SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1917

ANIMACIJA MODELA VODENIH POVRŠINA

Mario Volarević

Zagreb, lipanj 2011.

Sadržaj

1. Uvod

Vodene površine u svakodnevnom životu susrećemo svugdje, a kako u mnogim područjima gdje se koriste računala želimo što vjernije prikazati razne aspekte iz stvarnog svijeta onda je bilo potrebno pronaći način da se to učini i s vodenim površinama. Neka od mnogih područja kojima trebaju takve tehnike su video igre, stvaranje specijalnih efekata za filmove ili simulacije prilikom dizajniranja vodenih plovila. Kako je vrlo teško modelirati vodene površine na konvencionalne načine pomoću statičnih modela (kao što se modeliraju ljudi, zgrade ili vozila), iz jednostavnog razloga jer je vodena površina dinamična, stalno se kreće i naizgled je nepredvidiva, onda su za prikaz na računalu iskorištene fizikalne formule koje opisuju gibanje vode. Nekoć se na osobnim računalima nisu mogle koristiti prave fizikalne simulacije zbog nedostatka procesorske snage, nego su se koristile aproksimacije sa sinusom i kosinusom koje su bile brze i davale su zadovoljavajuće rezultate prilikom vizualnog prikaza, ali se takve formule nisu mogle koristiti za simulacije u znanosti gdje nije toliko bio bitan izgled nego ponašanje fluida. U znanosti su se koristile kompleksnije fizikalne formule koje su također davale aproksimaciju u odnosu na stvarni fluid, ali su te aproksimacije bile dovoljno dobre za praktične primjene. Kasnije, kako je razvoj računalnih komponenti napredovao, polako su se počele koristiti i fizikalne animacije na osobnim računalima, ispočetka jednostavnije, a kasnije sve kompleksnije, dok je u današnje doba na jačem (eng. *high-end*) osobnom računalu moguće izvoditi iznimno precizne fizikalne simulacije u stvarnom vremenu. Da bi se to postiglo uglavnom se iskorištava mogućnost paralelizma u algoritmima i programi se izvršavaju na modernim programibilnim grafičkim procesorima koji više odgovaraju za takav posao, a i njihov razvoj je duplo brže napredovao nego razvoj glavnih procesora. Postoji nekoliko tehnika za animaciju/simulaciju vodenih površina, koje će biti opisane u sljedećem poglavlju.

2. Fizikalna animacija fluida

Za animaciju vodenih površina potrebno je prvo proučiti načine animacije fluida jer voda, uz plinove (dim i vatru), fizikalno pripada kategoriji fluida, koja je još i uz to nestlačivi fluid što je jedan od bitnih uvjeta da bi se simulacija mogla ispravno izvesti na računalu. To je potrebno jer se zbog kompleksnosti određenih izračuna dijelovi formule koji opisuju fluid moraju zanemariti, a naječešće je to dio vezan za stlačivost. U slučaju da se simuliraju stlačivi fluidi, tj. plinovi, oni se također tretiraju kao nestlačivi, iako se po

potrebi simulira i to svojstvo, ali to uvodi poprilično dodatnu kompleksnost u simulaciju i ne primjenjuje se ako nije zaista potrebno. Stoga je moguće umjesto naziva voda koristiti općenitiji naziv fluid jer su im sva glavna svojstva zajednička, a razlikuju se samo u nekim estetskim parametrima prilikom implementacije.

2.1. Računalna dinamika fluida

Podloga za sve tehnike koje se koriste prilikom animacije ili simulacije fluida su Navier-Stokesove jednadžbe koje modeliraju ponašanje fluida na makroskopskoj razini. Naime, sve tehnike aproksimiraju u određenoj mjeri te jednadžbe koje opisuju fluid na temelju raznih parametara, kao što su viskoznost, tlak, brzina itd, čijom se interpretacijom dobiva gibanje ili neka druga svojstva fluida. Grana znanosti koja se bavi prilagođavanjem Navier-Stokesovih za upotrebu primjenjivu na računalu ovisno o području kojim se bavimo naziva se računalna dinamika fluida (eng. *Computational Fluid Dynamics, CFD*). Primarno stvorena za simulacije toka zračne struje prilikom razvoja avionskih krila, tijekom godina se razvila te se uz mehaniku fluida trenutno modeliraju kemijski i termodinamički procesi u fluidima, dodaju se modeli gibanja čestica unutar fluida, modeli biosustava (rast vodene flore i mikroflore i faune unutar vodotokova) te ima još mnoge druge primjene. No te tehnike iz *CFD* nisu mogle biti direktno iskorištene u računalnoj animaciji jer su ciljevi kojima se težilo bili različiti. Mnogi algoritmi i ideje su posuđeni, ali su morali biti prilagođeni jer su u računalnoj animaciji važniji vizualni rezultat i brzina izvođenja nego točnost simulacije, a osim toga je bilo potrebno prikazati i razne efekte koji su vidljivi u stvarnom svijetu prilikom interakcije s fluidima. Trenutno postoje tri metode koje se koriste u animaciji [5]. To su Lagrangeova metoda, Eulerova metoda i metoda Boltzmannove rešetke (eng. *Lattice Boltzmann Method, LBM*). Sve tri metode su zapravo različiti pristupi i gledišta istih zakona fizike, zakona o očuvanju mase i energije i 2. Newtonovog zakona te su zapravo te tri metode matematički ekvivalentne i moguće je jednu svesti na drugu.

2.2. Lagrangeova metoda

Glavna ideja ove metode je opis fluida pomoću velikog broja čestica, gdje su svakoj čestici pridruženi njen vektor brzine i položaj. Osim fluida ovom tehnikom se također mogu simulirati deformabilna tijela, tkanine ili drugi kaotični fenomeni. Takav sustav čestica je nejednolika diskretizacija kontinuiteta koji predstavlja fluid. Da bi se ta nejednolikost ublažila uvedena je metoda koja se naziva „zaglađena čestična

hidrodinamika“ (eng. *Smoothed Particle Hydrodynamics, SPH*). Ona definira funkciju koja interpolira razna fizikalna svojstva (tlak, temperaturu, brzinu, gustoću...) između susjednih čestica koje predstavljaju fluid. *SPH* je jedna od najpoznatijih i najčešćih metoda koja se koristi prilikom čestičnog prikaza fluida. Glavne prednosti čestičnog prikaza i to u slučaju kad se koristi *SPH* su:

- Olakšan rad s „nepravilnim“ (eng. *irregular*) granicama između fluida i objekta ili prilikom interakcija među različitim fluidima
- Lako je prikazati turbulentna prskajuća strujanja i sitne detalje kao što su mjehurići
- Općenito zahtjeva manje računalnih resursa od ostalih metoda

Glavni nedostaci su:

- Vrlo je teško napraviti dobru osnovnu funkciju (eng. *smoothing kernel*) u *SPH*, a o njoj ovise stabilnost, preciznost i brzina simulacije te je čak za različita svojstva fluida potrebno napraviti različite jezgre (eng. *kernel*)
- Ne može se garantirati nestlačivost fluida samo određenim ograničenjima nego je potrebno koristiti dodatne metode, ali se gubi prednost brzog izračunavanja
- Prilikom ostvarivanja prikaza (eng. *rendering*) teško je dobiti glatku površinu jer je površina stvorena od sitnih čestica, a vrijednosti na razmacima između su interpolirani te druge metode uglavnom daju bolje rezultate u tom slučaju

2.3. Eulerova metoda

Kod Eulerove metode se umjesto praćenja pojedinih čestica stvori fiksno višedimenzionalno polje (broj dimenzija ovisno o potrebi) te se unutar njega čuvaju podaci o raznim parametrima, kao što su brzina, tlak, temeperatura ili gustoća. Protjecanjem vremena se te vrijednosti mijenjaju prema fizikalnim zakonima. Ispočetka se Eulerova metoda koristila za uvelike pojednostavljene slučajeve Navier-Stokesovih jednadžbi jer je radila brzo i učinkovito, ali na taj način nije bilo moguće simulirati mnoge efekte. Daljnjim razvojem je omogućeno rješavanje potpunih Navier-Stokesovih jednadžbi u *3D* prostoru gdje je jedna od važnih ideja za postizanje stabilnosti bilo korištenje Lagrangeove metode u izrazu za konvekciju. Algoritam Eulerove metode se sastoji od četiri glavna koraka nakon kojih se dobiva vrijednost brzine za određeno mjesto u mreži. Ti koraci su: dodavanje sile, advekcija, difuzija i projekcija. Danas je Eulerova metoda toliko uznapredovala da se uz napredne tehnike vizualizacije dobivaju animacije koje nije

moguće razlikovati od stvarne scene, no i ona ima svojih nedostataka. Prednosti Eulerove metode su:

- Glatka površina fluida (uz odgovarajuće metode za prikaz)
- Veliki vremenski korak (vremenska tolerancija za izračunavanje)

Nedostaci su :

- Računski vrlo zahtjevna metoda
- Problemi s neravnim granicama, ćelije u mreži su kubne ili kvadratne te dolazi do aliasinga, postoje metode za ispraljanje, ali sve imaju svoje mane
- Loša skalabilnost (povećavanje mreže ne povećava zahtjeve za memorijom linearno nego kvadratno ili kubno, a vremenski još i više)

2.4. Metoda Boltzmannove rešetke

Metoda Boltzmannove rešetke (eng. *Lattice Boltzmann Method, LBM*) je opisana i predstavljena 1998. godine, što je čini poprilično novom metodom s obzirom da su ostale nastale u 80-im godinama. Ona se za razliku od drugih metoda ne zasniva direktno na Navier-Stokesovim jednadžbama nego se predočava modelom mikroskopskih čestica koje putuju po rešetki (stoga se i jednadžbe kojima su opisane nazivaju jednadžbe Boltzmannove rešetke, eng. *Lattice Boltzmann Equations, LBE*) čija prosječna vrijednost daje makroskopska svojstva koja zadovoljavaju Navier-Stokesove jednadžbe. Izveden je i dokaz kojim se *LBE* svodi na Navier-Stokesove jednadžbe. Kako je *LBM* moderna metoda ima mnogo prednosti prethodnih metoda s vrlo malo nedostataka. Prednosti su:

- Vrlo jednostavan algoritam za implementaciju i razumijevanje koji je u svojoj prirodi paralelan te ga je moguće prilagoditi za rad na GPU
- Nema problema sa složenim i nepravilnim granicama te miješanjem fluida
- Glavna formula je jednostavna i brzo se izračunava
- Može lako simulirati mezoskopska i mikroskopska svojstva fluida

Nedostaci su:

- Loša skalabilnost (isti slučaj kao Eulerova metoda jer se podaci čuvaju u mreži)

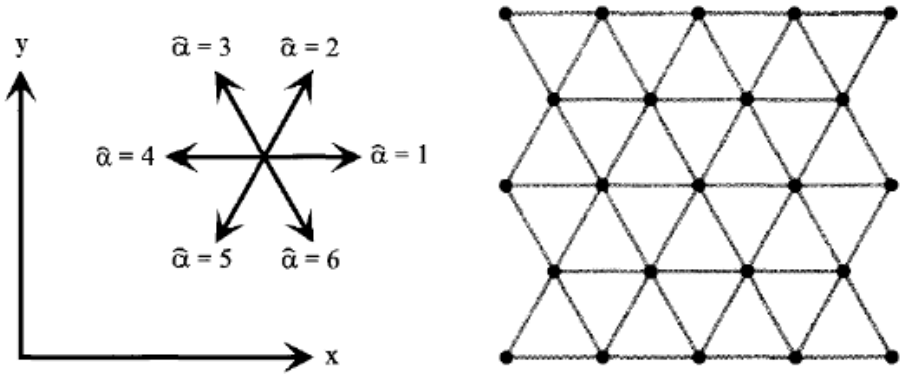
- Strog, tj. mali vremenski korak zbog kojeg može doći do nestabilnosti (iako nije toliki nedostatak jer se uglavnom jednadžbe vrlo brzo izračunavaju pa do ovog problema najčešće ne dolazi)

2.5. Odabir metode

Kako su neka od svojstava koje sam tražio kod metode za animaciju modela površine vode bile brz rad, mogućnost paralelizma i „jednostavna“ implementacija na GPU zbog eventualnog budućeg rada te dobro dokumentiran i jasno opisan algoritam, na temelju prethodno navedenih prednosti i nedostataka naposljetku sam se odlučio za *LBM*.

3. *LBM*

LBM se razvio iz *LGCA* (eng. *Lattice-Gas Cellular Automata*), statističkog modela koji je simulirao plin pomoću čestica koje su se nalazile na diskretnim pozicijama u pravilnoj kvadratnoj ili šesterokutnoj mreži [4]. Čestice plina su bile predstavljene Booleovim vrijednostima (na određenom mjestu u mreži čestica ili ima ili nema određeno svojstvo, kao što je npr. masa, ako nema mase, čestica ne postoji na tom mjestu), a te vrijednosti su se izračunavale i rasprostirale po mreži u određenim vremenskim intervalima i fiksnim smjerovima.



Slika 1. Smjerovi u šesterokutnoj *LGCA* mreži

Zakoni prema kojima su se čestice gibale bili su direktna kvantizacija Boltzmannovih jednadžbi u prostoru i vremenu. Ova metoda nije bila loša za prikaz fluida, ali sadržavala je previše šuma za praktičnu primjenu i bilo ju je teško prilagoditi za rad u više dimenzija. Kasnije je razvijena metoda gdje su se varijable s Booleovim vrijednostima zamijenile „pravim brojevima“ (za izračun su služile novo-uvedene distribucijske funkcije) te je

izmijenjena rešetka što je ispravilo sve mane *LGCA*, no time se uveo problem greške kod zaokruživanja racionalnih brojeva na računalima na koji treba pripaziti.

3.1. Fizikalna podloga

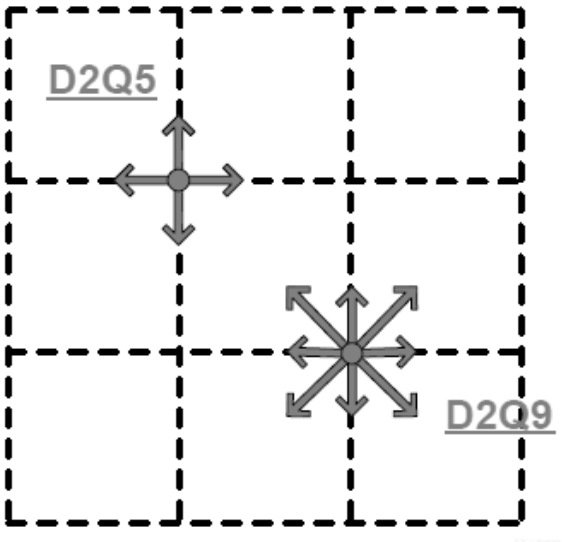
Kao što je prije navedeno *LBM* se temelji na Boltzmannovoj jednadžbi koja opisuje statističku distribuciju čestice unutar fluida i ona u kontinuiranom obliku glasi:

$$\partial_t f + \vec{v} \cdot \nabla f = \Omega(f) \qquad (1)$$

gdje f predstavlja vrijednost vjerojatnosti distribucijske funkcije, $\vec{v}$ je brzina čestice, a $\Omega(f)$ je operator kolizije (ili sudara, eng. *Collision Operator*) koji opisuje što se s česticom događa nakon sudara s drugim česticama. Da bi se ta jednadžba mogla upotrijebiti na računalu potrebno ju je diskretizirati. Ova jednadžba se diskretizira u prostoru i vremenu i po smjerovima u kojima se čestica može gibati:

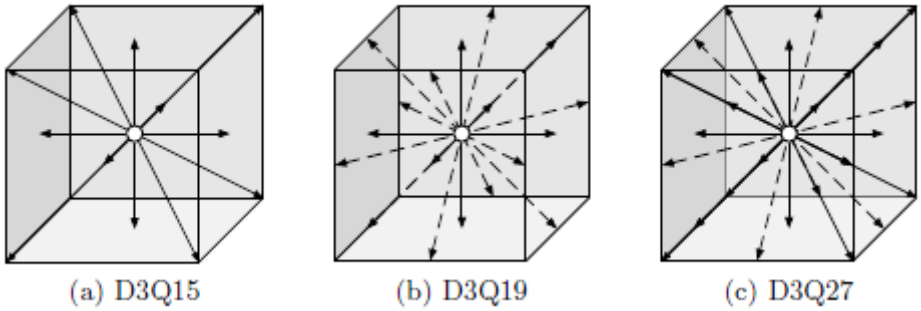
$$f_i(\vec{x} + \vec{e}_i \Delta t, t + \Delta t) = f_i(\vec{x}, t) + \Omega_i\big(f(\vec{x}, t)\big) \qquad i = 0,1, \dots, n \qquad (2)$$

gdje je f_i vrijednost distribucijske funkcije u smjeru i (broj mogućih smjerova je $n - 1$), $\vec{x}$ položaj čestice u mreži, a $\vec{e}_i$ vektor brzine koji opisuje u kojem se smjeru čestica giba. Kako se jednadžbe mogu riješavati u više dimenzija postoji konvencija kojom se označava koja se metoda koristi, tj. kako je oblikovana mreža.



Slika 2. Verzije *2D* geometrija mreže

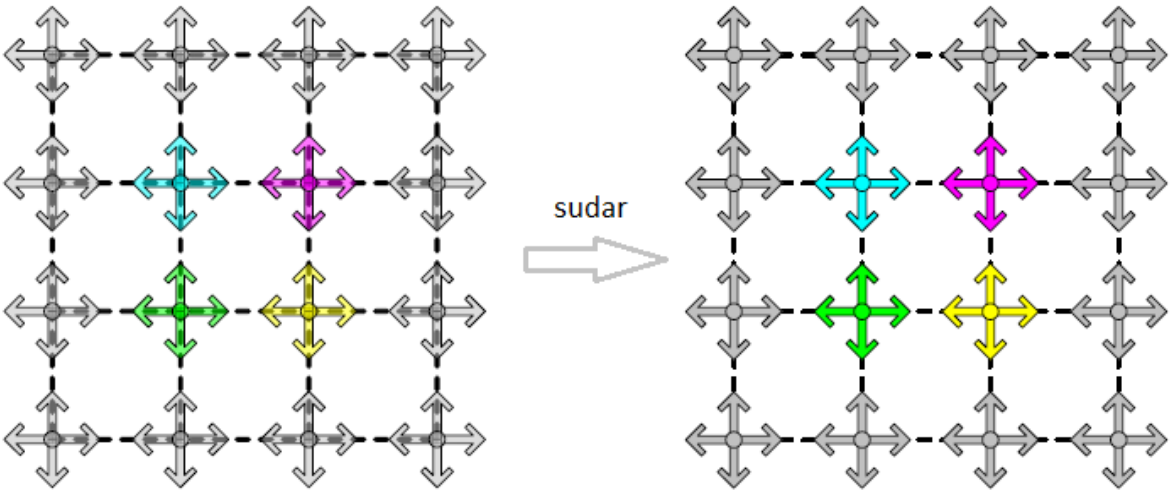
Oznaka se daje u obliku $DdQq$ (d označava broj dimenzija u kojima se odvija simulacija, a q broj smjerova u kojima se čestice mogu gibati), a ćelije je moguće zamišljati kao čvorove mreže ili kao prostore koje omeđuju rešetke u mreži.



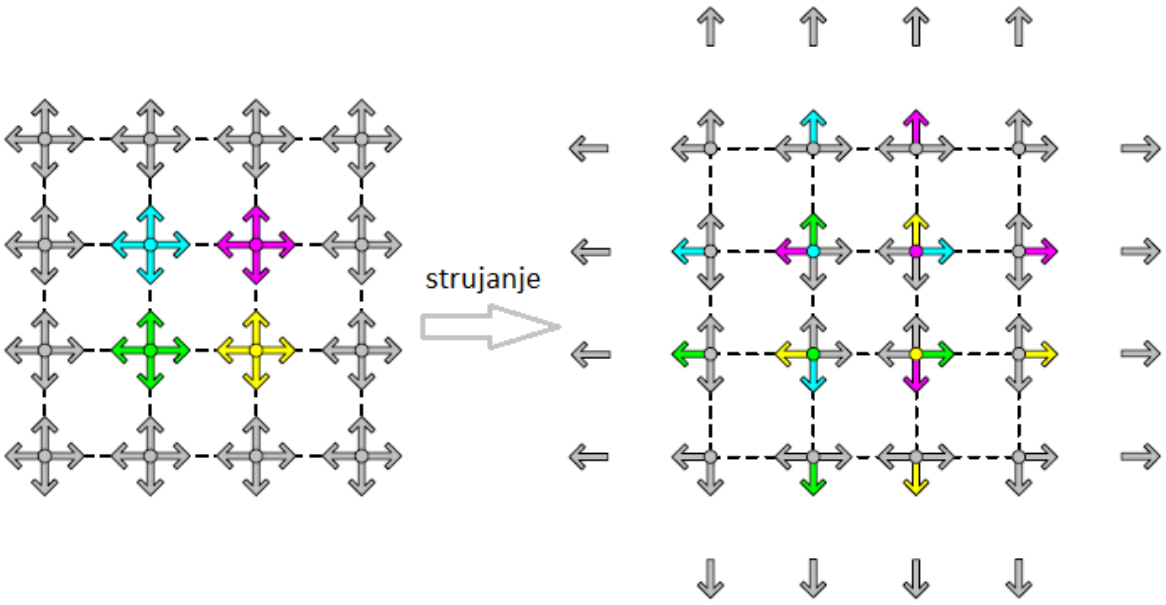
Slika 3. Verzije *3D* geometrija mreže

3.2. Opis algoritma

Sam algoritam se dijeli u dvije faze koje se nazivaju faza strujanja (eng. s*treaming*) i faza sudara (eng. *collision*). U fazi sudara se računaju nove vrijednosti svojstava čestice (u svakom čvoru za sve točke istovremeno) na temelju operatora kolizije, a u fazi strujanja se te vrijednosti propagiraju kroz rešetku, tj. mrežu. Na slici 4 prikazano je izračunavanje sudara na temelju vrijednosti iz prošle iteracije, a centralni čvorovi su istaknuti bojama radi označavanja koje vrijednosti pripadaju kojem čvoru (u ovom slučaju je vidljivo da položaj još nije promijenjen), kasnije, na slici 5, u fazi strujanja, moguće je vidjeti kako su se vrijednosti svake pojedine ćelije raširile po mreži. Nije bitno koja faza dolazi prva, bitno je samo njihovo naizmjenično izvršavanje.



Slika 4. Faza sudara



Slika 5. Faza strujanja

Za opis sudara molekula unutar fluida zadužen je operator Ω_i koji ima oblik:

$$\Omega_i = -\frac{1}{\tau}\left[f_i(\vec{x},t) - f_i^{eq}(\vec{x},t)\right] \tag{3}$$

u kojem je τ vrijeme relaksacije povezano s viskoznošću fluida, a f_i^{eq} funkcija koja opisuje stanje lokalne ravnoteže. U Boltzmannovoj jednadžbi nije moguće izračunati točno riješenje operatora kolizije i zato su za praktičnu primjenu Bhatnagar, Gross i Krook razvili aproksimaciju tog operatora 1954. godine te se stoga prema njima još naziva i *BGK operator*. U *LBM* je uveden 1992. za primjenu na računalima, a cilj tog operatora je navesti čestice da se približe ravnotežnom stanju prilikom sudara. Funkcija ravnotežnog stanja je u najčešćem slučaju opisana tako da simulira plitku vodu, koja se izvodi iz jednadžbi plitke vode (eng. *Shallow Water Equations, SWE*). Razlog je taj što se zbog loše skalabilnosti *LBM* rijetko koristi za modeliranje velikih volumena vode. U diskretnom obliku funkcija ravnoteže izgleda ovako:

$$f_i^{eq} = \omega_i \rho \left(1 - \frac{3}{2}\vec{u}^2 + 3(\vec{e_i}\cdot\vec{u}) + \frac{9}{2}(\vec{e_i}\cdot\vec{u})^2\right) \tag{4}$$

gdje ω_i predstavlja konstantu koja opisuje udjel određenog smjera u mreži (još se koristi i naziv težinski faktori, vrijednosti za različite geometrije su vidljivi u tablici 1, gdje se može primijetiti da najveću težinu nosi centar ćelije), ρ je makroskopska vrijednost gustoće, a $\vec{u}$ je makroskopska vrijednost brzine.

Tablica 1. Težinski faktori različitih geometrija

	i	$\rightarrow\omega_i$		i	$\rightarrow\omega_i$
D2Q9	$i = 0$	$\rightarrow\omega_i = \frac{4}{9}$	D3Q15	$i = 0$	$\rightarrow\omega_i = \frac{2}{9}$
	$1 \leq i \leq 4$	$\rightarrow\omega_i = \frac{1}{9}$		$1 \leq i \leq 6$	$\rightarrow\omega_i = \frac{1}{9}$
	$5 \leq i$	$\rightarrow\omega_i = \frac{1}{36}$		$7 \leq i$	$\rightarrow\omega_i = \frac{1}{72}$
D3Q27	$i = 0$	$\rightarrow\omega_i = \frac{8}{27}$	D3Q19	$i = 0$	$\rightarrow\omega_i = \frac{1}{3}$
	$1 \leq i \leq 6$	$\rightarrow\omega_i = \frac{2}{27}$		$1 \leq i \leq 6$	$\rightarrow\omega_i = \frac{1}{18}$
	$7 \leq i \leq 14$	$\rightarrow\omega_i = \frac{1}{216}$		$7 \leq i$	$\rightarrow\omega_i = \frac{1}{36}$
	$15 \leq i$	$\rightarrow\omega_i = \frac{1}{54}$			

Ove posljednje dvije vrijednosti se koriste prilikom vizualizacije i simulacije te su njihove vrijednosti ekvivalentne vrijednostima za gustoću i brzinu koje bi se dobile kad bi se riješavale Navier-Stokesove jednadžbe. Njihova vrijednost za pojedino polje u mreži se dobiva kad se naprave prosjeci distribucijske funkcije po svim smjerovima:

$$\rho = \sum_{i=0}^{n-1} f_i \tag{5}$$

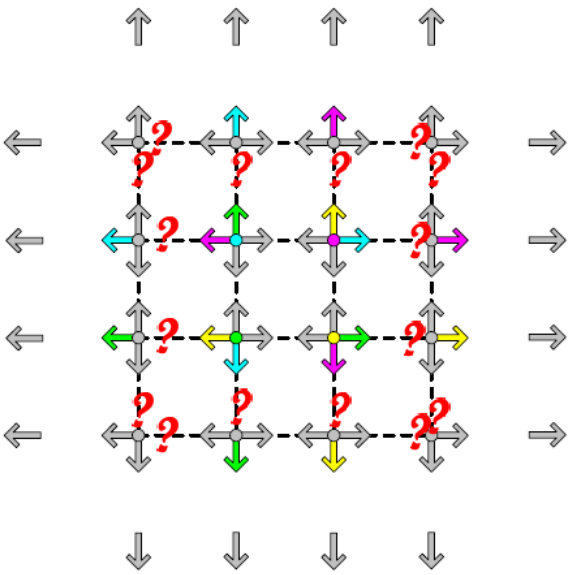
$$\vec{u} = \frac{1}{\rho}\sum_{i=0}^{n-1} f_i \vec{e}_i \tag{6}$$

U konačnoj jednadžbi možemo vidjeti koji su dijelovi zaduženi za fazu strujanja, a koji za fazu sudara, još je uvedena i vrijednost S_i koja odgovara vanjskoj sili:

$$\underbrace{f_i(\vec{x} + \vec{e}_i \Delta t, t + \Delta t) = f_i(\vec{x}, t)}_{\textbf{\textit{Faza strujanja}}} \underbrace{- \frac{1}{\tau}\left[f_i(\vec{x}, t) - f_i^{eq}(\vec{x}, t)\right]}_{\textbf{\textit{Faza sudara}}} + S_i \tag{7}$$

3.3. Granični slučajevi

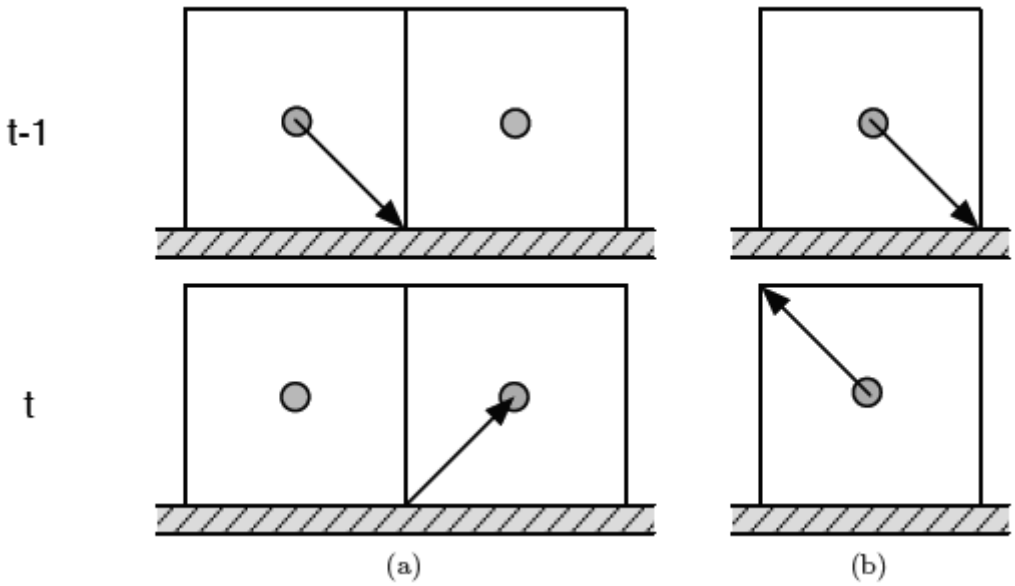
Kada je potrebno obavljati izračune i izvoditi simulaciju uz rub mreže ili uz prepreke (tj. na različitim granicama) potrebno je uvesti posebna pravila, jer nam u tim slučajevima nisu poznate vrijednosti fluida koje dolaze „izvana“. Na slici 6 se vidi gdje su nepoznata područja.



Slika 6. Granični problemi

Postoje tri osnovna načina implemenacije granice koje je moguće upotrijebiti u tom slučaju. To su:

1. **Zatvorena granica** (eng. *Closed boundary*): Zatvorene granice se ostvaruju tako da se primijeni pravilo odbijanja. Kada čestica dođe do ruba odbije se i nastavi dalje u suprotnom smjeru. Postoje dva načina odbijanja slobodno-klizni (eng. *free-slip*) i ne-klizni (eng. *no-slip*).



Slika 7. Primjer: a) *free-slip* i b) *no-slip* odbijanja

Princip rada obe tehnike je opisan na slici 7, a primjer formule za *no-slip* odbijanje je:

$$f_{-i}(\vec{x}, t+1) = f_i(\vec{x}, t) \tag{8}$$

2. **Otvorena granica** (eng. *Out-flow boundary*): Tok koji prelazi u područje izvan mreže se odbacuje, a u posljednju ćeliju se dodaje prosječna količina izgubljenog fluida. To je potrebno da se ne naruši zakon o očuvanju mase. U formuli koja opisuje takvo ponašanje funkcija ovisi o položaju, a N_z predstavlja broj ćelija u određenoj dimenziji (u ovom, 3D slučaju, predstavlja *z* koordinatu):

$$f_i(i, j, N_z - 1) = f_i(i, j, N_z) \tag{9}$$

3. **Periodična granica** (eng. *Periodic boundary*): Fluid koji na jednoj strani izlazi iz mreže pojavljuje se na suprotnoj strani s jednakim iznosom i smjerom. Takvo ponašanje je opisano formulom gdje članovi imaju isto značenje kao u prethodnom slučaju:

$$f_i(i, j, 0) = f_i(i, j, N_z - 1) \tag{10}$$

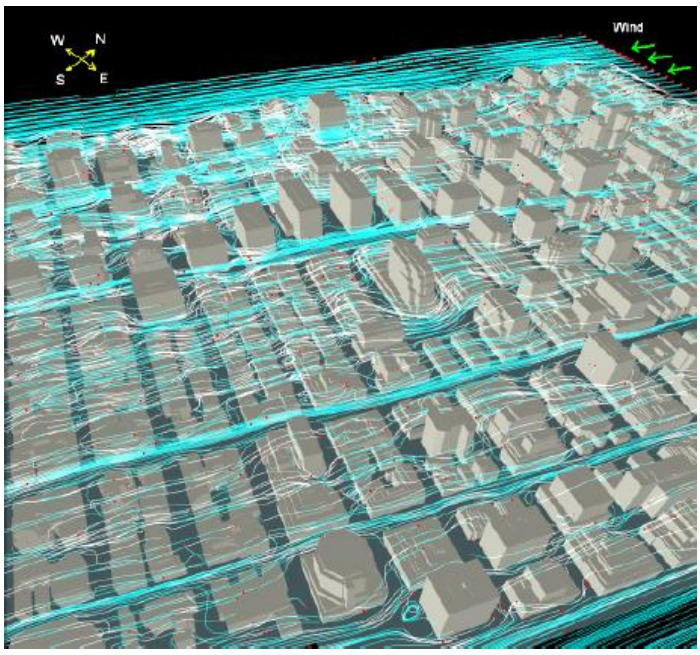
U slučaju nepravilnih ili zakrivljenih granica najčešće se radi linearna interpolacija vrijednosti između susjednih čvorova iako postoje i drugi napredniji pristupi.

3.4. Varijante *LBM*

Osim podjele *LBM*-a prema broju dimenzija i smjerova koje sadrži mreža, što se označava nomenklaturom *DdQq,* postoji još i klasifikacija prema vremenu relaksacije τ na *LBM* s jedinstvenim vremenom relaksacije (eng. *Single-Relaxation-Time LBM, SRTLBM*) i na *LBM* s višestrukim vremenom relaksacije (eng. *Multiple-Relaxation-Time LBM, MRTLBM*) [3]. Glavna značajka *SRTLBM* je kao što ime kaže jedno vrijeme relaksacije. Osim toga simulaciju je jednostavno implementirati i brzo radi, ali je nedostatak nestabilnost prilikom simulacije visokoturbulentnih fluida. *MRTLBM* je općenitiji slučaj, teže je implementirati simulaciju, ali često radi brže i stabilnije od *SRTLBM*. Različiti parametri relaksacije se koriste za kontrolu fizikalnih svojstava (masa, moment, energija, viskoznost). Također postoji i termalna verzija *LBM* koja može simulirati razne efekte vezane za prijenos topline kao što su kondukcija među krutinama, konvekcija i radijacija. Prethodno opisane formule su pojednostavljeni slučaj *LBM*-a koji ne sadrži termalnu komponentu.

3.5. Primjene *LBM*

Broj područja u kojima se *LBM* primjenjuje je iznimno velik i nastavlja rasti. Stalno se nalaze nove metode za simulacije stvari koje prije nisu bile moguće i ostvaren je iznimno velik napredak u posljednjih 20 godina otkad se *LBM* počela upotrebljavati. Osim simulacije toka i površine vode ili drugih fluida moguće je simulirati realistično ponašanje vatre i dima. Koristi se za vizualizaciju kretanja mjehura na vjetru te deformacije i oscilacije koje se pritom događaju. Također može opisati taljenje i prelijevanje preko raznih objekata. Kao što je prije spomenuto koristi se i u termalnim simulacijama prilikom interakcija voda-zrak, krutina-krutina, krutina-zrak te pritom može opisati razne efekte kao što su ljeskanje površine (eng. *shimmering*) ili fatamorgana. U znanstvene svrhe se koristi prilikom predviđanja rasprostiranja opasnih čestica u struji zraka u urbanim sredinama gdje je ta struja iznimno dinamična i nepredvidiva. Te čestice mogu biti nuklearne padaline ili razni virusi ili bakterije. Uz to može simulirati i rasprostiranje topline na stvarnim lokacijama uz detaljne modele područja. [3]



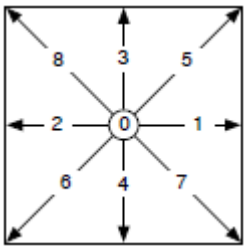
Slika 8. Simulacija strujnica na *Time Square-u*

Kako *LBM* nije samo metoda za simulaciju fluida nego je i iznimno brz i učinkovit *solver* za parcijalne diferencijalne jednadžbe postoji još mnogo drugih korisnih primjena. Neke od njih su obrada slike gdje je gustoća pixela parametar *LBM*, zaglađivanje volumena gdje je parametar gustoća voxela, pomoću nje je moguće i uklanjanje šuma na površini objekta ili izobličavanje objekata (eng. *morphing*). Osim toga lako se prilagodi za rad na *GPU* i ubrzanja su fascinantna, čak i do 200 puta brže nego izvršavanje na *CPU*. Nažalost, jedno

od bitnih ograničenja *LBM* je nemogućnost rješavanja tokova s visokim *Mach* brojem koja se koristi u aerodinamici te je za to potrebno koristiti druge metode, ali se radi na tehnikama da se *LBM* osposobi i za tu primjenu.

4. Implementacija

Kako je moj zadatak bio jednostavna animacija modela vodene površine zasnovana na fizikalnim zakonima, odlučio sam se se za dvodimenzionalnu atermalnu verziju *LBM*-a s jednim vremenom relaksacije. Do zaključka da mi je takva specijalizacija sasvim dovoljna za postizanje rezultata sam došao proučavanjem radova drugih osoba koje su se bavile istom tematikom.



Slika 9. Smjerovi moje implementacije *D2Q9*

Dvodimenzionalna verzija koju sam koristio klasificira se kao *D2Q9*, a smjerovi koje sam koristio su navedeni na slici 9. Zbog nedovoljnog znanja programiranja grafičkog procesora simulaciju sam implementirao na *CPU*, a ne na *GPU*. Za pisanje programa koristio sam programski jezik *C#* u razvojnoj okolini *Microsoft Visual Studio 2010*, a za vizualizaciju sam koristio *OpenGL* i *GLUT* koji su dio *TaoFrameworka* za *C#*.

4.1. Struktura programa

Program je podijeljen u četiri klase, od kojih su dvije povezane s grafičkim sučeljem i inicijalizacijom scene, a druge dvije uz simulaciju. Naglasak će biti na klasama i metodama vezanima za simulaciju. Klasa koja se naziva *LBM.cs* sadrži metode koje su zadužene za izvođenje glavnog dijela algoritma simulacije, izračune prilikom interakcije s korisnikom te se u toj klasi nalaze metode za vizualizaciju. U drugoj klasi vezanoj za simulaciju naziva *Cell.cs* nalaze se podaci i metode koje su vezane za pojedinu ćeliju u mreži.

4.1.1. Klasa *LBM.cs*

U konstruktoru klase inicijaliziram s potrebnim dimenzijama liste i polja koja koristim, ovisno o veličini mreže. Mrežu mi predstavlja dvodimenzionalna lista ćelija koje su tipa *Cell*. Glavna metoda za izračun se naziva *StreamCollide* i u njoj se odvijaju dvije glavne faze, *Streaming* i *Collision*. Algoritam se izvršava tako da se iterira kroz dvostruku petlju kojom se prolaze sve ćelije u mreži, a zatim se ulazi u još jednu petlju gdje se izračunavaju vrijednosti za svaki pojedini smjer. Prije ulaska u petlju s mogućim smjerovima osvježavaju se podaci o funkciji ravnoteže za pripadnu ćeliju. U mom slučaju prvo ide faza sudara koja je opisana ovako:

```
double temp = nc.fi[k] - r_tau * (nc.fi[k] - nc.feq_i[k]);
```

koja je izgledom ista kao i fizikalna formula (7) bez člana za vanjsku silu. Objekt koji predstavlja trenutnu ćeliju se naziva `nc`, funkcija distribucije koja odgovara pojedinom smjeru je `fi[k]`, a funkcija lokalne ravnoteže je `feq_i[k]`. S `r_tau` sam označio recipročni parametar vremena relaksacije koji je povezan s viskoznošću. Prema navedenom logično je zaključiti da varijabla `k` predstavlja smjer.

U fazi strujanja u kojoj se odvija propagacija vrijednosti kroz mrežu ima malo više posla. Potrebno je odrediti koje ćelije su susjedne te zatim u njih upisati nove vrijednosti. U ovom koraku riješavam i granične slučajeve gdje koristim *bounce back no-slip* tehniku.

```
Vector np = cp + va[k];
int nX = (int)np.X;
int nY = (int)np.Y;
if (k != 0 && (nX < 0 || nY < 0 ||
    nX >= lattice.Count ||
    nY >= lattice[i].Count))
{
    if (clk == 0)
        nn = temp_lattice[i][j];
    else nn = lattice[i][j];
    int t = k;
    if (t % 2 == 0) t -= 1;
    else if (t % 2 != 0) t += 1;
    nn.fi[t] = temp;
}
else
{
    if (clk == 0)
        nn = temp_lattice[nX][nY];
    else nn = lattice[nX][nY];
    nn.fi[k] = temp;
}
```

Za spremanje položaja koristim vektor, novi položaj izračunavam na temelju vektora brzine koji odgovara pripadnom smjeru. Kasnije će biti objašnjeno zašto se koriste dvije mreže i varijabla `clk`.

Sada je još potrebno izračunati novu vrijednost ćelije u slučaju da je na toj lokaciji primjenjena sila. Za to koristim prethodno prikazan fizikalni izraz proširen varijablom *S*:

```
nn = lattice[x][y];
nc = temp_lattice[x][y];
nn.fi[k] = nc.fi[k] - r_tau * (nc.fi[k] - nc.feq_i[k]) + S;
```

Efekt sile se ostvaruje povećanjem vrijednosti funkcije distribucije ravnomjerno u svim smjerovima čime se indirektno mijenjaju brzina i gustoća te se na temelju te promjene izračunava nova vrijednost funkcije ravnoteže u sljedećoj itearciji, koja se propagira dalje kroz mrežu.

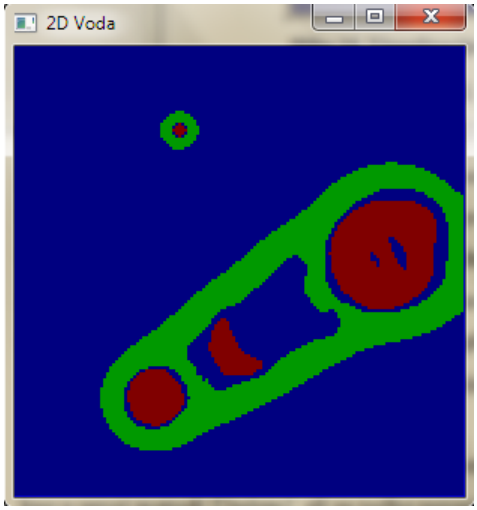
U ovoj klasi postoje i dvije metode za vizualizaciju koje se nazivaju *Update1* i *Update2*. U ovim metodama se odvija poziv metode za osvježavanje makroskopskih podataka gustoće i brzine koji su mi potrebni za računanje boje. U obe metode koristim točke (iz *OpenGL-a* GL_POINTS) za prikaz površine vode. Svaka točka predstavlja jednu ćeliju. Da bi smanjio zahtjev za računalnim resursima svaku ćeliju sam vizualizirao s *pixelima* veličine *2x2* čime dobivam površinu prozora četiri puta veću od površine mreže *LBM*-a.

Prva metoda (Slika 10.) pokušava dočarati „realističan“ prikaz površine vode s pogledom iz tlocrta pomoću točaka različite nijanse sive boje i α-komponente za prozirnost. Parametar koji određuje boju ovisi o vrijednosti gustoće u ćeliji. Napravljena je linearna ovisnost o gustoći gdje je za početni slučaj (početnu mirnu površinu vode kada je gustoća *1000*) vrijednost parametra na polovici mogućeg raspona. Točke se iscrtavaju preko poligona plave boje te je stoga i površina plava jer točka sadrži prozirnu komponentu.

Slika 10. Vizualizacija metode *Update1*, odvod isključen/uključen

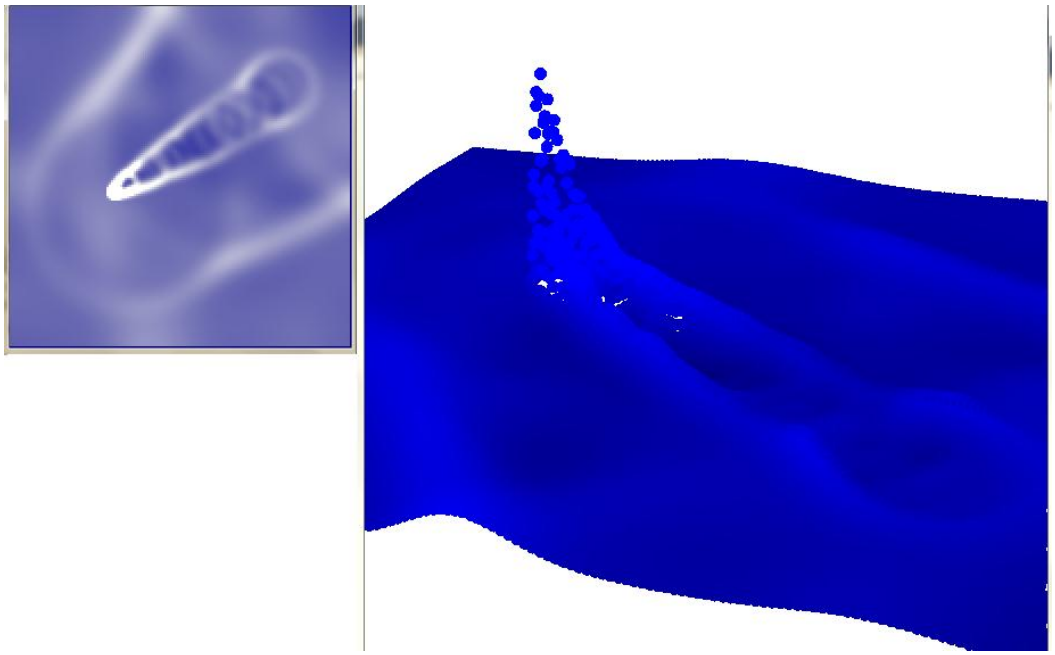
Druga metoda (Slika 11.) služi za vizualnu provjeru je li površina vode mirna jer je u prvoj metodi teško vidjeti male fluktuacije na površini. U ovoj metodi računam prosječnu gustoću površine te zatim provjeravam je li gustoća u određenoj ćeliji veća ili manja od

nje. Ako je veća onda se ta točka iscrtava zeleno, inače crveno, a ako je blizu prosjeka onda je plava. Osim za vizualizaciju, ove metode su zaslužne i za obavljanje funkcionalnosti kada je aktivna funkcija odvoda. Odvod se ponaša tako da po želji smanjuje gustoću na određenom mjestu u mreži. To je potrebno jer katkad kompletna gustoća mreže prilikom rada programa naraste toliko da iziđe iz raspona koji je moguće prikazati linearnom funkcijom za bojanje.



Slika 11. Vizualizacije metode *Update2*

Kasnije je dodan *3D* prikaz fluida gdje se površina iscrtava na sličan način pomoću točaka kao u prvoj metodi *Update1*, ali su točke povećane i gustoća se osim za izračun boje koristi za z-koordinatu položaja točke, tj. koliko će ta točka kao dio vala biti visoko ili nisko. Time se na „jeftin“ način dobivaju atraktivni rezultati.



Slika 12. Vizualizacija *2D* i *3D*

4.1.2. Klasa *Cell.cs*

Ova klasa obavlja funkcionalnost osvježavanja podataka vezanih za pojedini čvor, tj. ćeliju u mreži. U njoj se nalazi metoda za izračun funkcije ravnoteže:

```
feq_i[k] = wi*RO*(1 - 1.5 * (U*U) + 3 * (ei[k]*U) + 4.5 * (ei[k]*U) * (ei[k]*U));
```

Kao što možemo vidjeti izgled ove funkcije je identičan fizikalnom modelu prema kojem je nastao. Kako ova funkcija ovisi o gustoći i brzini, potrebno je inicijalizirati te vrijednosti da bi funkcija u prvoj iteraciji imala s čim računati. Te vrijednosti postavljam na *1000* za gustoću i *(0, 0)* za brzinu što na početku rezultira s mirnom površinom vode. U ovoj klasi se čuvaju podaci o gustoći i brzini pa se zato tu nalazi i metoda koja te podatke računa na temelju vrijednosti distribucijske funkcije:

```
Vector tempU = new Vector(0, 0);
RO = 0;
for (int i = 0; i < 9; i++)
{
    RO += fi[i];
    tempU += fi[i] * ei[i];
}
U = tempU / RO;
```

Ta metoda se naziva *calcRoU* i u istom prolazu računa i gustoću i brzinu, ali u suštini je jednaka kao i fizikalne formule (5) i (6).

4.2. Optimizacije

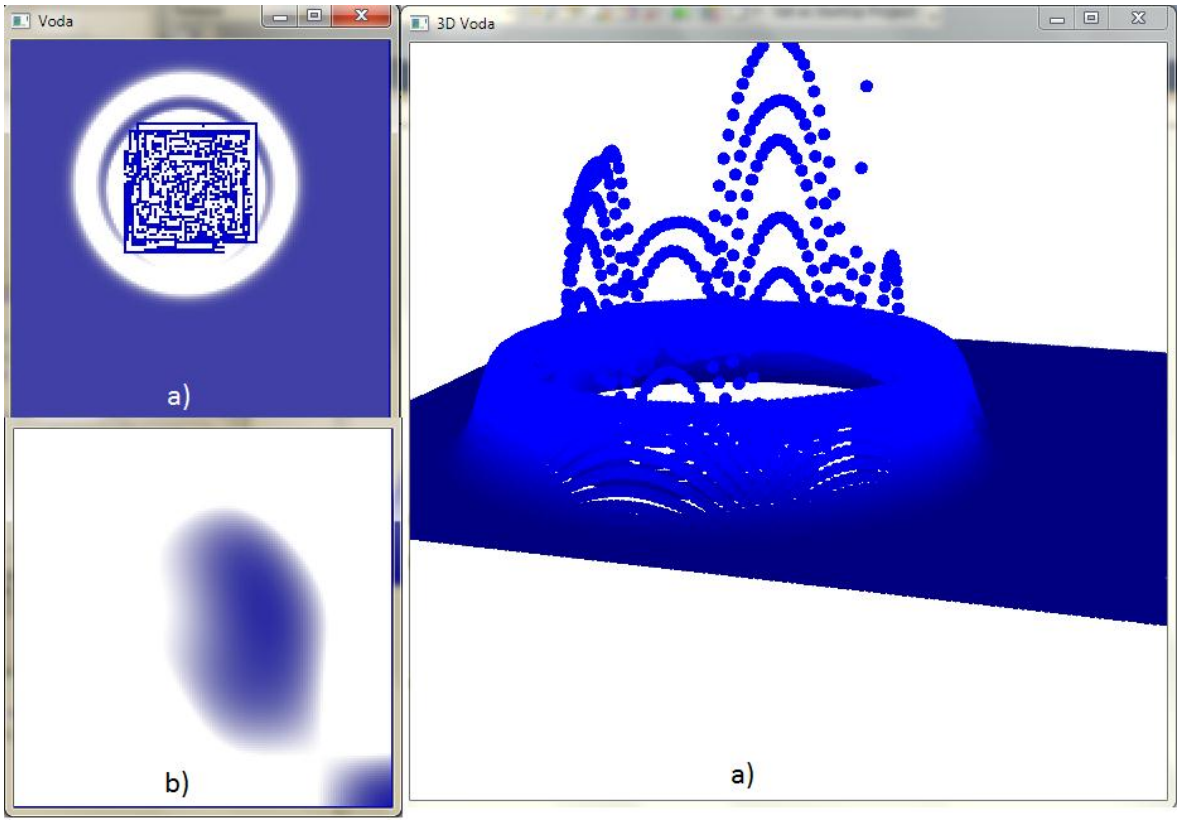
Glavni cilj kojem sam težio prilikom pisanja programa je bio ostvariti što veću brzinu izvođenja simulacije. Postojalo je nekoliko verzija programa, a svaka je bila brža od prethodne. Uglavnom sam mijenjao strukturu metoda u klasi *LBM.cs*. Prvo sam program napisao direktno onako kako je bio opisan formulama da se upoznam s principom rada algoritma, a kasnije sam ga modificirao. U početnoj verziji sam koristio jednu dvodimenzionalnu listu i dvije petlje. Svaka petlja za jednu fazu, ali ovakav način nije radio kako treba. Faza sudara bi se dobro izračunala, ali kad bih u fazi strujanja zapisivao podatke na nova mjesta u ćeliji, stari podaci bi na novim mjestima bivali prepisani. Stoga sam morao dodati još jednu dvodimenzionalnu listu. Sad sam podatke od sudara zapisivao u jednu, a prilikom strujanja sam ih zapisivao u drugu listu. Kad bi iteracija završila zamijenio bih im uloge, tj. u sljedećoj iteraciji bih čitao podatke iz one liste gdje su „svježiji“ podaci, u ovom slučaju to je lista koja je u prethodnoj iteraciji služila za strujanje. Sada bih u fazi sudara pročitao podatak iz te liste, izračunao novu vrijednost i ponovno ga spremio na istu lokaciju (u ovoj fazi mi ne smeta prebrisivanje starog podatka

jer sam ga već iskoristio). Zatim sam te podatke zapisivao u mrežu koja je u prošloj iteraciji bila „mreža za sudare“. Sljedeća optimizacija je bila izbjegavanje dvije petlje. U jednoj petlji sam obavljao i fazu sudara i strujanja. Također sam koristio dvije liste i mijenjao sam im uloge, ali kako su sad koraci bili spojeni, nisam morao čuvati podatak od sudara u mreži nego sam ga odmah mogao zapisati na poziciju koja odgovara onoj koja se izračuna u fazi strujanja. Ne znam koliko sam tada dobio ubrzanje jer u ranim fazama razvoja nisam pratio brzinu izvođenja, ali sam primijetio da je iz „trzave“ animacije prešlo u glatku animaciju što znači da je prethodno brzina bila ispod 24 *FPS-a*, a kasnije je prešla tu granicu, tj. za mrežu 150x150 se dobiva rezultat od 30 *FPS-a*. Za dobivanje vrijednosti *FPS-a* sam koristio komercijalno dostupan program koji se naziva *Fraps* u besplatnoj inačici. Za njega sam se odlučio jer u zanemarivoj mjeri utječe na performanse programa prilikom mjerenja. Nakon toga sam otkrio da za *C#* u *.NET Frameworku v4.0* postoji paralelna for petlja koju sam mogao iskoristiti u fazi sudara i strujanja jer sam zadržao lokalnost izračuna. Time mi je *FPS* narastao na 43 *FPS-a* na dvojezgrenom procesoru za istu mrežu. Posljednja optimizacija koju sam napravio je smanjenje broja poziva prema grafičkoj kartici korištenjem polja *vertex-a* (eng. *Vertex Array*) umjesto pojedinačnih poziva za svaki *vertex.* Time mi je broj *FPS-a* narastao na 51. Kasnijim dodavanjem *3D* prikaza *FPS* pada na 46 što smatram prihvatljivom vrijednošću za mrežu navedene veličine. Zbog ljepšeg prikaza sam aktivirao i *anti-aliasing*, ali on nije uzrokovao pad performansi. Stoga pretpostavljam da je u ovoj simulaciji procesor usko grlo jer grafička kartica brže obrađuje podatke nego što ih on šalje što je i logično jer scena koju je potrebno iscrtati nije kompleksna, sastoji se samo od točaka različitih boja.

4.3. Problemi

Prilikom razvoja programa susreo sam se s nekoliko problema. Prvi se pojavio u u početcima pisanja programa kada su mi se neispravno prepisivale vrijednosti u mreži prilikom faze strujanja, što je kasnije ispravljeno s dvije mreže i njihovom naizmjeničnom zamjenom uloga što je detaljnije opisano u prethodnom poglavlju. Najveći problem na koji sam naišao i koji je još uvijek u određenoj mjeri prisutan je nalaženje dobrog načina za vizualizaciju tekućine. U ranim fazama razvoja programa odlučio sam da će tekućina biti predočena točkama jer je vizualizacija simulacije trebala biti dvodimenzionalna, ali trebalo je odlučiti kakvu formulu izabrati za ovisnost boje o gustoći da se dobiju vizualno atraktivni rezultati koji bi doprinijeli realizmu scene. Tijekom simulacije na površini vode događaju se velike i male promjene razine vode (valovi), a sve trebaju biti vidljive, no kad

se izabere formula koja dobro prikazuje male promjene onda se velike promjene loše prikazuju i obrnuto. Stoga je potrebno naći kompromis, ali tada dolazi do drugog problema koji se javlja kod bojanja. Naime, dodavanjem novih kapljica povećava se gustoća ukupne površine vode, a kako boja linearno ovisi o gustoći i ima određenu maksimalnu vrijednost koja je valjani ulaz funkcije za bojanje u *OpenGL-u* nakon određenog broja kapljica, kada ih se doda previše, ta maksimalna vrijednost se prijeđe i u nastavku izvođenja programa se koristi stalno ista boja (Slika 13b). To je djelomično popravljeno dodavanjem funkcije odvoda koja smanjuje vrijednost gustoće, a time i boju na razinu ispod maksimalne. Drugi veliki problem koji se pojavljivao prilikom simulacije se manifestirao kada bih koristio velike kapljice koje su u formuli rezultirale jako velikom silom te bi mi vrijednost distribucijske funkcije postala negativna, čime bi i gustoća postala negativna što je u stvarnom životu nemoguće.



Slika 13. Problemi: a) negativna gustoća; b) zasićenje boje

Time bi se pojavio nepravilan poremećaj (Slika 13a) koji bi se proširio po cijeloj površini i upropastio simulaciju jer je sam sebe pojačavao zbog funkcije ravnoteže koja je u ovom slučaju imala suprotan učinak od željenog. Taj problem je ispravljen dodavanjem provjere je li rezultat funkcije distribucije negativan te ako jest, tada joj se vrijednost postavlja na 0. Takvo ponašanje je analogno onom iz stvarnog života kada u posudu s vodom bacimo veliki kamen i početna najveća dolina vala udari u dno posude.

5. Budući razvoj

Logičan nastavak razvoja ovog programa je povećanje brzine, realističniji prikaz i proširenje mreže za još jednu dimenziju da simulacija postane *3D* čime dobivamo mogućnost simuliranja više vodenih efekata. Za povećanje broja dimenzija je nužno povećanje brzine da bi dobili prihvatljive rezultate, a to se u slučaju ovakvog alogritma postiže prepravljanjem programa za izvođenje na grafičkom procesoru s obzirom da algoritam može iskoristiti masivni paralelizam koji *GPU* pruža. Ova metoda se primarno i počela koristiti zbog takve mogućnosti i prve praktične primjene su bile implementirane odmah na *GPU*, a ne *CPU*. Implementacija na *CPU* je korištena samo kao mjerilo performansi, a ne kao program namijenjen ozbiljnom radu. Za realističniji prikaz bi se mogli napisati programi za sjenčanje (eng. *shader*) koji bi dodali efekte poput refleksije, refrakcije ili kaustike čime bi virtualnu scenu bilo teško razlikovati od stvarne, a ne bi previše utjecali na performanse jer se također izvršavaju na *GPU*, naravno, u slučaju da se koristi grafička kartica dovoljne snage, ali uz današnji razvoj za to je dovoljna kartica za stolna računala iz srednjeg segmenta. Primjer vizualizacije programa koji koristi sve navedene efekte te kojem je osnova *LBM* prikazan je na slici 14.

Slika 14. Realistična površina vode

6. Zaključak

Zaključak do kojeg dolazim je da je *LBM* iznimno korisna metoda koja ima masu primjena van standarnih okvira metode za simulaciju fluida, ali nije svemoguća. No, kako se koristi i u komercijalne primjene da programi koji je koriste ne bi zastarijeli potrebna su buduća proširenja s kojima će ti problemi biti riješeni prije ili kasnije (primjer takvog programa je *PowerFLOW* tvrtke *Exa*). Ako takvi problemi i ne budu riješeni postoji dovoljno drugih metoda koje mogu ispraviti njene nedostatke, a postoje i hibridne metode koje kombiniraju mogućnosti više različitih postupaka uz pokrivanje međusobnih nedostataka. Radi ispravnog odabira potrebno je određeno poznavanje metoda i dobra definicija problema koji želimo riješiti što uz današnje izvore informacija nije teško. Što se praktičnog dijela rada tiče, programske implementacije *LBM*, ispočetka se činilo kao nemoguć posao zbog naizgled kompliciranе fizikalne podloge, ali nakon proučavanja materijala i postojećih radova i nekoliko pokušaja implementacije, nekih više, nekih manje uspješnih, dobiva se znanje potrebno za pisanje konačnog, funkcionalnog produkta uz zaključak da je osnovna implementacija algoritma poprilično jednostavna te da daje lijepe rezultate, i vizualno i brzinski.

7. Literatura

[1] Bauza C. G.,Boroni G., Vénere M., Clausse A.: Real-time Interactive Animations of Liquid Surfaces With Lattice-boltzmann Engines, Australian Journal of Basic and Applied Sciences, 4-2010., 3730-3740

[2] Monitzer A.: Fluid Simulation on the GPU with Complex Obstacles Using the Lattice Boltzmann Method, Diplomski, Institut für Computergraphik und Algorithmen und VRVis Zentrum für Virtual Reality und Visualisierung Forschungs, 2008.

[3] Zhao Y., Kaufman A., Mueller K., Thuerey N., Rüde U, Iglberger K.: Tutorial, Interactive Lattice-Based Flow Simulation and, Visualization, VisWeek08, VIS, INFOVIS, VAST, Kent State University, 2008, 1-206

[4] Chirila B. D.: Introduction to Lattice Boltzmann Methods, lecture, 2010.

[5] Tan J., Yang X.: Physically-based Fluid Animation: A Survey, Science in Chine Series F: Information Sciences, vol. 52., 2009.

[6] Braley C., Sandu A.: Fluid Simulation For Computer Graphics: A Tutorial in Grid Based and Particle Based Methods

[7] Wagner J. A.: A Practical Introduction to the Lattice Boltzmann Method, North Dakota State University, Fargo, 2008.

[8] Li W., Wei X., Kaufman A.: Implementing Lattice Boltzmann Computation on Graphics Hardware, Center for Visual Computing (CVC) and Department of Computer Science, Stony Brook

[9] Chen S., Doolen G.: Lattice Boltzmann Method For Fluid Flows, Fluid Mech, 30-1998. 329-364

[10] Kwak Y., Kuo J., Nakano A.: Hybrid Lattice-Boltzmann/Level-set Method for Liquid Simulation and Visualization, International Journal of Computational Science, 6677-2009.

8. Sažetak

Naslov: Animacija modela vodenih površina

U ovom radu predstavljen je značaj fizikalnih metoda simulacije fluida i razlozi koji su doveli do njihove sve šire primjene u svakodnevnom životu. Dan je kratak pregled trenutno aktualnih metoda iz računalne dinamike fluida, grane računarske znanosti koja proučava to područje. Navedene su njihove primjene, prednosti i nedostaci. Konačan cilj rada je bio razviti program koji će koristiti jednu od spomenutih fizikalnih metoda simulacije za ostvarivanje animacije modela vodene površine. Odabrana je metoda Boltzmannove rešetke, a u pismenom dijelu rada je ta metoda detaljnije opisana s razlozima zašto je baš ona odabrana. Osim toga, opisan je način programske implementacije te metode i problemi koji su se javili prilikom razvoja te moguća buduća poboljšanja i proširenja funkcionalnosti.

Ključne riječi: voda, fluid, fizikalna animacija, računalna dinamika fluida, *LBM*, metoda Boltzmannove rešetke

9. Abstract

Title: Animation of water surface model

In this thesis I present the importance of physical methods of fluid simulation and reasons that led to their broader use in everyday life. Short summary of currently actual methods in computational fluid dynamics, branch of computer science that studies this area, was given. Their uses, advantages and disadvantages were listed. Final goal of the thesis was development of a program that would use one of the mentioned physical methods of simulation for creation of water surface model animation. Lattice Boltzmann method was chosen. In the written part of the thesis the method was described in more detail together with reasons why it was chosen. Apart from that, way of creating programmatic implementation of the method was described, as well as problems that I encountered during development. Also, possible future improvements and extensions of functionality were listed.

Keywords: water, fluid, physical animation, computational fluid dynamics, *LBM*, Lattice Boltzmann Method

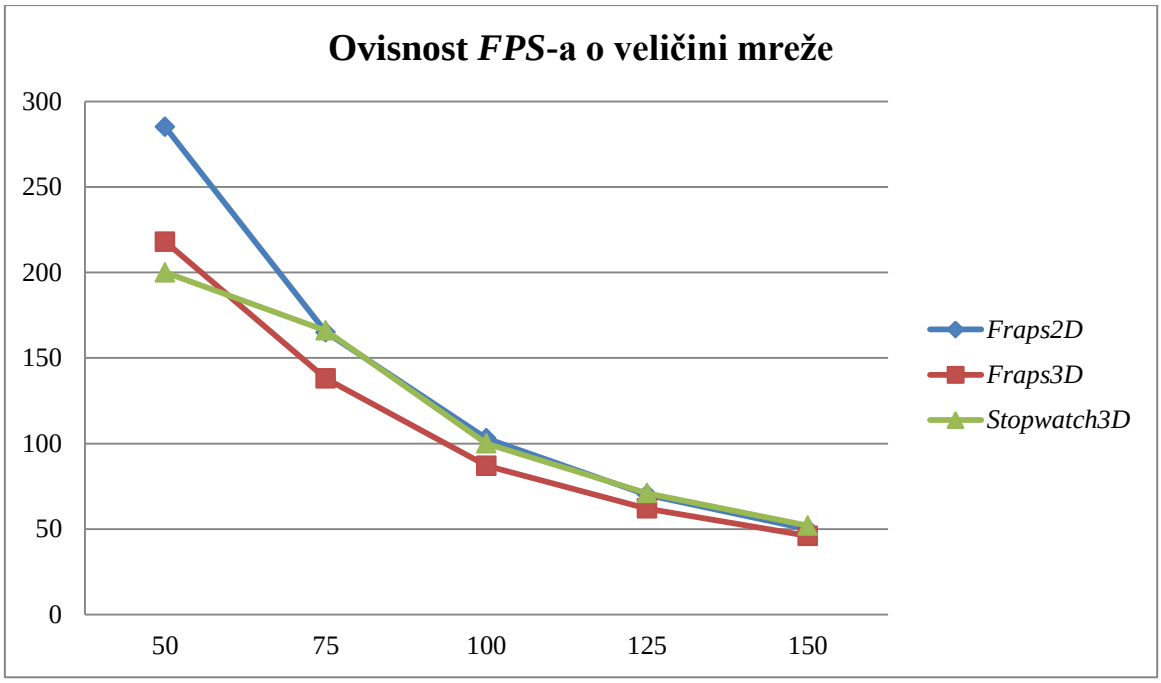
10. Privitak

10.1. Testovi i rezultati

Program je testiran na dvojezgrenom procesoru T5800, *Core2Duo*, takt 2 GHz, s 2MB dijeljene *L2* priručne memorije i 2GB radne memorije na operativnom sustavu Windows7 – x86. Za testiranje je korištena klasa *Stopwatch* iz .NET Frameworka te komercijalno dostupan program *Fraps* u besplatnoj verziji za privatnu upotrebu. U početku su performanse mjerene samo *Fraps-om* pa u slučaju samo *2D* prikaza postoje samo ti podaci, a u konačnoj verziji programa je mjereno vrijeme na raznim mjestima u programu s klasom *Stopwatch* i naposljetku s programom *Fraps*. Mjereno je zasebno vrijeme potrebno za računanje u fazama strujanja i sudara te zasebno vrijeme potrebno za punjenje *VertexArray-a* i iscrtavanje (*Fraps* je bio isključen tijekom testiranja). Zatim su ta vremena uspoređena s rezultatima *Fraps-a* koji je mjerio konačnu brzinu osvježavanja prikaza. Sve rešetke koje su mjerene su bile kvadratne, a dimenzije su bile: 50, 75, 100, 125, 150. Također je potrebno uzeti u obzir da je slika koja se iscrtava u slučaju *2D* prikaza 4 puta veće površine od dimenzije rešetke, a u slučaju *3D* 16 puta veće površine od dimenzije rešetke (svaka od dimenzija rešetke je množena s 2 u slučaju *2D*, a s 4 u slučaju *3D*). Na temelju dobivenih rezultata vidimo da se rezultati ne podudaraju u potpunosti, ali je okvirno moguće vidjeti razliku u brzini simulacije ovisno o veličini rešetke. Prilikom mjerenja pomoću metode *Stopwatch* primijetio sam da prvi put kada je potrebno iscrtati sliku cijeli proces traje otprilike 20-ak puta dulje nego kasnija vremena, pretpostavljam zato jer se prvi put puni priručna memorija s nekim poljima koja koristim, a kasnije se uglavnom čita i piše iz njih.

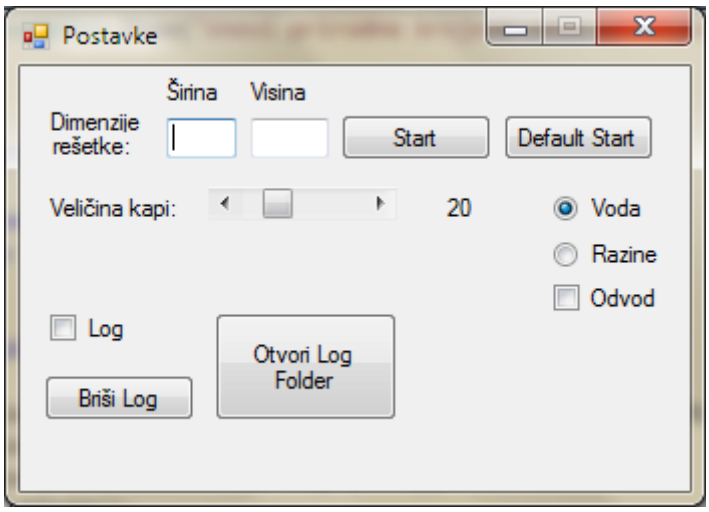
Tablica 2. Mjerenja *2D* i *3D*

Dimenzije	*Fraps2D* / FPS	*Fraps3D* / FPS	*StreamCollide* / ms	Iscrtavanje / ms	Izračun / FPS
50	285	218	2	3	200
75	165	138	3	3	166
100	103	87	5	5	100
125	70	62	7	7	71
150	50	46	10	9	52



10.2. Upute za korištenje

Kada se progam pokrene pojavi se prozor s postavkama simulacije. Funkcija kontrola bi trebala biti logična, ali u slučaju da nije, slijedi opis. Simulaciju je moguće pokrenuti gumbom *Start* (tada je potrebno prethodno upisati dimenzije rešetke simulacije, gdje nije dopušteno unijeti vrijednosti manje od 50 zbog premale slike ili vrijednosti veće od 500 zbog opasnosti da se računalo ne zaglavi) ili ga je moguće pokrenuti gumbom *Default Start* što pokreće simulaciju s rešetkom veličine *150x150*. Takva rešetka je bila korištena u prethodno navedenim testovima, a te vrijednosti su izabrane jer su dobar kompromis između izgleda i performansi prilikom izvođenja na mom računalu.



Slika 15. Prozor s postavkama

Nakon pokretanja simulacije interakcija je moguća pomoću miša u *2D* tlocrtnom pogledu (klikanje ili klikanje i povlačenje) te je moguće rotirati *3D* scenu tipkama WSADQE (dok je aktivan *2D* prozor). Prilikom trajanja rada simulacije može se utjecati na veličinu kapljice, tj. sile pomoću klizačau prozoru postavki. Postoje *radio* gumbi kojima se bira način vizualizacije i *toggle* gumbi gdje se s jednim aktivira funkcija spremanja podataka u datoteku (također postoje gumbi kojima se jednostavno briše *log-datoteka* ili za otvaranje direktorija u kojem se *log-datoteka* nalazi ako je želimo pročitati), a drugim se aktivira funkcija odvoda ako se scena zasiti bojom (ili ako samo želimo vidjeti efekt odvoda).

Nije predviđeno mijenjanje veličine prozora, ali zbog tehničkih nedostatka *Glut-a* nisam uspio spriječiti promjenu veličine na *3D* prozoru. Tako da, iako se može mijenjati veličina prozora perspektivna projekcija neće biti pravilno podešena.

Program se zatvara gašenjem bilo kojeg od prozora na znak X, ili kombinacijom *Alt+F4* na prozoru s postavkama.