sil. FLIP komponenta pak vzniká přečtením tohoto rozdílu k původní rychlosti dané částice.

8. Nově přiřazené rychlosti částic jsou použity k jejich advekci.

Pokud jde o samotnou implementaci této metody v jazyce C++, tak ta je tvořena 3 různými třídami: actor třídou AFluidSim, actor třídou AParticle a obyčejnou C++ třídou FGrid3D.

FGrid3D je implementace trojrozměrného pole, které je používáno jako mřížka pro reprezentaci tekutiny při samotné simulaci. Reálně jde jednoduchý wrapper okolo pole TArray, který se stará o přepočet souřadnic ve formě FIntVectoru do obyčejného indexu jednorozměrného pole. Kromě toho však poskytuje také metody GetInterpolated() a AddInterpolated(), které se starají o trilineární interpolaci hodnot tak, aby bylo možné přistupovat také k necelým souřadnicím mřížky. To je pro fungování simulace zásadní, jelikož je potřeba správně zapsat a přečíst rychlosti pro částice, které mohou být na libovolných místech uvnitř buněk mřížky.

Actor AParticle reprezentuje částici tekutiny a z hlediska simulace si tato třída ukládá pozici a rychlost dané částice Interně se však stará také o samotnou vizualizaci částice ve formě koule a při přiřazení pozice má tak na starosti přepočet souřadnic z prostoru simulace do prostoru světa, které poté nastaví své vnořené StaticMeshComponent.

Actor AFluidSim je hlavním stavebním blokem celé simulace. Tato třída spravuje 2 instance UBoxComponent, které slouží jako vizualizace bounding boxu a spawn volume, dále TArray obsahující veškeré částice a konečně také veškeré instance mřížky FGrid3D, které jsou pro simulaci potřeba. Její metoda BeginPlay() se stará o naspawnování všech částic dovnitř oblasti definované spawn volume a také nastavení správných dimenzí všem instaním FGrid3D. Metoda Tick() pak obsahuje samotný kód simulace, tak jak byl popsán v tomto textu. Třída také poskytuje několik editovatelných hodnot UPROPERTY, pomocí kterých lze upravit chování simulace. Tyto hodnoty jsou:

- GridSize - velikost mřížky, použité při simulaci
- SpawnPosition - pozice spawn volume relativní k bounding boxu
- SpawnSize - velikost spawn volume
- CellSize - velikost jedné buňky mřížky v jednotkách UE (celkový bounding box simulace má automaticky velikost GridSize * CellSize)
- MaxDensity - maximální vyžadovaný počet částic uvnitř jedné buňky mřížky
- Gravity - velikost gravitace
- JacobiIters - počet iterací, které jsou použity při výpočtu tlaku
- FlipRatio - poměr, který je použit při lineární interpolaci mezi PIC a FLIP komponentami rychlosti

Závěr

Podařilo se nám vytvořit funkční implementaci tekutiny využívající metody PIC/FLIP běžící na CPU. Demo: https://youtu.be/Djtf9-wwafo

V budoucích iteracích by bylo možné přepsat implementaci do částicového systému Niagara uvnitř Unreal Engine a využít paralelismu GPU k více získání lepšího výkonu. Dalšími kroky by mohla být například implementace kolizcí s tuhými tělesy nebo rekonstrukce povrchu tekutiny.

Reference

https://shahriyarshahrabi.medium.com/gentle-introduction-to-fluid-simulation-for-programmers-and-technical-artists-7c0045c40bac
[1] Zhu, Yongning & Bridson, Robert. (2005). Animating Sand as a Fluid. ACM Trans. Graph.. 24. 965-972. 10.1145/1073204.1073298.
https://www.cs.ubc.ca/~rbridson/docs/zhu-siggraph05-sandfluid.pdf
[2] Stam, Jos. (2001). Stable Fluids. ACM SIGGRAPH 99. 1999. 10.1145/311535.311548.
https://dl.acm.org/doi/10.1145/311535.311548

Open-source implementace použité pro referenci:

- https://github.com/dli/fluid
- https://github.com/kbladin/Fluid_Simulation
- https://github.com/roberlozcar/PIC-FLIP
- https://github.com/ArtNlk/FlipSolver2d