SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1589

ANIMACIJA TOKA FLUIDA U OKRUŽENJU S PREPREKAMA

Zoran Dropić

Zagreb, lipanj 2006.

Zahvaljujem se

mentorici doc.dr.sc. Željki Mihajlović

obitelji i prijateljima koji su bili uz mene tijekom cijelog studija

i posebno roditeljima koji su često moje obaveze

shvaćali ozbiljnije nego ja sam

Sadržaj:

UVOD:

Stari su Grci tvari oko sebe tumačili različitim udjelima samo četiri temeljna građevna elementa. Od ta četiri, vatre, vode, zraka i zemlje, tri su fluid. Iako pogrešno, ovo je tumačenje bilo empirijsko, posljedica doživljavanja svijeta kroz neposredno iskustvo. Naša se znanost promijenila, ali naša su svakodnevna iskustva jednaka kao ona starih Grka. Svijet oko nas je isti i u velikoj mjeri protječe oko nas.

Na iskustvenoj razini, današnja klasifikacija predviđa još uvijek jednak broj osnovnih građevnih elemenata. Četiri stanja u kojima se može naći bilo koja tvar su kruto, tekuće, plinovito i plazma. Od ova četiri, opet su tri fluidi.

Bilo koji sustav sastavljen od velikog broja manjih elemenata može se na razini tih elemenata ponašati na dva moguća načina. Ili su međusobne veze čvrste i nepromjenjive, što rezultira krutošću tvari, ili se mijenjaju, što je osnovno svojstvo fluida.

Gibanje galaksija kroz svemir, molekula vode u kapljici kiše, kretanje ljudi na ulici ili ponašanje atmosferskih poremećaja, kao i nefizikalne veličine kao strujanje novca u svijetu globalne ekonomije, sve se te pojave mogu modelirati relativno jednostavnim modelom fluida.

S obzirom na tako sveobuhvatan utjecaj na svijet oko nas i nas same, proučavanje fluida je jedan od najznačajnijih zadataka mnogih područja ljudskog djelovanja.

Cilj ovoga rada je dati uvid u temeljne zakone i pojmove ovoga područja, ukazati na osnovne probleme i ciljeve te dati primjer modela sustava i njegove simulacije.

1 Model fluida

Kako je već rečeno, pod pojmom "fluid" može se naći vrlo široka paleta sustava sa sasvim različitim svojstvima, poljima djelovanja i ciljevima modeliranja. Temeljni principi su jednaki, a različiti parametri razlikuju jedan slučaj od drugoga. S obzirom na ovo, naizgled potpuno različiti sustavi mogu bitnim svojstvima biti međusobno bliži nego neki naizgled srodni.

Bez obzira na specifičnosti konkretnih slučajeva, neka temeljna svojstva su zajednička svima.

1.0.1 Čestični i statistički modeli

Svijet oko nas je svijet čestica. Na kemijski smislenoj razini, sve oko nas je građeno od atoma i molekula. Najočitiji primjer fluida, voda, sastoji se od molekula koje se dalje mogu rastaviti na atome vodika i kisika.

Ove čestice su u najvećem broju slučajeva nekoliko redova veličine udaljene od specifičnih utjecaja kvantne mehanike. To znači da se njihovo ponašanje može zadovoljavajuće objasniti korištenjem ničeg drugoga pored Newtonovih zakona mehanike. Dvije molekule vode će se u slučaju sudara elastično odbiti ne previše različito od bilijarskih kugli. Model vode koji uključuje samo dva međudjelovanja između dvije molekule, odbijanje kada su preblizu i privlačenje kada su udaljenije, ali ne previše daleko, dovoljan je da realno prikaže veliki raspon tekućina. Mijenjanjem samo nekoliko parametara, kao što su intenzitet međudjelovanja, brzina rasta i pada u ovisnosti o udaljenosti, najveća udaljenost na kojoj se utjecaj jedne molekule osjeća i slično, moguće je simulirati viskoznost, plastičnost, površinsku napetost, adheziju i koheziju i slična svojstva tekućina. Čestični model u točnosti prikaza i korištenju stvarne slike svijeta prednjači nad statističkim modelom.

Međutim, čestični model u svojoj srži nosi promatranje jedne jedine molekule vode. Bila ona usred ustajale bare ili na pola puta do podnožja slapa, svo računanje počiva na istim postulatima i jednako je zahtjevno u oba slučaja. Još očitiji je problem što jedna obična čaša vode sadrži broj molekula reda veličine 10^{24}. Sam model koji ima toliki broj elemenata je fizikalno korektan i matematički točan, ali ustvari neiskoristiv.

Kao rješenja navedenoga problema kristaliziraju se dvije mogućnosti. Problem je, naime, da nije moguće (a uglavnom niti nema smisla) gledati svaku česticu posebno kada ih je prisutan toliki broj. U tako postavljenom problemu vidljive su dvije promjenjive veličine. Prvo, pitanje je moramo li zaista gledati baš svaku česticu posebno. Drugo, da li nam je potrebno baš toliko pojedinačnih čestica. Iako se ta dva pitanja čine slična, iz njih izniče suštinska razlika dva temeljna pogleda na modeliranje fluida.

Jedan je pogled čestični. Ideja je da je veliki broj čestica bespotreban i da se fluid može korektno modelirati daleko manjim brojem elemenata. Ovdje se od stvarnog slučaja nasljeđuje navika odvojenog promatranja svake pojedinačne čestice podložne Newton-ovim zakonima (i moguće dodatnim utjecajima već u ovisnosti o ciljevima simulacije). Ponašanje svakog pojedinog elementa fizikalno je točno i manje više jednako ponašanju čestica stvarnog fluida koji se simulira u granicama točnosti koja se traži.

Drugi pogled na svijet nije ništa manje utemeljen na stvarnosti. Naime, pojedine čestice su slobodne da se kreću u bilo kojem smjeru u bilo kojem trenutku, da imaju bilo kakvu brzinu i bilo kakva ostala svojstva koja su im pridijeljena. Međutim, u čaši vode s 10^{24} molekula individualnost svake pojedine teško može doći na vidjelo. Na takvoj razini bitni su trendovi u svojstvima, a ne značajke svakog pojedinog elementa. Slikovitije, jedna srdela u jatu može u nekom trenutku odlučiti zastati ili krenuti prema začelju, ali na sonaru na ribarskom brodu ovo neće utjecati na ukupno kretanje jata. Ako se mogu razraditi statističke formulacije ovakvih trendova, statistički model fluida postaje moguć.

Oba ova modela imaju prednosti i mane. Čestični je model istinit i sveobuhvatan model fluida. Međutim, on je apsolutno točan tek pri odnosu broja elemenata modela i broja elemenata stvarnog sustava od jedan prema jedan. Prosječna se rijeka ne može simulirati s par desetaka čestica. Mirna, nizinska rijeka se, međutim, može vrlo realno simulirati koristeći tek nekoliko čvorova statističkog modela. S druge strane, potpuno turbulentni, nepredvidivi sustavi se ni na koji način ne mogu opisati prosječnim veličinama, jer prosjeci ovdje nemaju smisla. Tu se svaka čestica ponaša zasebno. Nadalje, statistička veličina, tlak, bolje regulira međudjelovanje dvaju različitih fluida nego razmjena impulsa sile, isto kao što statistička temperatura bolje simulira razmjenu energije nego količina gibanja. Diskretne čestice, međutim, čine osiguranje očuvanja mase i detekciju prisutnosti fluida u nekom volumenu trivijalnim zadatkom.

Prednosti i mane određenog modela imaju različite težinske faktore u ovisnosti o potrebama konkretne simulacije. Koji će model biti korišten ovisi o svakom pojedinom slučaju.

1.0.2 Temeljne veličine

Bez obzira koji je model odabran, simulacija se uvijek izvršava s ciljem dobijanja bolje predodžbe o ponašanju određenog fluida u zadanim uvjetima. Kad se govori o ponašanju, misli se nužno na ponašanje temeljnih veličina koje opisuju sustav. Kako rezultati simulacije trebaju biti opis stvarnosti, očito je da će te temeljne veličine također biti naslijeđene iz stvarnog svijeta, a ako se i radi o nekim apstraktnim konceptima, oni su samo međukorak između sustava modela i realnog sustava.

Na razini čestica postoje samo dvije bitne veličine. To su brzina i položaj u odnosu na druge čestice. U statističkom modelu situacija je ponešto složenija, ali zahvaljujući našem iskustvu u interakciji s fluidima na statističkoj razini, apstraktniji pojmovi ovoga modela su nam bliži od stvarne čestične podloge. Brzina se kao temeljna odrednica prenosi i na makroskopsku skalu, ali kao kontinuirana veličina, a ne čestično diskretizirana. Relativni položaji na ovoj razini donose gustoću, koja je također kontinuirana veličina. Povezanost gustoće i položaja čestica, iako nije preslikavanje svojstva kao u slučaju brzine, intuitivno je jasno. Međutim, dodatna dva bitna svojstva makroskopskog svijeta nisu u tolikoj mjeri jednostavna. Jedno od njih je tlak, koji je u potpunosti apstraktna i statistička veličina dobivena prosječnom silom svih čestica primijenjenom na neku zamišljenu ravninu interesa. Druga veličina je temperatura, koja ovisi o prosjeku razmijenjenih impulsa sile čestica. Iako sasvim apstraktne i u potpunosti statističke, bez ikakve vrijednosti u čestičnom modelu, ove dvije veličine su ključne za statistički model. One su savršen primjer pojednostavljenja u odnosu na čestični model utoliko što na jednostavan način grupiraju doprinose velikog broja čestica u veličine iskoristive za točno određene namjene. One su prave temeljne veličine makroskopskog svijeta.

Ovdje je bitno napomenuti da četiri navedena parametra statističkog modela (brzina, gustoća, tlak i temperatura) nisu jedini koji mogu biti od interesa. Specifični model se

može proširiti dodatnim svojstvima (na primjer električni naboj, radioaktivnost, ali i razna nefizikalna svojstva ako se radi o takvoj simulaciji). Također, u nekim slučajevima sva četiri parametra nisu ni potrebna. Za simulaciju nestlačivih fluida stabilne toplinske energije gustoća i temperatura su konstantne veličine i nemaju utjecaja. U takvom su slučaju brzina i tlak jedine bitne veličine. Čak je i apsolutni iznos tlaka u određenim uvjetima nebitan, a važna je samo razlika tlakova. O tome će još biti riječi.

1.0.3 Usporedba matematičkog i fizikalnog modela

Model promatranog sustava mora zadovoljiti najmanje dva zahtjeva. Jedan je fizikalna točnost, tj. sličnost stvarnom sustavu koji se simulira. Drugi je iskoristivost za specifične potrebe simulacije.

Modeliranje je kao tehnika testiranja razrađenih (i manje razrađenih) teorija prisutno vjerojatno već desecima tisuća godina, od prvih iskri ljudske inovativnosti (a možda i ne tek ljudske). Ideja na kojoj modeliranje počiva relativno je jednostavna. Želja je dobiti rezultate zanemarivo različite od onih koji se očekuju od stvarnog sustava i to uz manje troškove. Troškovi ovdje ne znače samo novac, nego i druga uložena sredstva, kao što je vrijeme, radna snaga, sirovine itd. Naime, projekt novog prekooceanskog broda najbolje bi bilo testirati u punoj veličini i u realnim uvjetima, ali graditi novi brod na svakom koraku projekta je, naravno, besmisleno. Nadalje, često je potrebno testirati samo određene ključne dijelove sustava.

Temeljno je pitanje kakav model izgraditi. S rastom snage računala matematički se modeli guraju bliže prednjim redovima. Donedavno računala jednostavno nisu bila u stanju računati utjecaje svih potrebnih veličina na promatrani sustav. Danas se to mijenja. Naravno, svaki specifični slučaj je uvijek bolje predstavljen realnim sustavom, nego hrpom brojki u računalu. Najveći kapacitet za točno i pouzdano računanje izlijevanja vode iz čaše ima sama čaša vode. Ono što računalne modele izdiže iznad ovoga je sposobnost interakcije sa sustavom u potpuno kontroliranim uvjetima. Ne postoji prebrza reakcija niti presitan prostor kada su vrijeme i prostor samo neki od parametara proračuna. Ne postoji neizvjesnost iščekivanja točno određenog događaja kad se s par

brojki taj događaj može garantirati. Nema troškova izrade novog modela kad se stari može prilagoditi u bilo kojem smislu.

Međutim, da bi se prednosti matematičkog modela zaista ostvarile, matematička formulacija mora biti točna i sveobuhvatna, nešto što fizikalni model automatski zadovoljava. Dok je dakle jedan nužno točan, ali možda ne omogućuje uvid u željene pojave, drugi cilja izravno na područje interesa, ali se njegova točnost mora posebno osigurati.

1.1 Matematička formulacija

Kod razmatranja ponašanja fluida bitno je u svakom trenutku voditi računa o fizikalnoj podlozi. Kakva god matematika bila primijenjena, osnovni zakoni fizike ne mogu biti narušeni. Jednadžbe koje opisuju sustav moraju zadovoljavati zakone očuvanja mase, energije, momenta, kutnog momenta i druge zakone bitne za promatrani sustav (očuvanje količine naboja i slično). S obzirom da su ovi zakoni fundamentalni, iz njih se mogu izvesti opisi bilo kakvog sustava.

1.1.1 Navier Stokesove jednadžbe

U devetnaestom stoljeću, neovisno i s petnaestak godina razlike, engleski fizičar George Gabriel Stokes i francuski Claude Louis Navier dali su matematičku formulaciju gibanja fluida. Te su jednadžbe danas poznate kao Navier Stokesove jednadžbe. U svom standardnom obliku glase:

$$\frac{\delta \vec{u}}{\delta t} = -(\vec{u}\nabla)\vec{u} - \frac{1}{\rho}\nabla p + \frac{\nu}{\rho}\nabla^2\vec{u} + \vec{g} \qquad (1.1.1)$$

gdje je **u** vektor brzine, p tlak, **g** akceleracija sile teže, ρ gustoća fluida, υ statički koeficijent viskoznosti, a ∇ standardni matematički operator za koji vrijedi

$$\nabla = \frac{\delta}{\delta x}\vec{i} + \frac{\delta}{\delta y}\vec{j} \qquad (1.1.2)$$

Djelovanje operatora je pojašnjeno u Dodatku A.

Navier Stokesove jednadžbe za slučaj bez utjecaja vanjske sile (u gornjoj formulaciji gravitacija) mogu se određenim supstitucijama ispisati u bezdimenzionalnom obliku koji omogućava višu razinu slobode u izradi modela. Matematička pozadina je dana u Dodatku B, a nakon nje jednadžbe imaju oblik:

$$\frac{\delta \vec{u}}{\delta t} = -(\vec{u}\nabla)\vec{u} - \nabla p + \frac{1}{\mathrm{Re}}\nabla^2\vec{u} \qquad (1.1.3)$$

gdje je Re bezdimenzionalni Reynoldsov broj kako je objašnjeno u Dodatku B. Ovo je vektorska jednadžba, pa iz toga dolazi upotreba množine. S lijeve strane se nalazi

vremenska derivacija brzine, odnosno akceleracija. S desne strane su izrazi koji reguliraju doprinose konvekcije, advekcije, difuzije i gravitacije promjeni brzine.

Za potrebe kasnije upotrebe u poglavlju o diskretizaciji, zapis Navier Stokesovih jednadžbi kako je dan u (1.1.3) treba raspisati do diferencijalnih jednadžbi po komponentama brzine. Prvi korak je zamjena operatora ∇ njegovim diferencijalnim oblikom prema (1.1.2) i prema pravilima iz Dodatka A:

$$\frac{\delta \vec{u}}{\delta t} = -\left(u_x \frac{\delta}{\delta x} + u_y \frac{\delta}{\delta y} \right) \vec{u} - \left(\vec{i} \frac{\delta p}{\delta x} + \vec{j} \frac{\delta p}{\delta y} \right) + \frac{1}{\mathrm{Re}} \left(\frac{\delta^2}{\delta x^2} + \frac{\delta^2}{\delta y^2} \right) \vec{u}$$

Sada slijedi raspisivanje vektora **u** u komponentama brzine s jediničnim vektorima

$$\frac{\delta \vec{u}}{\delta t} = -\left(u_x \frac{\delta u_x}{\delta x} \vec{i} + u_x \frac{\delta u_y}{\delta x} \vec{j} + u_y \frac{\delta u_x}{\delta y} \vec{i} + u_y \frac{\delta u_y}{\delta y} \vec{j} \right) - \left(\vec{i} \frac{\delta p}{\delta x} + \vec{j} \frac{\delta p}{\delta y} \right) +$$
$$+ \frac{1}{\mathrm{Re}} \left(\frac{\delta^2 u_x}{\delta x^2} \vec{i} + \frac{\delta^2 u_y}{\delta x^2} \vec{j} + \frac{\delta^2 u_x}{\delta y^2} \vec{i} + \frac{\delta^2 u_y}{\delta y^2} \vec{j} \right)$$

Grupiranjem izraza uz jedinične vektore i rastavljanjem vremenske derivacije s lijeve strane dobijaju se dvije momentne jednadžbe, svaka s utjecajima koji se odnose samo na odgovarajuću komponentu brzine

$$\frac{\delta u_x}{\delta t} = -\left(u_x \frac{\delta u_x}{\delta x} + u_y \frac{\delta u_x}{\delta y} \right) - \frac{\delta p}{\delta x} + \frac{1}{\mathrm{Re}} \left(\frac{\delta^2 u_x}{\delta x^2} + \frac{\delta^2 u_x}{\delta y^2} \right) \qquad (1.1.4)$$

$$\frac{\delta u_y}{\delta t} = -\left(u_x \frac{\delta u_y}{\delta x} + u_y \frac{\delta u_y}{\delta y} \right) - \frac{\delta p}{\delta y} + \frac{1}{\mathrm{Re}} \left(\frac{\delta^2 u_y}{\delta x^2} + \frac{\delta^2 u_y}{\delta y^2} \right) \qquad (1.1.5)$$

U ovom obliku jednadžbe su spremne za diskretizaciju i upotrebu u algoritmu i simulaciji.

1.1.2 Jednadžba kontinuiteta

Dok je momentna jednadžba jednadžba u čijoj pozadini stoji zakon o očuvanja momenta, u jednadžbi kontinuiteta se manifestira zakon o očuvanju mase. Uzme li se bilo koji proizvoljni volumen fluida, razlika količine fluida u njemu između bilo koja dva trenutka mora biti jednaka razlici fluida koji su ušli i izašli iz promatranog volumena. Matematičkim rječnikom:

$$\frac{\delta\rho}{\delta t} + \nabla\rho u = 0$$

Za nestlačive fluide međutim vrijedi da je gustoća konstantna. To znači da se fluid ne može akumulirati u volumenu i jednadžba kontinuiteta glasi

$$\nabla\vec{u} = 0 \qquad (1.2.1)$$

Raspiše li se ovaj izraz dobija se

$$\left(\frac{\delta}{\delta x}\vec{i} + \frac{\delta}{\delta y}\vec{j}\right)\vec{u} = \frac{\delta u_x}{\delta x} + \frac{\delta u_y}{\delta y} = 0 \qquad (1.2.2)$$

tj. ukupna količina fluida koji ulazi u bilo koji odabrani volumen mora u svakom trenutku biti jednaka količini fluida koja izlazi iz toga volumena. U sustavima simulacije toka fluida jednadžba kontinuiteta mora biti zadovoljena u svakom trenutku.

Navier Stokesove jednadžbe su diferencijalne jednadžbe drugoga stupnja. Osim u nekim specijalnim slučajevima, analitički su nerješive. Naime, zbog ovisnosti svakog djelića volumena fluida o svim svojim susjedima i obrnuto, cijeli je sustav povezan. Da bi se izračunale promjene brzine i tlaka za svaki dio sustava potrebno je znati već promijenjene veličine svih susjednih elemenata, što je naravno nemoguće jer je za njihovo računanje potrebno znati upravo ono što se pokušava izračunati. Nadalje, čak i kada su poznate vrijednosti susjednih veličina, računanje polja brzina ovisi o poznavanju polja tlaka, ali i obratno.

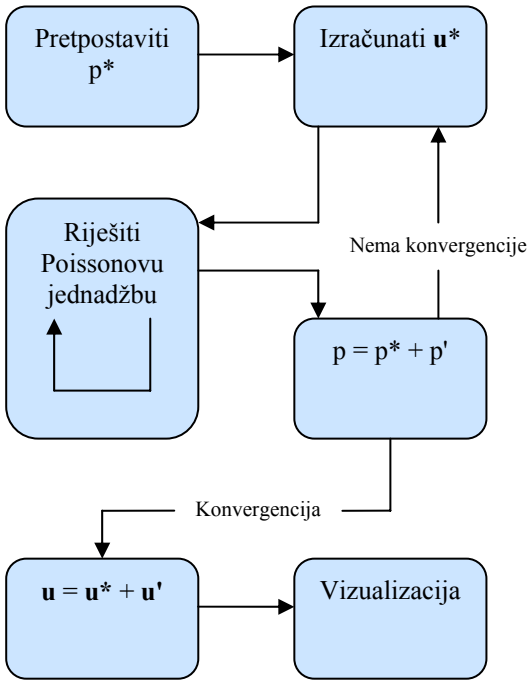
Osim u specijalnim slučajevima, dakle, Navier Stokesove jednadžbe su nerješive analitičkim metodama. Jedini način računanja je numerički, iterativni postupak. S obzirom da je takav pristup računalno skup, uvijek se traži mogućnost izravnije metode. Zbog važnosti ovoga problema, institut za matematiku Clay ga je uvrstio u popis sedam tzv. milenijskih problema za čije je rješenje (ili dokaz nepostojanja rješenja) ponuđena nagrada od milijun američkih dolara.

Dok se problem ne riješi, međutim, numerička metoda je jedina raspoloživa, a s obzirom da za ukupno rješenje sustava treba doći do ispravnih vrijednosti nekoliko međusobno povezanih veličina, sam postupak je dovoljno kompliciran da je razvijen velik broj algoritama. Neki od njih su specijalizirani za određene slučajeve, neki su ograničeni na

dvije dimenzije, ali postoje i općenitija rješenja, od kojih će ovdje biti predstavljen možda najpoznatiji. Algoritam nosi naziv SIMPLE.

1.2 Algoritam SIMPLE

Ime SIMPLE je akronim engleskoga naziva *Semi Implicit Method for Pressure Linked Equations* (djelomično implicitna metoda za tlakom povezane jednadžbe). Objašnjenje



Slika 1. SIMPLE algoritam

imena je dano u Dodatku C. Algoritam datira još iz 1972-e godine, a predložili su ga Patankar i Spalding. Prema [10], može se opisati u šest glavnih koraka:

1. Pretpostaviti vrijednosti tlaka.
2. Koristeći tekuće vrijednosti tlaka izračunati nove iznose brzina.
3. Koristeći nove brzine iterativnom metodom riješiti Poissonovu jednadžbu za dobijanje korekcija tlaka.
4. Primijeniti korekcije na iznose tlaka.
5. Ako su korekcije manje od uvjeta konvergencije, to su konačne vrijednosti i nastavlja se s korakom 6. Ako ne, nastavlja se s korakom 2.
6. Iz sada točnih vrijednosti tlaka izračunati korigirane brzine.

Algoritam je prikazan i na Slici 1.

1.2.1 Prvi korak – postavljanje početnih uvjeta

U prvom koraku potrebno je pretpostaviti vrijednosti tlaka na cijeloj mreži. Ako se radi o *steady state* problemu, dobro je početne vrijednosti postaviti na iznose karakteristične za promatrani problem. Ako se simulira sustav koji polazi iz mirovanja, postavljanje tlaka na nulu je obično dovoljno. Iako će ovo vrlo vjerojatno dovesti do pojave negativnih tlakova u sustavu, što je fizikalno nemoguće, za simulaciju nestlačivih fluida ova je varka nebitna. Naime, kod nestlačivih fluida o tlaku ovisi samo promjena brzina, a i ona samo o razlikama tlaka, a ne o apsolutnom iznosu. Iako bi direktna metoda rješavanja sustava jednadžbi zbog nedefinirane vrijednosti dovela do singularne matrice i nemogućnosti davanja rješenja, iterativna metoda će se prilagoditi i raditi će sa bilo kojom srednjom vrijednošću tlaka. Zbog potrebe računanja razlika, početna vrijednost nula zato ima više smisla jer se zbog prirode *floating point* zapisa brojeva na taj način zadržava najviša preciznost.

1.2.2 Drugi korak – rješavanje momentne jednadžbe

U drugom koraku algoritma se iz zadanih vrijednosti brzina i pretpostavljenih vrijednosti tlaka izračunavaju nove brzine koristeći momentnu jednadžbu. Zbog korištenja pogrešnog polja tlaka ove brzine neće biti točne i biti će im potreban ispravak. Tijekom iterativnog postupka će ispravci biti sve manji, a rješenje sve bliže.

1.2.3 Treći korak – rješavanje Poissonove jednadžbe

Treći korak predstavlja algoritam u malom. S obzirom da polje korekcija tlaka ovisi o samome sebi, rješenje je moguće dobiti jedino iterativno. Uvjet konvergencije je ovdje jednostavan. Promatraju se promjene iznosa u uzastopnim iteracijama i kada najveća promjena u čitavom sustavu postane manja od zahtijevane razine točnosti krajnje vrijednosti su dostignute. Također, u ovom se koraku bilježi najveća izračunata korekcija u apsolutnom iznosu, koja se kasnije koristi za provjeru konvergencije vanjskog iterativnog postupka.

1.2.4 Četvrti korak – primjena korekcija tlaka

Četvrti korak je po kompleksnosti suprotnost prethodnom. Starim iznosima tlaka pridodaju se izračunate korekcije da bi se dobile nove vrijednosti.

1.2.5 Peti korak – provjera konvergencije

U petom koraku se provjerava konvergencija cjelokupnog postupka. S obzirom da tlak ovisi o brzinama, a brzine o tlaku, postojanje korekcije tlaka ukazuje na neuravnoteženost ove dvije veličine. Ako korekcija nema, znači da već postojeći tlak odgovara postojećim brzinama. To dalje znači da bi se korištenjem postojećeg tlaka opet dobile iste brzine. Na taj način je za konvergenciju glavnog iterativnog postupka dovoljno da najveća korekcija tlaka u cijelom sustavu, koja je izračunata u četvrtom koraku, iznosi manje od zadanog uvjeta konvergencije. Ako to nije tako, algoritam se nastavlja drugim korakom. Korisno je broj iteracija ograničiti da bi se u slučaju divergencije spriječio ulazak u beskonačnu petlju. Treba naravno paziti da najveći dopušteni broj iteracija bude dovoljno velik za dostizanje konvergencije. Za prve korake ovo može biti dosta velika brojka, dok je kasnije dovoljno sve manje i manje. S druge strane, što simulacija dalje odmiče, ako se dosegne stabilno stanje, divergencija je sve manje vjerojatna. Ograničenje dakle treba oprezno odabrati, ali u svakom slučaju je bolje izabrati preveliki nego premali broj.

1.2.6 Dodatne napomene

Da bi momentna jednadžba bila točno riješena, dovoljno je poznavati točne vrijednosti tlakova u sustavu. To, međutim, nije moguće bez poznavanja brzina. Cijeli algoritam SIMPLE je zato samo postupak dobijanja točnog polja tlaka. U posljednjoj iteraciji postupka u drugom će se koraku koristiti stare vrijednosti brzina, jednako kao i u prvom koraku, ali uz njih nove vrijednosti tlaka. Na taj način će nove brzine biti točno dobivene.

Nakon što jedan cijeli korak algoritma završi, novodobiveni se tlak koristi kao novi pretpostavljeni i nastavlja se opet s drugim korakom algoritma, prilagođavajući tlak još jednom da bi se na kraju mogle izračunati sljedeće brzine. S vremenom, kako sustav dolazi bliže konačnom, stabilnom stanju, broj iteracija će se smanjivati, da bi na kraju sve završilo u jednostrukim prolazima kroz algoritam. S obzirom da se brzine i tlakovi više ne mijenjaju, korekcije će u svakom koraku biti nula, pa će i uvjeti konvergencije uvijek biti

zadovoljeni već u prvom prolazu. Zbog ovoga razloga najkritičniji dio simulacije je obično upravo prvi korak. Naime, tada su iznosi najudaljeniji od realnih vrijednosti i divergencija rezultata je najvjerojatnija. Zato dobro pogođene početne vrijednosti mogu značiti razliku između dobijanja dobrih rezultata ili nedobivanja rezultata uopće.

Da bi se problem barem djelomično ublažio, uvode se sitne preinake u jednadžbe. S obzirom da na početku iznosi temeljnih veličina mogu imati velike raspone na uskom području, izračunate korekcije mogu biti preuveličane i na taj način mogu doprinijeti ionako pogrešnoj slici. Zato se uvode težinski faktori za korekcije, tj. koeficijenti koji skaliraju korekcije. Namjernim smanjenjem promjene algoritam se usporava u slučaju da su dobivene brojke realne, ali mu se zato daje mogućnost da ispravi greške u slučaju da je rješenje krenulo krivim smjerom.

Koeficijente treba pažljivo odrediti jer premale vrijednosti mogu pretjerano usporiti simulaciju. Isto tako, prevelike vrijednosti vjerojatno neće biti dovoljne da spriječe eventualnu divergenciju rezultata.

Ove preinake ne pomažu samo u početnim trenucima simulacije. Promatrani sustav može završiti u stabilnom stanju, ali u velikom broju slučajeva cilj je promatrati sustave koji nikada ne dostignu vremensku stalnost. U takvim slučajevima su ovakve metode također korisne.

1.3 Diskretizacija

Navier Stokesove jednadžbe daju matematički okvir ponašanja fluida. Kako su te jednadžbe analitički nerješive, razvijaju se algoritmi dizajnirani za što brži, lakši ili razumljiviji put ka rješenju, a koji koriste numeričke metode. Jedan od takvih algoritama je i SIMPLE.

No, Navier Stokesove jednadžbe u svom diferencijalnom obliku nisu pogodne za upotrebu u računalnim programima. Diferencijabilne funkcije su nužno kontinuirane, dok računalna oprema podržava računanje samo diskretnih veličina. Da bi se kontinuirane veličine mogle koristiti, one moraju proći proces diskretizacije.

S obzirom da kontinuirane funkcije imaju beskonačno finu uzorkovanost, a diskretne nužno samo konačnu, očito je da pri diskretizaciji dolazi do gubitka podataka. Također, pri diskretizaciji vremena gube se ili netočno računaju sve promjene čije je trajanje usporedivo s trajanjem jednog vremenskog koraka.

Ako se radi s linearnim funkcijama, gubitaka nema, jer se na jednostavan način linearne promjene mogu rekonstruirati iz diskretnih vrijednosti. Ako se poznaje oblik funkcije, u mnogim se slučajevima također originalna funkcija može rekonstruirati. Međutim, ako se radi o nekoj općoj raspodjeli, kao što je raspodjela brzina i tlaka u nekom fluidu, gubitak informacija kroz prijelaz u diskretnu domenu je ireverzibilan. To upućuje na potrebu da se odabiru glavnih parametara diskretizacije pristupi oprezno. Uzorkovanje podataka na krivim mjestima može lako dovesti do nerealnih, a time i beskorisnih rezultata.

1.3.1 Postupak diskretizacije

Prvo što je potrebno učiniti je prepoznati bitne veličine čije vrijednosti treba uzorkovati. Kada se radi o nestlačivim fluidima, to su brzina i tlak. Obje ove veličine su kontinuirane i mogu imati relativno velike promjene unutar malih prostornih pomaka. Kod toka oko oštre prepreke, na primjer, promjene brzine su iznimno velike na samoj oštrici u odnosu na područje neposredno uz prepreku. Slična je stvar i s tlakom. U takvim situacijama je pravilna diskretizacija prostora od presudne važnosti.

Kad se o prostoru radi, postoje dvije moguće podjele. Jedna je stvaranje pravilne mreže, a druga stvaranje takve mreže koja će biti specijalno prilagođena specifičnom zadatku.

Pravilne mreže imaju prednost u jednostavnosti implementacije. U takvom slučaju granice građevnih elemenata su strogo zadane, razmaci među uzorcima temeljnih veličina su konstantni i bilo kakve interpolacije su najjednostavnije moguće. S druge strane, pravilne mreže u potpunosti zanemaruju bilo kakve nehomogenosti u simuliranom sustavu i u potpunosti se jednako odnose prema nezahtjevnim, kao i prema najzahtjevnijim dijelovima sustava. Računalna snaga se troši na nepotrebne kalkulacije dok na područjima gdje se događaju brze ili lokalizirane izmjene finoća uzorkovanosti ne zadovoljava zahtjeve za realnošću simulacije. Nepravilne mreže nadoknađuju ove nedostatke, ali i stvaraju probleme kod određivanja granica među elementima, utjecajima susjednih elemenata, interpolaciji podataka i slično. Kao i uvijek, odabir ovisi samo o potrebama specifične situacije.

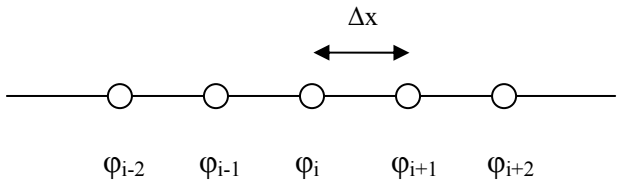
Iako se kad se govori o prostornoj mreži govori o, naravno, prostoru, to ne znači da podjela sustava mora biti vremenski nepromjenjiva. Kod dinamičnih sustava daleko se najbolji rezultati dobiju primjenom dinamičnih mreža. Ovdje se mreža u svakom koraku prilagođava trenutnom stanju sustava. Za primjer se može uzeti tok fluida oko kružne prepreke (ali sličan efekt nastaje pri toku oko raznih prepreka). Ovdje se u pozadini prepreke naizmjenično na jednoj i drugoj strani stvaraju vrtlozi koji se zatim odvajaju od prepreke i putuju niz struju s vremenom gubeći na snazi. Pravilna mreža može prikazati ovaj klasični problem, ali nije u stanju zabilježiti sve sitne detalje toka unutar vrtloga. Nepravilna statična mreža neće dati ništa bolje rezultate (naprotiv, uz isti broj elemenata može dati i gore rezultate od pravilne mreže). Međutim, dinamična mreža, koja u svakom koraku najfiniju podjelu ima upravo na vrtlozima, dati će najtočnije podatke o ponašanju sustava. Nedostatak ove metode je lako shvatljiv. Svi problemi s detekcijom granica među elementima, interpolacijom podataka i ostalim zadaćama koje se moraju vršiti na razini elemenata mreže, a što je problem i statičnih nepravilnih mreža, ovdje poprimaju novu dimenziju s obzirom da se ne rješavaju samo jednom nakon diskretizacije prostora, nego u svakom vremenskom koraku simulacije.

Vremenska diskretizacija ima slične probleme. I ovdje je, iako daleko manje zastupljena, nepravilna podjela moguća i u određenim slučajevima bi dala bolje rezultate. Trenuci kada dolazi do stvaranja ili odvajanja vrtloga su trenuci kada se na vremenskoj skali promjene događaju relativno brzo. Ipak, vremenska je diskretizacija obično pravilna i to uglavnom zbog potrebe vremenski ujednačene vizualizacije simulacije.

1.3.2 Derivacije diskretne funkcije

Nakon što su iznesene opće napomene vezane za prostorno vremensku diskretizaciju potrebno je još pokazati principe derivacije diskretnih funkcija. Pod uvjetom da je neka veličina uzorkovana, sve informacije o njoj sada su sadržane isključivo u diskretnim točkama sustava. Deriviranje se više ne obavlja na kontinuiranoj funkciji. Diferencijalne jednadžbe postaju jednadžbe diferencija. Prije formulacije Navier Stokesovih jednadžbi i Poissonove jednadžbe u njihovim diskretnim oblicima, potrebno je definirati derivacije na diskretnim veličinama.

Na slici 2 prikazan je jednodimenzionalni slučaj zamišljenog sustava s karakterističnom varijablom φ. Vrijednosti koje se pojavljuju u proračunima i stoga moraju biti poznate su iznos same funkcije u promatranoj točki, zatim njena derivacija i druga derivacija na istom mjestu. Iznos funkcije je očito na raspolaganju bez potrebe za ikakvim računanjem. Iznos derivacije je donekle problematično pitanje. Naime, derivacija je definirana kao iznos promjene derivirane veličine u ovisnosti o promjeni veličine po kojoj se derivira. Kod diskretnih funkcija iznos promjene veličine po kojoj se derivira je također diskretiziran. U slučaju sa slike 2 on iznosi Δx. Promjena derivirane veličine, funkcije φ, unutar toga intervala je ($\varphi_{i+1} - \varphi_i$). Derivacija je dakle



Slika 2. Jednodimenzionalni prostor u kojem je u diskretnim točkama definirana funkcija φ

$$\frac{\delta\varphi}{\delta x} = \frac{\varphi_{i+1} - \varphi_i}{\Delta x} \qquad (1.3.2.1)$$

Problem je, kako se vidi, u činjenici da na ovaj način izračunata derivacija neće biti iznos koji vrijedi u točki i, nego onaj koji bi vrijedio u točki i+1/2. Taj se indeks, međutim, nalazi između elemenata i u diskretizaciji prostora nema smisla. Druga je mogućnost da se promjena φ računa kao ($\varphi_i - \varphi_{i-1}$), ali ovo je identičan problem. Jedini način da se iznos derivacije dobije točno na indeksu i je taj da se promjena računa na širem području, čime se dobije formula

$$\frac{\delta\varphi}{\delta x} = \frac{\varphi_{i+1} - \varphi_{i-1}}{2\Delta x} \qquad (1.3.2.2)$$

Ovako izračunata derivacija zaista daje iznos točno na indeksu *i*, ali je problem u korištenju vrijednosti funkcije razmaknutih za dvostruku širinu elementa mreže. Na taj način se preciznost računa smanjuje i efektivno se radi s podacima točnima jednako kao da mreža ima dvostruko manje elemenata (u jednodimenzionalnom slučaju). Iz ovoga se razloga ipak odabire formula (9.1). Iako loše postavljena, ona u sebi nosi veću preciznost i zato je bolji izbor.

Druga derivacija sada ne predstavlja problem. S obzirom da je definirana kao derivacija prve derivacije, a prve derivacije su definirane u točkama i+(2k+1)/2, k$\in$**N**, rezultat se jednostavno dobije točno na željenom indeksu

$$\frac{\delta^2\varphi}{\delta x^2} = \frac{\dfrac{\delta\varphi_{i+1}}{\delta x} - \dfrac{\delta\varphi_{i-1}}{\delta x}}{\Delta x} = \frac{\dfrac{(\varphi_{i+1}-\varphi_i)-(\varphi_i-\varphi_{i-1})}{\Delta x}}{\Delta x} = \frac{\varphi_{i+1}-2\varphi_i+\varphi_{i-1}}{\Delta x^2} \qquad (1.3.2.3)$$

Općenito, svaka derivacija parnog stupnja u odabranom načinu računanja će biti definirana točno na elementu mreže, pa se naziva centralno izračunatom vrijednošću, a svaka derivacija neparnog stupnja će biti definirana pola širine elementa mreže ispred elementa za koji se računa, pa se naziva unaprijedno (engl. *forward*) izračunatom vrijednošću. No, za potrebe simulacije fluida dovoljne su upravo definirane derivacije prvog i drugog stupnja.

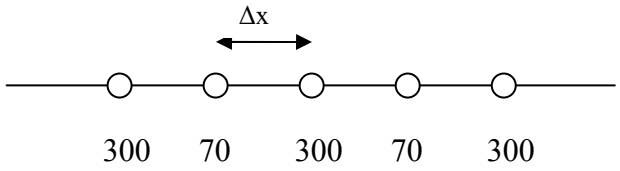
1.4 Diskretizacija jednadžbi

1.4.1 Efekt šahovske ploče

Poznavajući diferencijalni oblik Navier Stokesovih jednadžbi i metode diskretizacije iz prethodnoga poglavlja, moguće je formulirati diskretni oblik jednadžbi. Međutim, ovdje dolazi do određenih problema koji direktno utječu na rezultate simulacije. Naime, jedan od članova desne strane Navier Stokesovih jednadžbi je utjecaj razlike tlaka na promjenu brzine fluida. Da bi se izračunao taj utjecaj potrebno je izračunati kolika je njegova promjena unutar elementa diskretiziranog prostora. Da bi ova promjena mogla biti izračunata, potrebno je znati iznose tlaka na granicama elementa prostora. Pošto su ključne veličine definirane u centrima elemenata, koristi se jednostavna linearna interpolacija. Zatim se ove vrijednosti koriste za računanje derivacije. Dobije se dakle

$$p_{i+0.5} - p_{i-0.5} = \frac{p_{i+1} + p_i}{2} - \frac{p_i - p_{i-1}}{2} = \frac{p_{i+1} - p_{i-1}}{2} \qquad (1.4.1.1)$$

Kako je vidljivo, razlika tlaka na granicama bilo kojeg elementa mreže biti će izračunata kao razlika vrijednosti tlaka zapisanih u susjednim elementima. U računima se nikada neće pojaviti neka druga kombinacija, tako da susjedne vrijednosti neće nimalo ovisiti jedna o drugoj. Zahvaljujući ovome, potpuno nerealno polje tlaka u kojem se izmjenjuju dvije po volji različite vrijednosti zadovoljiti će momentnu jednadžbu. Primjer takvoga jednodimenzionalnoga problema je prikazan na slici 3.



Slika 3. Iako potpuno nerealne, vrijednosti polja zadovoljavaju momentnu jednadžbu

Slično problemu s momentnom jednadžbom, problemi nastaju i s jednadžbom kontinuiteta. Za jednodimenzionalni sustav, ta će jednadžba jednostavno glasiti

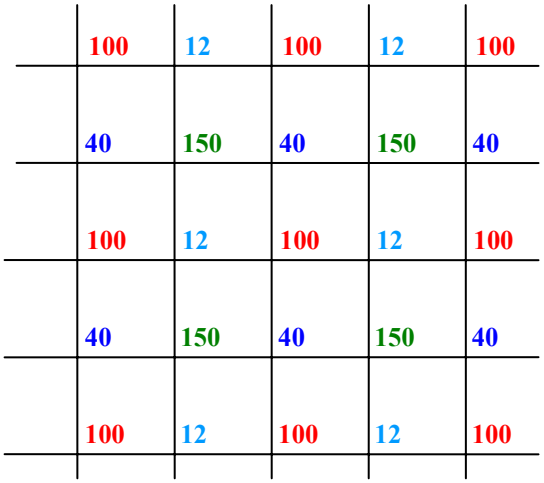
$$\frac{\delta u}{\delta x} = 0 \qquad (1.4.1.2)$$

Dakle, količina fluida koja uđe u jedan element prostora mora u svakome trenutku biti jednaka količini fluida koja izađe. Pošto su granice jednako velike, količina ovisi samo o

brzini, što znači da ulazna i izlazna brzina moraju biti jednake. Brzine na granicama se dobiju linearnom interpolacijom kao i u slučaju tlaka za momentnu jednadžbu, pa je izraz za jednadžbu kontinuiteta

$$u_{i+0.5} - u_{i-0.5} = \frac{u_{i+1} + u_i}{2} - \frac{u_i - u_{i-1}}{2} = \frac{u_{i+1} - u_{i-1}}{2} \qquad (1.4.1.3)$$

Očito je da je problem identičan. Slična se stvar događa u dvije dimenzije, kako je prikazano na slici 4. Problem se naziva problemom šahovskog polja (eng. *checkerboard*). Razlika u dvodimenzionalnom slučaju je da je ponavljajući element veličine četiri



Slika 4. Problem šahovskog polja u dvije dimenzije

elementa prostora. U trodimenzionalnom slučaju radilo bi se o osam elemenata u kocki dimenzija 2x2x2.

Očito je da se ne može dopustiti da susjedne brzine u mreži budu toliko različite, a da te razlike nemaju nikakav utjecaj na ponašanje fluida.

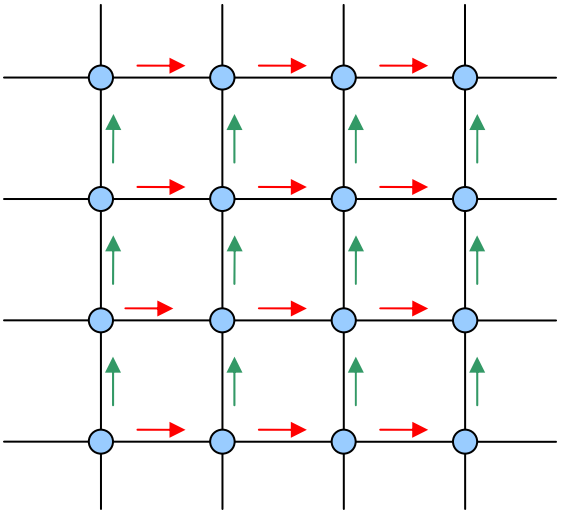
Da bi se ovaj problem riješio, predloženo je nekoliko rješenja. Neka od njih su postavljanje dodatnih ograničenja na vrijednosti brzina i tlakova, dodavanje dodatnih uvjeta za granice sustava, posebno odabrani početni uvjeti i slično. Najkorištenija i za opći slučaj najprimjerenija metoda posmaknute mreže je pojašnjena u sljedećem poglavlju.

Zanimljivo je primijetiti da probleme u rješavanju gotovo isključivo stvaraju prve derivacije, dok se druge redovno ponašaju stabilno i ne unose nered u sustav.

1.4.2 Rješenje problema - Posmaknuta mreža

Posmaknuta mreža (eng. *staggered grid*) je metoda stvorena na ideji da sve ključne veličine ne moraju biti definirane u istim točkama. Narav diskretnog sustava zahtijeva diskretizaciju svih veličina, ali ne govori ništa o potrebi jednake diskretizacije prostora za sve varijable. Dakle, za svaku varijablu može se odabrati posebna mreža elemenata. Naravno, u ovako nešto se ne bi ulazilo da takav pristup sa sobom ne nosi određene prednosti.

U slučaju brzina, prednost koju donosi ova metoda upravo je rješavanje efekta šahovskog polja. Kako je vidljivo na slici 5, brzine su definirane na granicama elemenata, tj. na polovici udaljenosti od jednog čvora do drugoga i to tako da su brzine u x smjeru



Slika 5. Lokacije definicija tlaka (plavo), brzine u x smjeru (crveno) i y smjeru (zeleno)

pomaknute duž x osi, a one u y smjeru duž y osi. Tlakovi i dalje ostaju definirani u središtima elemenata. Ovako definiran sustav zapravo je logičniji izbor i bliži stvarnosti od sustava s jedinstvenom mrežom. Naime, tlak, kao svojstvo elementa, definiran je upravo za jedan element. Brzine, s druge strane, definiraju razmjenu fluida i u tom smislu logički znače opis granice.

Prednosti ovakvoga načina diskretizacije su dvojake. S jedne strane riješen je problem opisan u prethodnom poglavlju. Tamo je cilj bio, kad se radi o brzinama i jednadžbi kontinuiteta, izračunati razliku brzine na ulazu i izlazu iz jednog elementa mreže. Kako ove vrijednosti nisu bile definirane, dobijale su se interpolacijom glavnih čvorova, što je rezultiralo eliminacijom vrijednosti promatranog čvora iz proračuna. Sada međutim

interpolacija nije potrebna jer su poznate upravo potrebne veličine. Zato sada jednadžba kontinuiteta za jednodimenzionalan problem glasi jednostavno

$$\frac{\delta u}{\delta x} = u_{i+0.5} - u_{i-0.5} = 0 \qquad (1.4.2.1)$$

Druga prednost koja izniče iz upotrebe posmaknute mreže dolazi iz pozicija tlaka. Iako ta mreža nije promijenjena, odnos njenog i položaja mreža brzina se promijenio. Razlika tlaka među međusobno susjednim elementima sada postaje prirodna mjera utjecaja polja tlaka na polja brzina, s obzirom da su središta elemenata jedne mreže na granicama elemenata druge mreže i obratno. Zahvaljujući ovom novom svojstvu i drugi problem iz prethodnog poglavlja se automatski rješava. Naime, taj problem, vezan uz momentnu jednadžbu, zahtijevao je računanje promjene tlaka na granicama elementa zbog potreba računanja utjecaja na brzinu. Zbog potrebe interpolacije dolazilo je do eliminiranja vrijednosti iz promatranog elementa. S obzirom da su sada vrijednosti na granicama upravo one koje su poznate, razlika se računa na susjednim čvorovima, a ne na svakom drugom.

Za oba problema posmaknuta mreža daje mogućnost računanja razlika tlaka i brzina na susjednim elementima. Kako je svaki element sada povezan sa svojim susjedima, situacije prikazane u prethodnom poglavlju su nemoguće. Naime, velike razlike tlakova u susjednim čvorovima sada će nužno dovesti do velike promjene brzine na njihovoj granici, a velike razlike brzina nužno će promijeniti tlak, izglađujući postepeno cijeli sustav. Ovakva metoda na kraju daje realne rezultate.

1.4.3 Diskretizacija Navier Stokesovih jednadžbi

Sada je napokon definirano sve potrebno za formulaciju diskretnog oblika Navier Stokesovih jednadžbi. Diferencijalni zapis sadrži različite oblike i stupnjeve derivacija tlaka i brzina. S lijeva na desno članovi su derivacija brzine po vremenu, brzine po prostoru, tlaka po prostoru i na kraju druga derivacija brzine po prostoru. Prvo je najjednostavnije:

$$\frac{\delta u}{\delta t} = \frac{u^{n+1} - u^{n}}{\Delta t} \qquad (1.4.3.1)$$

gdje oznaka *n* označava vremenski korak. Kako je vrijeme diskretizirano u trenucima T+kΔt, k∈**N**, derivacija se svodi na jednostavnu promjenu vrijednosti u uzastopnim vremenskim koracima.

Derivacija tlaka po prostoru, s obzirom da je tlak i dalje definiran u središtima elemenata mreže, identična je već prikazanoj općoj metodi diskretizacije prve derivacije

$$\frac{\delta p}{\delta x} = \frac{p_{i+1,j} - p_{i,j}}{\Delta x} \tag{1.4.3.2}$$

$$\frac{\delta p}{\delta y} = \frac{p_{i,j+1} - p_{i,j}}{\Delta y} \tag{1.4.3.3}$$

Problemi o kojima je bilo riječi, a odnosili su se na računanje prve derivacije na granici elementa umjesto u središtu, sada više ne predstavljaju nedostatak nego dobrodošlu podudarnost pošto su brzine na čiji se iznos računa utjecaj promjene tlaka definirane upravo na granicama.

Prva derivacija same brzine, međutim, rezultate daje točno u središtima elemenata, što je sada nepovoljno. Međutim, prema već pokazanome, jedina alternativa je umjetno smanjenje rezolucije mreže, što je još gore rješenje. Prva derivacija brzine je zato:

$$\frac{\delta u}{\delta x} = \frac{u_{i+0.5,j} - u_{i-0.5,j}}{\Delta x} \tag{1.4.3.4}$$

$$\frac{\delta u}{\delta y} = \frac{u_{i,j+0.5} - u_{i,j-0.5}}{\Delta y} \tag{1.4.3.5}$$

Druga derivacija brzine će sada opet davati rezultat relevantan za traženu točku:

$$\frac{\delta^2 u}{\delta x^2} = \frac{\frac{\delta u_{i+0.5,j}}{\delta x} - \frac{\delta u_{i-0.5,j}}{\delta x}}{\Delta x} = \frac{(u_{i+1.5,j} - u_{i+0.5,j}) - (u_{i+0.5,j} - u_{i-0.5,j})}{\Delta x} = \frac{u_{i+1.5,j} - 2u_{i+0.5,j} + u_{i-0.5,j}}{\Delta x^2}$$

(1.4.3.6)

$$\frac{\delta^2 u}{\delta y^2} = \frac{\frac{\delta u_{i,j+0.5}}{\delta y} - \frac{\delta u_{i,j-0.5}}{\delta y}}{\Delta y} = \frac{(u_{i,j+1.5} - u_{i,j+0.5}) - (u_{i,j+0.5} - u_{i,j-0.5})}{\Delta y} = \frac{u_{i,j+1.5} - 2u_{i,j+0.5} + u_{i,j-0.5}}{\Delta y^2}$$

(1.4.3.7)

U ovom trenutku su definirane samo derivacije brzina u smjerovima na koje se odnose. Definirana je dakle samo prva i druga derivacija *x* komponente brzine po *x*-u i derivacije *y* komponente po *y*-u. Da bi se Navier Stokesove jednadžbe mogle ispisati u diskretnom obliku, potrebno je još dati objašnjenje derivacija brzina u suprotnim smjerovima.

Kako su brzine u *x* smjeru definirane u točkama (i+(2k+1)/2, j+k), k∈**N**, njihova prva derivacija u *y* smjeru neće iskoristiti prednosti posmaknute mreže iz jednostavnog razloga što *x* mreža nije posmaknuta u *y* smjeru. Zbog toga će ova vrijednost biti važeća za točku točno između dva dijagonalno postavljena glavna čvora. Slično razmatranje vrijedi i za *y* komponentu brzine. Prve derivacije će tako biti:

$$\frac{\delta u_x}{\delta y} = \frac{u_{i+0.5,j+1} - u_{i+0.5,j}}{\Delta y} \tag{1.4.3.8}$$

$$\frac{\delta u_y}{\delta x} = \frac{u_{i+1,j+0.5} - u_{i,j+0.5}}{\Delta x} \tag{1.4.3.9}$$

Derivacije drugoga stupnja će davati vrijednosti koje su važeće točno za čvorove mreža brzina koje se deriviraju.

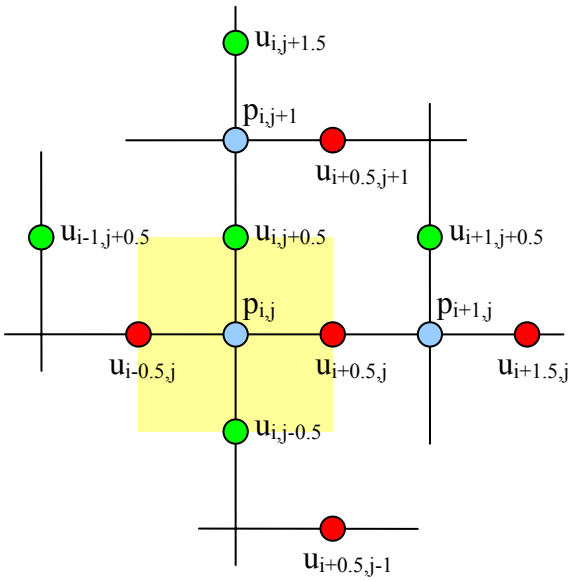
$$\frac{\delta^2 u_x}{\delta y^2} = \frac{u_{i+0.5,j+1} - 2u_{i+0.5,j} + u_{i+0.5,j-1}}{\Delta y^2} \tag{1.4.3.10}$$

$$\frac{\delta^2 u_y}{\delta x^2} = \frac{u_{i+1,j+0.5} - 2u_{i,j+0.5} + u_{i-1,j+0.5}}{\Delta x^2} \tag{1.4.3.11}$$

U ovom trenutku su u diskretnom obliku definirani svi članovi Navier Stokesovih jednadžbi. Prije krajnje formulacije zanimljivo je pogledati koje vrijednosti tlakova i brzina su potrebne za rješavanje momentne jednadžbe za jedan čvor mreže, dakle za računanje iznosa brzina u *x* i *y* smjeru.

Na slici 6 plavim, crvenim i zelenim krugovima su prikazani redom čvorovi tlaka, brzine u *x* smjeru i brzine u *y* smjeru. Lako se može primijetiti posmaknutost mreža. Žutom podlogom je označen element koji se obrađuje. Vidljivo je da je za rješavanje momentne jednadžbe za samo jednu diskretnu točku potrebno koristiti čak trinaest različitih vrijednosti. Iako se ovo može pročitati iz jednadžbi, na ovakvoj slici je lakše uočiti koliko su udaljene neke vrijednosti nužne za točan izračun. Pri implementaciji algoritma treba posebno paziti na elemente blizu rubova simuliranoga prostora ili blizu prepreka. Poseban oprez je potreban pri radu s tankim preprekama, širine jednoga elementa mreže, jer tada

druga derivacija brzina može iskoristiti vrijednosti iz udaljenih čvorova koje mogu biti prilagođene uvjetima s druge strane prepreke.



Slika 6. Vrijednosti tlaka i brzina potrebne za momentnu jednadžbu pri obradi osjenčanog elementa

Na kraju, Navier Stokesove jednadžbe, ili momentna jednadžba, u diskretnom i nevektorskom obliku glasi:

$$\frac{u_{i+0.5,j}^{n+1}-u_{i+0.5,j}^{n}}{\Delta t}=-\left(u_{i+0.5,j}\frac{u_{i+0.5,j}-u_{i-0.5,j}}{\Delta x}+u_{i,j+0.5}\frac{u_{i+0.5,j+1}-u_{i+0.5,j}}{\Delta y}\right)-\frac{p_{i+1,j}-p_{i,j}}{\Delta x}+$$
$$+\frac{1}{\mathrm{Re}}\left(\frac{u_{i+1.5,j}-2u_{i+0.5,j}+u_{i-0.5,j}}{\Delta x^2}+\frac{u_{i+0.5,j+1}-2u_{i+0.5,j}+u_{i+0.5,j-1}}{\Delta y^2}\right)$$

(1.4.3.12)

$$\frac{u_{i,j+0.5}^{n+1}-u_{i,j+0.5}^{n}}{\Delta t}=-\left(u_{i+0.5,j}\frac{u_{i+1,j+0.5}-u_{i,j+0.5}}{\Delta x}+u_{i,j+0.5}\frac{u_{i,j+0.5}-u_{i,j-0.5}}{\Delta y}\right)-\frac{p_{i,j+1}-p_{i,j}}{\Delta y}+$$
$$+\frac{1}{\mathrm{Re}}\left(\frac{u_{i+1,j+0.5}-2u_{i,j+0.5}+u_{i-1,j+0.5}}{\Delta x^2}+\frac{u_{i,j+1.5}-2u_{i,j+0.5}+u_{i,j-0.5}}{\Delta y^2}\right)$$

(1.4.3.13)

1.4.4 Diskretizacija Poissonove jednadžbe

Nakon pregleda diskretizacije svih derivacija iz prethodnoga poglavlja, za Poissonovu jednadžbu su već dati gotovo svi potrebni izrazi. Jedino što je ostalo nedefinirano, pošto se ne koristi u momentnoj jednadžbi, je derivacija tlaka drugoga stupnja. Međutim, s obzirom da je tlak definiran u točkama (i+k, j+k), k$\in$**N** taj je izraz već prikazan u poglavlju o općoj diskretizaciji derivacija. Izraz dakle glasi:

$$\frac{\delta^2 p}{\delta x^2} = \frac{p_{i+1,j} - 2p_{i,j} + p_{i-1,j}}{\Delta x^2} \qquad (1.4.4.1)$$

$$\frac{\delta^2 p}{\delta y^2} = \frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{\Delta y^2} \qquad (1.4.4.2)$$

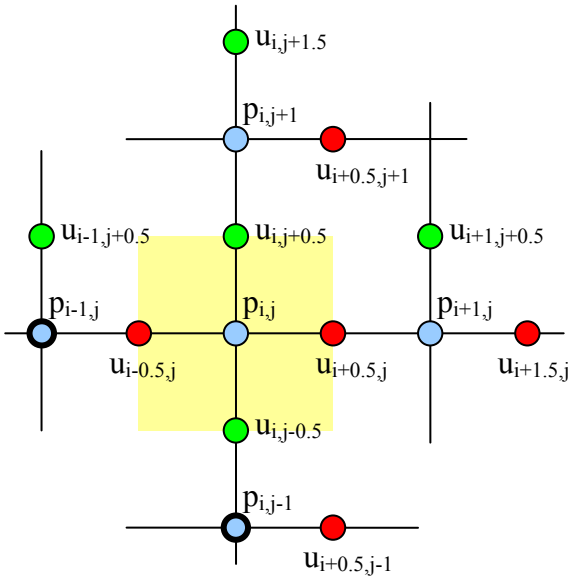
Diskretni oblik Poissonove jednadžbe je skalarni izraz s obzirom da je tlak skalarna veličina, pa tako ova jedna jednadžba glasi:

$$\frac{p'_{i+1,j} - 2p'_{i,j} + p'_{i-1,j}}{\Delta x^2} + \frac{p'_{i,j+1} - 2p'_{i,j} + p'_{i,j-1}}{\Delta y^2} = \frac{1}{\Delta t}\left(\frac{u_{i+0.5,j} - u_{i-0.5,j}}{\Delta x} + \frac{u_{i,j+0.5} - u_{i,j-0.5}}{\Delta y}\right) \qquad (1.4.4.3)$$

Izlučivanjem $p'_{i,j}$ iz gornje jednadžbe dobije se eksplicitni izraz za korekciju tlaka u točki (i, j). S obzirom da se susjedne korekcije dobijaju korištenjem jednake jednadžbe, proces računanja svih korekcija na mreži mora, kako je već pojašnjeno, biti iterativan.

Poissonova jednadžba u odnosu na momentnu jednadžbu donosi nove dvije vrijednosti potrebne za dovršetak jedne iteracije glavnog algoritma. Na slici 7 prikazana je konačna slika potrebnih točki slično kao na kraju prethodnoga poglavlja. Nove dvije vrijednosti su označene crnim krugom. To su vrijednosti čvorova tlaka koje nisu bile potrebne u momentnoj jednadžbi koja koristi samo prvu derivaciju tlaka, ali su potrebne u Poissonovoj gdje se koristi i druga derivacija. Za razliku od momentne jednadžbe, Poissonova ne koristi derivacije brzina drugoga stupnja, kao ni derivacije brzina po smjerovima kojima same ne pripadaju, pa su tako za rješenje ove jednadžbe dovoljne samo veličine koje omeđuju promatrani element, kao i vrijednosti tlakova u svim susjednim elementima.

Za kompletno rješenje momentne i Poissonove jednadžbe za svaki element potrebno je koristiti ukupno petnaest vrijednosti tlaka i brzina. Iz ovoga je očito kolika je međusobna



Slika 7. Vrijednosti tlaka i brzina potrebne za momentnu i Poissonovu jednadžbu pri obradi osjenčanog elementa

povezanost susjednih elemenata, a time i cjelokupne mreže elemenata. Iterativni se postupak, međutim, dobro nosi s ovim zadatkom.

Dosad učinjeno

Dosadašnjim pregledom je prikazan matematički model ponašanja fluida, primjer algoritma koji koristi numeričku metodu za rješavanje danog modela te način diskretizacije jednadžbi kao nužan korak prema implementaciji na računalu.

Slijedi sama implementacija, sa specifičnim zahtjevima i rješenjima izabranog radnog okruženja i izabranih problema računalne dinamike fluida.

2 Implementacija

2.0.1 Izbor platforme

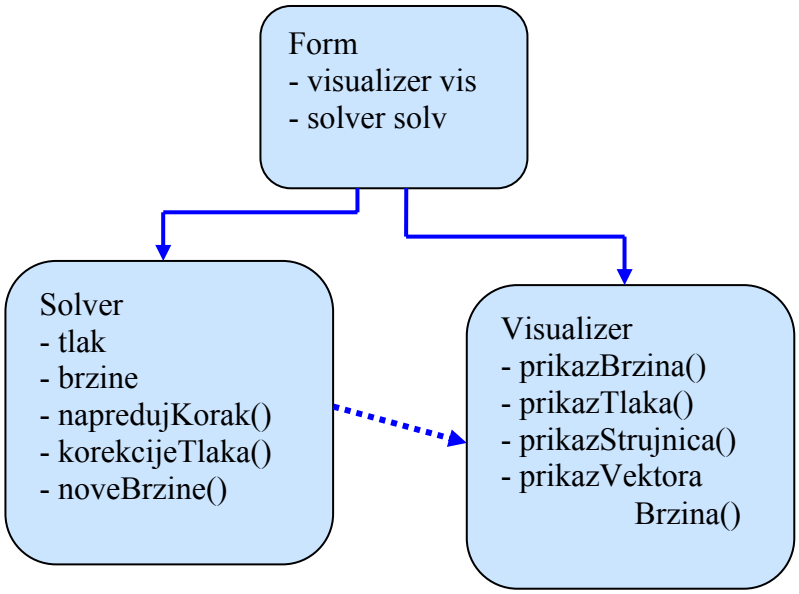
U izradi programskog rješenja za potrebe ovoga rada korišten je programski jezik C#. Pitanje izbora nije lagan zadatak s obzirom da mnogi kandidati imaju specifične prednosti. Iako C# nije najkorišteniji jezik niti stvara najbrži program ili najmanje izvršne datoteke, većina zamjerki koje mu se mogu naći nisu ključna pitanja za potrebe ovoga rada. Jednostavnost sintakse i gotovo univerzalna prepoznatljivost naslijeđena iz programskih jezika C i Java, zatim ogromna množina raspoloživih biblioteka unutar proizvoda Visual Studio .NET podržanih kroz platformu .NET Framework i objektna orijentiranost jezika samo su neke od prednosti. Ugrađeno rukovođenje memorijom i automatsko onemogućavanje najčešćih programerskih grešaka iz drugih jezika, kao što su pogreške s pokazivačima, izlazak iz granica polja i slično su možda najočitiji primjeri. Također, *drag and drop* način izrade Windows aplikacija raspoloživ u paketu Visual Studio .NET bitno skraćuje vrijeme razvoja programa i fokusira pažnju na osnovni problem umjesto na vizualno dotjerivanje.

Objektna orijentiranost jezika se može promatrati kao nedostatak. Naime, s obzirom da u ovom slučaju aplikacija mora simulirati ponašanje fluida koje je modelirano relativno kompleksnim matematičkim modelom koji se mora rješavati iterativno, zahtjevi za računalnom snagom su iznimno veliki. Potpuno objektno orijentirani jezik poput C#-a situaciju još pogoršava tretiranjem svih podataka kao objekata. Iako takav pristup u velikom broju slučajeva pomaže (npr. onemogućuje izlaz iz granica polja jer je polje objekt koji sadrži podatak o vlastitoj duljini i provjerava operacije nad sobom), u slučaju potrebe za brzinom može i odmoći. Dodatni podaci koje svaki objekt nosi pored korisnih podataka mogu biti relativno veliki.

Cilj ovoga rada međutim nije napraviti najbrži ili najmanji program simulacije fluida. Cilj je prikazati principe programiranja, implementaciju ključnih koraka i vizualizacije rezultata simulacije. S obzirom na tako strukturiran zadatak, objektno orijentirano programiranje nudi jednostavnost i preglednost kakvu čisto proceduralni jezici ne mogu postići.

2.0.2 Struktura projekta

Napravljeni projekt se sastoji od tri temeljna dijela: *Windows Form*, klasa *Solver* i klasa *Visualizer*. Njihova je povezanost prikazana na slici 8. *Windows Form* je osnovna klasa za izradu Windows aplikacija. U ovom programu osim standardnih podataka i metoda



Slika 8. Temeljna struktura programa. Objekti tipa *Solver* i *Visualizer* se instanciraju u *Form*-u i njihove se metode tamo pozivaju. Neke metode *Visualizer*-a koriste podatke iz *Solver*-a

vezanih za vizualno rješenje aplikacije sadrži samo nekoliko bitnih naredbi. Kako je ovo dio koda u kojem počinje izvođenje i koji je zadužen za regulaciju tijeka ostatka programa, ovdje se instanciraju objekti tipa *Solver* i *Visualizer*. Određuju se dimenzije prostora simulacije, potrebni parametri, postavlja se mapa prepreka, način toka fluida kroz promatrani prostor i slično. Ukratko, program je zamišljen tako da se iz ovog dijela koda pozivaju metode *Solver*-a koje će riješiti jedan vremenski korak simulacije i metode *Visualizer*-a koje omogućavaju nekoliko načina iscrtavanja rezultata simulacije. Redoslijed, broj poziva i slično su prepušteni izboru, ali implementacija algoritma i sve pojedinosti ostaloga koda su zanemarene s ciljem grupiranja specifičnih zadaća, što je i ideja objektnog programiranja.

2.1 Klasa Solver

Podaci:

- polja
- brzine
- stare brzine
- tlak
- korekcije tlaka
- stare korekcije tlaka
- mapa prepreka

- parametri
- Δt
- Δx
- Δy
- Reynolds-ov broj
- najveća korekcija tlaka
- uvjet konvergencije

Klasa *Solver* je što se simulacije fluida tiče temelj programa. Sve metode koje se bave simulacijom ponašanja fluida, računanjem tlakova i brzina, iterativnim postupkom, provjeravanjem konvergencije i slično smještene su isključivo na ovom mjestu. Klasa *Visualizer* služi samo za vizualizaciju rezultata dobivenih korištenjem podataka i metoda u ovoj klasi.

S obzirom na ovo, sva teorija iznesena u prethodnim poglavljima je priprema za računalnu implementaciju algoritama i struktura podataka korištenih u klasi *Solver*.

2.1.1 Podaci

Vrijednosti brzina su smještene u trodimenzionalno polje `double[,,] vel` tako da prve dvije dimenzije predstavljaju koordinate točke u mreži, a treća dimenzija određuje komponentu brzine. S obzirom da se obrađuje dvodimenzionalni problem, moguće su dvije vrijednosti, za x i y komponente brzine. Pošto se u simulaciji koristi posmaknuta mreža za čvorove brzina, a pravila jezika zahtijevaju cjelobrojne indekse polja, iznosi brzina se zapisuju na način da vrijedi

$$\texttt{vel[i, j, 0]} \equiv brzina(i+0.5, j)$$

$$\texttt{vel[i, j, 1]} \equiv brzina(i, j+0.5)$$

Na ovaj se način čvorovi za sve tri veličine vraćaju ponovno na iste indekse, ali, naravno, to se odnosi samo na zapise u memoriji računala. Sve jednadžbe koje su do sada izvedene se koriste pazeći na ovu razliku.

Momentna jednadžba govori kolika je derivacija brzine po vremenu, odnosno kolika je promjena u odnosu na prethodnu vrijednost, a ne daje apsolutni iznos nove brzine. Zbog toga je za izračun iznosa brzine potrebno poznavati i vrijednosti iz prethodnog vremenskog koraka. Te su stare vrijednosti pohranjene u polju `double[,,] oldVel`. Po dimenzijama i tipu podataka ovo je polje identično prethodnom, a tijekom iteracija algoritma podaci koje sadrže naizmjence se smatraju 'starim' ili 'novim' vrijednostima. Razlike u indeksima između ovoga polja i indeksa posmaknute mreže jednake su kao i za polje novih brzina.

Tlak je skalar i zbog toga je za čuvanje podataka o njemu dovoljno dvodimenzionalno polje `double[,] p`. Za razliku od brzina, stare vrijednosti tlaka se nigdje ne koriste, tako da u ovom slučaju čuvanje podataka iz prethodnog koraka nije potrebno. Također, za razliku od brzina, a kako je već rečeno u poglavlju o posmaknutoj mreži, vrijednosti tlaka zapisane na memorijskoj lokaciji `p[i, j]` odnose se točno na tlak u čvoru *(i, j)*.

U trećem koraku algoritma SIMPLE rješava se Poissonova jednadžba čiji su rezultat korekcije tlakova dobivene koristeći nove vrijednosti brzina. Te su korekcije pohranjene u polje `double[,] pCorrection`. Za njega vrijedi sve već rečeno za polje tlaka, osim potrebe čuvanja starih vrijednosti. Naime, pitanje konvergencije iterativnog postupka rješavanja Poissonove jednadžbe se evaluira mjerenjem najveće promjene korekcije tlaka između dvije uzastopne iteracije. Kada cjelokupni postupak dođe do konvergencije, ta će razlika teoretski biti nula. Praktično, može se odrediti razina točnosti koja je potrebna. U svakom slučaju, da bi se ova razlika mogla izračunati, potrebno je znati stare vrijednosti korekcija. Zapravo je dovoljno imati na raspolaganju vrijednosti onog elementa mreže na kojem se pojavljuje najveće odstupanje, ali kako nema načina da se taj element unaprijed odredi, moraju se sačuvati vrijednosti iz svih elemenata. Stare korekcije su sadržane u polju `double[,] oldPCorrection`, a najveća korekcija tlaka na cijeloj mreži u varijabli `double maxPCorrection`.

Još jedno vrlo bitno polje je `int[,] obstacleMap`. Ovo polje, kako mu i ime govori, sadrži mapu prepreka u promatranom sustavu. Tijekom simulacije elementi označeni kao dio prepreke imaju poseban tretman. O tome će više riječi biti u pojašnjenju metoda koje koriste polje. Tip podataka je cjelobrojni. Naime, iako je dovoljno koristiti logički tip, jer prepreke ima ili nema, za brže izvođenje i lakšu implementaciju nekih pojava ovo polje može označavati više nego jedan tip posebnih elemenata. Granice sustava se mogu

posebno promatrati, mjesta na kojima fluid ulazi ili izlazi iz sustava također, a za ubrzanje rada sve elemente koji su udaljeni od bilo kakvih prepreka može se tako i označiti. U ovoj implementaciji postoje četiri dozvoljene vrijednosti: oznaka prepreke (ujedno i granice sustava), oznaka utoka i istoka fluida iz područja simulacije i na kraju, pomični zid.

Što se ostalih podataka u ovoj klasi tiče, radi se uglavnom o raznim parametrima simulacije. Neke, kao `deltaT`, `deltaX` i `deltaY` ne treba posebno objašnjavati. Može se međutim posvetiti par rečenica pitanju postojanja dvije različite konstante za dimenzije elemenata mreže. Pošto se ovdje radi o implementaciji pravilne mreže, ove će dvije konstante biti jednake i tu postoji određena redundancija. Isto tako, istina je i da su vrijednosti obje konstante točno 1 i da nigdje ne utječu na izračunate vrijednosti. Ipak, zbog bolje čitljivosti koda i bliže veze s matematičkom podlogom, primijenjene formule svugdje odgovaraju onima ispisanima u prethodnim poglavljima. Također, na ovaj način se olakšava svaka buduća izmjena koda.

Uvjet konvergencije `double convergenceCondition` je mjera koja govori koliko blizu savršenoj konvergenciji algoritam treba doći da se konvergencija smatra dostignutom. Pošto se iterativni postupak konvergenciji približava asimptotski, a preciznost *double* tipa podataka je velika, savršeno podudaranje je najčešće bespotrebno perfekcionistički zahtjev. Apsolutno točan rezultat nije presudno bitan za simulaciju, a odstupanje od realnog sustava svakako više proizlazi iz diskretizacije postupka nego iz nedostizanja potpune konvergencije. Ova varijabla zato sadrži granicu na kojoj će iterativni postupak stati i dobiveno rješenje će se smatrati konačnim. Varijabla se koristi i za glavnu petlju algoritma SIMPLE i za rješavanje Poissonove jednadžbe.

Ovime je završen pregled bitnih podataka sadržanih u klasi *Solver*. Može se napomenuti i to da su izvana vidljivi (imaju dozvolu javnog pristupa) samo neki od njih. Korekcije tlaka, npr. su sasvim nebitne izvan klase. Nakon što je konvergencija postignuta i jedan vremenski korak obavljen, sve što je bitno su stanja polja brzina i tlaka. Dostupni su još i parametri koji određuju simulaciju da bi se mogli posebno postavljati.

Klasa sadrži još i niz metoda za rad s ovim podacima. Neke od njih zaslužuju poseban osvrt.

2.1.2 Metode

2.1.2.1 Iteriranje glavne petlje algoritma

Nakon što je prostor simulacije izgrađen postavljanjem vrijednosti polja `obstacleMap` i nakon što su postavljene početne vrijednosti bitnih parametara i polja, jedina metoda koju je potrebno pozvati je `bool advanceTimeStep()`. Ova metoda vraća logičku vrijednost `true` u slučaju da je korak simulacije završio konvergencijom ili `false` ako do konvergencije nije došlo. Sama metoda zapravo samo poziva druge metode u slijedu koji u potpunosti prati algoritam SIMPLE. Izvorni kod je dan u nastavku.

```
	public bool advanceTimeStep ()
	{
		switchVelocities ();

		for (int iteracija = 0; iteracija < maxIteracija;
iteracija++)
		{
			calculateNewVelocities ();
			calculatePCorrection ();
			applyPCorrection ();
			if (maxPCorrection < convergenceCondition)
			{
				applyVelocityCorrections ();
				return true;
			}
		}
		return false;

	}
```

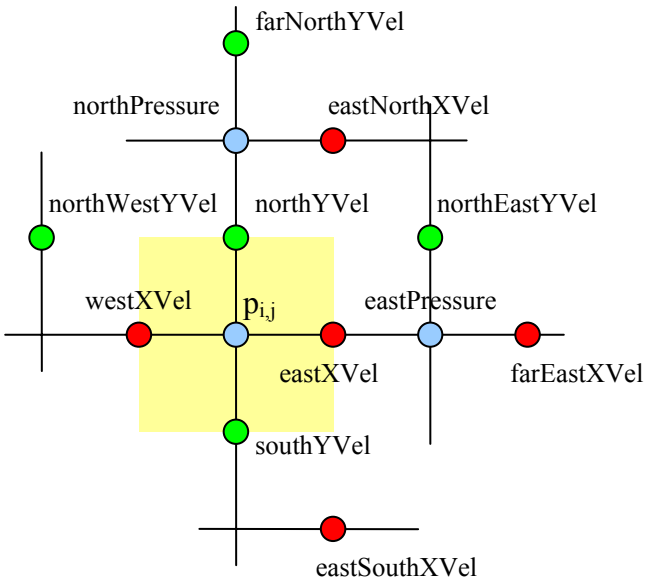
U kodu su vidljivi koraci SIMPLE algoritma. Unutar `for` petlje se prvo računaju novi iznosi brzina (drugi korak algoritma), zatim se iteracijskim postupkom koji je u potpunosti unutar metode `calculatePCorrection()` računaju korekcije tlaka koje se primjenjuju na stare vrijednosti tlaka da bi se dobile nove. Nakon toga se provjerava konvergencija rezultata. Ako je konvergencija postignuta, petlja se prekida i u pozivajući program se vraća vrijednost koja označava uspješno završen korak. Ako do konvergencije nije došlo, cijeli se postupak nastavlja. Petlja će se izvršiti najviše `maxIteracija` puta. Ovaj je broj postavljen dovoljno velik da se može smatrati da ako rezultat nije dostignut u tolikom broju pokušaja da neće ni biti dostignut (rješenje je divergiralo), ali ipak dovoljno nizak da program ne upadne u dugotrajnu besmislenu blokadu. Ako nakon `maxIteracija` konvergencija nije dostignuta, petlja se prekida i u pozivajući program se vraća odgovarajuća vrijednost.

2.1.2.2 Proračun vrijednosti brzina za novi vremenski korak

Računanje novih brzina, kako je vidljivo u metodi `advanceTimeStep()`, vrši metoda `calculateNewVelocities()`. Kako ova metoda ima vrlo jednostavan kod koji samo za svaki element mreže poziva metodu `calculateOneElementNewVelocity()`, bitno je obratiti pažnju na ovu drugu metodu. Njen pseudokod se može prikazati otprilike kako slijedi:

```
calculateOneElementNewVelocity (i, j)
    vrijednosti karakterističnih susjednih veličina postavi
    prema stvarnom susjedstvu
    za svaki susjedni element koji je prepreka prilagodi
    karakteristične susjedne veličine
    izračunaj nove brzine prema diskretnom izrazu momentne
    jednadžbe
kraj metode
```

Pri tome su karakteristične susjedne veličine sve one vrijednosti tlaka i brzina koje su potrebne za rješavanje momentne jednadžbe kako je već prikazano u teoretskom dijelu. Unutar ove metode te veličine imaju imena dana prema stranama svijeta kako je prikazano na slici 9. Ovakav način zapisa je razumljiviji od korištenja indeksa polja,



Slika 9. Karakteristične susjedne veličine označene su imenima strana svijeta. Jedino tlak u tekućem elementu nema ovakvu oznaku

pogotovo uzme li se u obzir kompleksnost momentne jednadžbe u diskretnom obliku. Kod koji predstavlja samo tu jednadžbu, a koristi susjedne veličine prema slici 9 prikazan je u nastavku:

```
vel[i, j, 0] = oldVel[i, j, 0] - velUndRelax * deltaT * ((eastXVel
   * (eastXVel - westXVel) / deltaX/2 + northYVel * (eastNorthXVel
   - eastXVel) / deltaY/2) + (eastPressure - p[i, j]) / deltaX - 1
   / Re * ((farEastXVel - 2 * eastXVel + westXVel) / (deltaX *
   deltaX) + (eastNorthXVel - 2 * eastXVel + eastSouthXVel) /
   (deltaY * deltaY)));
vel[i, j, 1] = oldVel[i, j, 1] - velUndRelax * deltaT * ((eastXVel
   * (northEastYVel - northYVel) / deltaX/2 + northYVel *
   (northYVel - southYVel) / deltaY/2) + (northPressure - p[i, j])
   / deltaY - 1 / Re * ((northEastYVel - 2 * northYVel +
   northWestYVel) / (deltaX * deltaX) + (farNorthYVel - 2 *
   northYVel + southYVel) / (deltaY * deltaY)));
```

Ovaj oblik jednadžbi identičan je diskretnom zapisu danom u poglavlju o diskretizaciji momentne jednadžbe.

2.1.2.3 Proračun korekcija tlaka

Sljedeća metoda koja se poziva u `advanceTimeStep()` je metoda `calculatePCorrection()`. Njen pseudokod se može prikazati kako slijedi:

```
bool calculatePCorrection()
     za svaki element mreže
         postavi vrijednost korekcije tlaka na nula
         postavi vrijednost stare korekcije tlaka na nula
     ponavljaj do konvergencije ili najviše do maxIteracija puta
         postavi najveću korekciju tlaka na nula
         postavi najveću razliku korekcija tlaka na nula
         za svaki element mreže
              izračunaj korekciju tlaka
              ako je razlika nove i stare korekcije tekućeg
              elementa veća od dosad najveće razlike korekcije
              tlaka
                   postavi najveću razliku korekcije tlaka na
                   razliku nove i stare korekcije tekućeg elementa
              ako je korekcija tlaka tekućeg elementa veća od
              dosad najveće korekcije tlaka
                   postavi najveću korekciju tlaka na korekciju
                   tlaka tekućeg elementa
     ako je najveća razlika korekcije tlaka manja od uvjeta
     konvergencije
         korekcije tlaka su dostigle konačnu vrijednost; izađi iz
         metode
     inače
         za svaki element mreže
              postavi staru korekciju na vrijednost nove korekcije
kraj metode
```

Ova je metoda posebno bitna zbog iterativnog postupka koji dosad nije objašnjen. No, radi se o iznimno jednostavnom postupku. Nove vrijednosti korekcija tlaka se

izračunavaju sve dok najveća razlika korekcije između dvije uzastopne iteracije ne padne na prihvatljivu razinu. Unutar cijelog postupka, koji ne ovisi o ovome koraku, vrlo je bitno da se pamti najveća korekcija tlaka na mreži u apsolutnom iznosu. Ta je veličina, naime, mjerilo konvergencije algoritma SIMPLE.

Ova metoda za svaki element mreže poziva zasebnu metodu koja zapravo rješava Poissonovu jednadžbu. Njen je naziv `calculateOneElementNewPressure()`, a pseudokod se može prikazati na sljedeći način:

```
calculateOneElementNewPressure(i, j)

    vrijednosti karakterističnih susjednih veličina postavi
    prema stvarnom susjedstvu

    za svaki susjedni element koji je prepreka prilagodi
    karakteristične susjedne veličine

    izračunaj nove korekcije prema diskretnom izrazu Poissonove
    jednadžbe

kraj metode
```

Karakteristične susjedne veličine se označavaju jednako kao kod rješavanja momentne jednadžbe, s tim da ovdje postoje dvije dodatne varijable za tlak: `westPressure` i `southPressure`. Kod koji računa samu korekciju je

```
pCorrection[i, j] = (eastPressure + westPressure) * deltaT /
   (deltaX * deltaX) + (northPressure + southPressure) * deltaT /
   (deltaY * deltaY)  - (eastXVel - westXVel) / deltaX - (northYVel
   - southYVel) / deltaY;
pCorrection[i, j] /= 2 * deltaT * (deltaX * deltaX + deltaY *
   deltaY) / (deltaX * deltaX * deltaY * deltaY);
```

Jednadžba je podijeljena na dva dijela zbog bolje preglednosti.

2.1.2.4 Ostale metode

U klasi *Solver* postoje još neke dodatne pomoćne metode. Tako metoda `switchVelocities()` prije ulaska u glavnu petlju algoritma SIMPLE zapisuje tekuće vrijednosti brzina u polje starih brzina. Da bi se postupak ubrzao, umjesto prepisivanja svih vrijednosti jednostavno se zamijene samo imena polja, tj. zamijene se reference na ove objekte. Metoda `applyPCorrection()` vrši jednostavno zbrajanje tekućih vrijednosti tlaka i njihovih korekcija kako je predviđeno u četvrtom koraku algoritma SIMPLE. Slično, ali s brzinama, radi i `applyVelocityCorrections()`. Također, postoje metode za postavljanje različitih veličina na njihove početne vrijednosti.

2.1.3 Zaključak o *Solver*-u

Klasa *Solver* je u potpunosti zatvoreno programsko rješenje za simuliranje ponašanja fluida korištenjem algoritma SIMPLE. Sve potrebne veličine su sadržane u njoj i nijedan dio koda ne ovisi o ikakvoj vanjskoj veličini. Iz toga razloga moguće je u istom programu instancirati više objekata klase *Solver* i vršiti simulacije više sustava pri čemu svaki ima čitav skup potrebnih podataka i metoda. Također, klasa nije ograničena na korištenje u ovom programu i može se uz korištenje drugačijega koda za vizualizaciju koristiti u bilo kojem programu bez potreba za izmjenama.

2.2 Klasa Visualizer

Podaci:

- polja
  boje (colorMap)

- dodatne varijable
  zoom
  slika
  Solver solv
  Graphics gr

Klasa *Visualizer* sadrži metode potrebne za vizualizaciju rezultata dobivenih radom objekta klase *Solver*. Zbog toga što koristi rezultate iz toga objekta ova klasa mu mora imati pristup i u svom konstruktoru očekuje predavanje podatka tipa *Solver*. Drugi argument koji se mora predati pri instanciranju objekta ovoga tipa je objekt unutar *Form*-a na koji će se rezultati iscrtavati. Konstruktor klase izgleda kako slijedi:

```
public Visualizer(System.Windows.Forms.PictureBox picBox, int
problemWidth, int problemHeight, Solver solv)
{
    slika = new System.Drawing.Bitmap(picBox.Width, picBox.Height);
    picBox.Image = slika;
    gr = System.Drawing.Graphics.FromImage(slika);

    this.solv = solv;
    this.problemWidth = problemWidth;
    this.problemHeight = problemHeight;

    colorMap = new System.Drawing.Color[problemWidth, problemHeight];

    if ((double)picBox.Width/problemWidth >
    picBox.Height/problemHeight)
    {
        zoom = (double)picBox.Height / problemHeight;
        offsetX = (int)((picBox.Width - problemWidth * zoom) / 2);
        offsetY = 0;
    }
    else
    {
        zoom = (double)picBox.Width / problemWidth;
        offsetX = 0;
        offsetY = (int)((picBox.Height - problemHeight * zoom) / 2);
    }
}
```

U slučaju ovoga programa iscrtavanje se vrši na *PictureBox*, ali to ne mora biti slučaj. Klasa sadrži vlastitu referencu na objekt tipa *Solver* koja se izjednačava s dobivenom referencom i na taj način čini objekt `solv` iz pozivajućeg programa dostupnim svim metodama klase. Ovo je bitno jer taj objekt sadrži sve rezultate simulacije koji se žele

prikazati. Varijabla `gr` predstavlja objekt koji omogućava rad s metodama zaduženima za grafiku. U ovom programu bitne su mogućnosti iscrtavanja punih pravokutnika i linija. Boja tih grafičkih elemenata zadaje se preko objekata tipa `Pen` i `Brush`, koji su zato također dio ove klase.

Polje `System.Drawing.Color[,] colorMap` sadrži elemente tipa `Color`, koji definiraju `RGB` vrijednosti unutar polja veličine prostora simulacije, a što su neki od podataka u objektu `solv`. Ovo se polje koristi u metodama za iscrtavanje tlaka i brzina, što će biti pojašnjeno pri pregledu tih metoda.

Kako veličina objekta na kojeg se iscrtava ne mora odgovarati veličini simuliranoga sustava (i obično ne odgovara), varijabla `zoom` sadrži odnos ove dvije veličine što se kasnije koristi kod iscrtavanja. Ako objekt na koji se iscrtava i simulirani sustav nisu u jednakom formatu (tj. odnos stranica im nije jednak) iscrtavanje će se vršiti tako da se iscrtavanje prilagodi dužoj stranici sustava, što znači da će dio prostora za iscrtavanje ostati neiskorišten. Alternativa je 'rastezanje' slike, što je manje prihvatljivo. Varijable `offsetX` i `offsetY` služe za pozicioniranje iscrtanog dijela slike na sredinu bez obzira u kojem su odnosu stranice područja iscrtavanja i područja simulacije.

2.2.1 Metode vizualizacije

Klasa *Visualizer* podržava četiri različita načina iscrtavanja rezultata. To su prikaz polja tlaka, prikaz polja brzina, prikaz vektora brzina i na kraju strujnica fluida. Svaki od četiri načina je ostvaren u jednoj od četiri javne metode. Da bi željena veličina bila iscrtana, u glavnom se programu nakon instanciranja objekata klasa *Solver* i *Visualizer* i nakon pozivanja metoda objekta tipa *Solver* da bi se dobili neki rezultati, treba odrediti jedna od metoda za iscrtavanje iz ove klase.

2.2.1.1 Iscrtavanje tlaka

Prva u nizu metoda koje omogućuju vizualizaciju rezultata simulacije fluida je `drawPressure()`. Njen je pseudokod:

```
drawPressure()
     izračunaj srednju vrijednost tlaka
     za svaki element mreže
```

```
            izračunaj intenzitet tlaka u odnosu na srednju
            vrijednost skaliran na 0-255
            postavi odgovarajući element colorMap-a na izračunati
            intenzitet
      iscrtaj colorMap
kraj metode
```

Srednja vrijednost tlaka se računa da bi se dobila procjena raspona vrijednosti tlaka na cijeloj mreži. Naime, ako su sve vrijednosti jako male, dobivena slika će biti pretamna. Ako su vrijednosti visoke, slika će biti presvijetla. Jednostavno traženje maksimuma i minimuma tlaka također nije dovoljno. U tom slučaju se može dogoditi da praktički svi iznosi na mreži, osim na par elemenata, budu jako niski. To se može dogoditi pri toku oko oštre prepreke. Tad bi dobivena slika bila previše kontrastna. Većina promjena bi bila izgubljena u crnilu, dok bi samo nekoliko elemenata bilo osvijetljeno. Slično bi se dogodilo da je situacija obratna i da su iznosi tlaka vrlo visoki u većini elemenata, a relativno niski u njih par. Tada bi većina detalja bila izgubljena u presvijetlim nijansama.

Zbog takvih problema najjednostavniji pristup je koristiti srednju vrijednost. Neke će visoke vrijednosti postati previsoke i dostići maksimum osvijetljenosti gdje nema razlika u nijansama, dok će ekstremi na suprotnoj strani biti jako tamni, međutim najveća diferencijacija boja će biti upravo na onoj razini na kojoj se nalazi većina elemenata.

Izračunatim intenzitetima se puni polje `colorMap`, koje se na kraju iscrtava pozivom posebne metode.

2.2.1.2 Iscrtavanje brzina

Metoda `drawVelocities()` radi dosta slično prethodnoj metodi. No, pošto brzina ima dvije komponente, u *x* i u *y* smjeru, ova metoda omogućava i izbor prikaza jedne od njih. Argumenti su dvije logičke vrijednosti od kojih se jedna odnosi na brzinu u *x* smjeru, a druga na brzinu u *y* smjeru. Ako je argument logička jedinica, odgovarajuća brzina će se iscrtavati, a ako je logička nula, neće. Oba argumenta mogu biti *true*, što znači da će se iscrtavati obje komponente.

Pseudokod ove metode je gotovo identičan prethodnome:

```
drawVelocities(bool, bool)
      ako treba iscrtavati brzinu u x smjeru
            izračunaj srednju brzinu u x smjeru
```

```
        ako treba iscrtavati brzinu u y smjeru
            izračunaj srednju brzinu u y smjeru
        izračunaj ukupnu srednju brzinu
        za svaki element mreže
            ako treba iscrtavati brzinu u x smjeru
                izračunaj intenzitet brzine u x smjeru skaliran
                na 0-255
                postavi odgovarajući element colorMap-a na
                izračunati intenzitet
            inače
                postavi odgovarajući element colorMap-a na nulu
            ako treba iscrtavati brzinu u y smjeru
                izračunaj intenzitet brzine u y smjeru skaliran
                na 0-255
                postavi odgovarajući element colorMap-a na
                izračunati intenzitet
            inače
                postavi odgovarajući element colorMap-a na nulu
        iscrtaj colorMap
    kraj metode
```

Računanje srednjih vrijednosti brzina u određenim smjerovima identično je računanju srednjih vrijednosti tlaka. Ukupna srednja vrijednost se dobije kao aritmetička sredina srednjih vrijednosti u *x* i *y* smjeru.

2.2.1.3 Iscrtavanje iz mape boja

Na kraju obje prethodne metode se poziva metoda `drawFromColorMap()` koja iscrtava podatke sadržane u `colorMap`-u. Kod ove metode je jednostavan:

```
public void drawFromColorMap()
    for (int i = 0; i < problemWidth; i++)
    {
        for (int j = 0; j < problemHeight; j++)
        {
            b.Color = colorMap[i, j];
            gr.FillRectangle(b, (float)(offsetX + i * zoom),
            (float)(offsetY + j * zoom), (float)(zoom),
            (float)(zoom));
        }
    }
}
```

Dakle za svaki se element mreže jednostavno iscrtava pravokutnik boje pohranjene u `colorMap`-u. Mjesto iscrtavanja je određeno indeksom elementa s obzirom da mjesto u memoriji računala odgovara mjestu u simuliranom sustavu. Točne koordinate se dobijaju množenjem indeksa iznosom `zoom`, čije je značenje već objašnjeno. Istom varijablom određena je i veličina pravokutnika, odnosno kvadrata. Točnije veličina odgovara umnošku dimenzija jednog elementa mreže i vrijednosti `zoom`, ali kako su dimenzije mreže jedinične, mjere su kao u gornjem kodu. Cijelo je područje dodatno pomaknuto za iznose `offsetX` i `offsetY` u *x* i *y* smjerovima da bi se iscrtavanje vršilo centrirano u danome prostoru.

2.2.1.4 Iscrtavanje vektora brzina

Dosadašnje metode su direktno iscrtavale vrijednosti temeljnih veličina tlaka i brzine. Takav prikaz nije uvijek dovoljno informativan, koristan i, u krajnjoj liniji, zanimljiv. Iako je gledajući raznobojne brzine moguće zamisliti kako se fluid giba, prirodniji je prikaz smjera vektora brzine jer se na taj način neposredno može vidjeti smjer gibanja fluida. Sljedeće dvije metode pokušavaju upravo to.

Prva od njih je `drawStreamlines()`. Ova metoda iscrtava vektore brzina u čvorovima mreže. Krajnja slika u ovome slučaju će dakle biti niz linija usmjerenih u ovisnosti o vektoru brzine. Duljina će ovisiti o normi vektora. Linije su pozicionirane u centrima elemenata mreže, tj. polovišta linija su točno na koordinatama čvorova. Kod ove metode:

```
public void drawStreamlines ()
{
      double zoomDivider = 20;
      double meanVelocity = (getMeanVelValue(0) +
      getMeanVelValue(1)) / 2;
      clearPicBox(System.Drawing.Color.Black);
      for (int i = 0; i < problemWidth; i++)
      {
           for (int j = 0; j < problemHeight; j++)
           {
                p.Color = System.Drawing.Color.Red;

                gr.DrawLine(p, (float)(offsetX + i * zoom - 0.5 *
                solv.vel[i, j, 0] / meanVelocity *
                zoom/zoomDivider), (float)(offsetY + j * zoom - 0.5
                * solv.vel[i, j, 1] / meanVelocity *
                zoom/zoomDivider), (float)(offsetX + i * zoom),
                (float)(offsetY + j * zoom));

                p.Color = System.Drawing.Color.White;
```

```
                    gr.DrawLine(p, (float)(offsetX + i * zoom),
                    (float)(offsetY + j * zoom), (float)(offsetX + i *
                    zoom + 0.5 * solv.vel[i, j, 0] / meanVelocity *
                    zoom/zoomDivider), (float)(offsetY + j * zoom + 0.5
                    * solv.vel[i, j, 1] / meanVelocity *
                    zoom/zoomDivider));
                }
            }
            obstacleVisualization();
        }
```

Dodatna varijabla `zoomDivider` je dodana da bi se omogućilo lakše skaliranje duljine linija. Naime, razlike ekstremnih vrijednosti brzina su tolike da u nekim slučajevima duljine linija na elementima s maksimalnim iznosima brzina prelaze nekoliko elemenata ili čak cijeli prostor simulacije dok na elementima s minimalnim iznosima brzina ne prelaze širinu jednoga *piksela* i nisu vidljive. Ovisno o tome koje se područje promatra, skaliranje postaje nužno. Jedna od ideja koje bi se mogle primijeniti na ovaj problem je nelinearna ovisnost duljine linija i iznosa brzina. Logaritamska ljestvica bi, na primjer, vjerojatno dala bolje rezultate. Međutim, tada bi se posebna pažnja trebala posvetiti negativnim veličinama koje nisu prihvatljiva podlogaritamska vrijednost.

2.2.1.5 Iscrtavanje strujnica

Druga metoda koja iscrtava smjerove gibanja fluida trebala bi dati najbolje rezultate, ali zato je računalno najzahtjevnija i znatno usporava rad aplikacije. Problem s prethodnom metodom je da iscrtava linije koje predstavljaju brzine u čvorovima mreže. Međutim, iako diskretiziran, sustav se simulira da bi se dobile kontinuirane veličine. Skokoviti prikaz brzina, iako informativan, a ujedno računalno nezahtjevan, nije vizualno najatraktivniji. Ideja na kojoj počiva metoda `drawStreams()` je interpolacija. Stvara se novo polje `double[,] marker`. Ovo polje sadrži proizvoljan broj elemenata, čestica, koje su inicijalno postavljene jednoliko u sustavu. Svaka se čestica zatim pomiče u ovisnosti o brzini na mjestu na kojem se nalazi. Ako se ne nalazi točno na nekom od čvorova, što je s *double* preciznošću položaja gotovo sigurno, brzinu je potrebno dobiti interpolacijom četiri susjedna iznosa. Nakon pomaka u ovisnosti o tom interpoliranom iznosu na novom će položaju trebati ponovno računati brzinu. Ovo se ponavlja za svaki pomak i za svaku česticu uvedenu u sustav. Očito je da su računalni zahtjevi ogromni. Potrebe iscrtavanja u praksi za red veličine premašuju potrebe iteracijskog postupka algoritma SIMPLE.

Kod metode drawStreams() je najkompliciraniji od dosad prikazanih u ovoj klasi. U pseudokodu izgleda kako slijedi:

```
drawStreams()
      za onoliko čestica koliko ih se traži
            postavi inicijalni položaj čestice
      za sve čestice
            izračunaj interpoliranu brzinu na položaju čestice
            ako čestica nije izašla iz područja simulacije
                  iscrtaj česticu
            pomakni česticu u ovisnosti o novoizračunatoj brzini
kraj metode
```

S obzirom da se simulira fluid koji je zatvoren u simuliranom sustavu, naizgled je provjeravanje izlaska čestice iz područja simulacije nepotrebno. Naime, ako fluid ne može izaći, a čestice efektivno putuju nošene fluidom, to ne bi trebalo biti moguće. Postoje ipak najmanje dva razloga zašto je takav događaj ipak moguć. Prvo, da bi se ubrzalo iscrtavanje i naglasile strujnice čestice se kreću brzinama koje su veće od brzine fluida, ali linearno ovisne o njoj. Tako čestica može izaći iz sustava tamo gdje bi fluid naglo skrenuo. Drugo, ako se radi o sustavu kroz koji fluid protječe, dakle ima utok i istok, tada i fluid napušta sustav, noseći čestice sa sobom. Pitanje što učiniti s takvim česticama se može obraditi na niz načina. U ovoj se implementaciji jednostavno zanemaruju.

Najzahtjevniji dio gornjeg pseudokoda je računanje interpoliranih brzina. Zbog bolje preglednosti ovaj je dio koda izdvojen u posebnu metodu. Njen kod je:

```
private void getMarkerVel (double[,,] marker, int i, int j,
double[] vel)
{
    double[] A = new double[2];
    double[] B = new double[2];
    double a1, a2, b1, b2;
    int ix, iy;

    ix = (int)marker[i, j, 0];
    iy = (int)marker[i, j, 1];

    a1 = marker[i, j, 0] - (int)marker[i, j, 0];
    a2 = 1 - a1;
    b1 = marker[i, j, 1] - (int)marker[i, j, 1];
```

```
        b2 = 1 - b1;

        try
        {
            A[0] = a2 * solv.vel[ix, iy, 0] + a1 * solv.vel[ix + 1, iy,
            0];
            A[1] = a2 * solv.vel[ix, iy, 1] + a1 * solv.vel[ix + 1, iy,
            1];
            B[0] = a2 * solv.vel[ix, iy + 1, 0] + a1 * solv.vel[ix + 1,
            iy + 1, 0];
            B[1] = a2 * solv.vel[ix, iy + 1, 1] + a1 * solv.vel[ix + 1,
            iy + 1, 1];
        }
        catch
        {
            vel[0] = 0;
            vel[1] = 0;

            return;
        }

        vel[0] = b2 * A[0] + b1 * B[0];
        vel[1] = b2 * A[1] + b1 * B[1];
    }
```

Sam je kod jednostavan. Varijable `a1`, `a2`, `b1` i `b2` sadrže udaljenosti od lijevih i desnih, odnosno gornjih i donjih susjeda. Odnos udaljenosti govori koliki je utjecaj svakog od čvorova. Polja `A` i `B` su pomoćna i sadrže međurezultate. Krajnji rezultat se dakle dobija u dva stupnja, prvo interpoliranjem gornjeg i donjeg para susjeda po *x* osi, a zatim interpoliranjem ta dva rezultata po *y* osi. Iako jednostavan, postupak se izvršava za svaki pomak za svaku česticu, iz čega proizlazi računalna zahtjevnost.

2.2.2 Zaključak o *Visualizer*-u

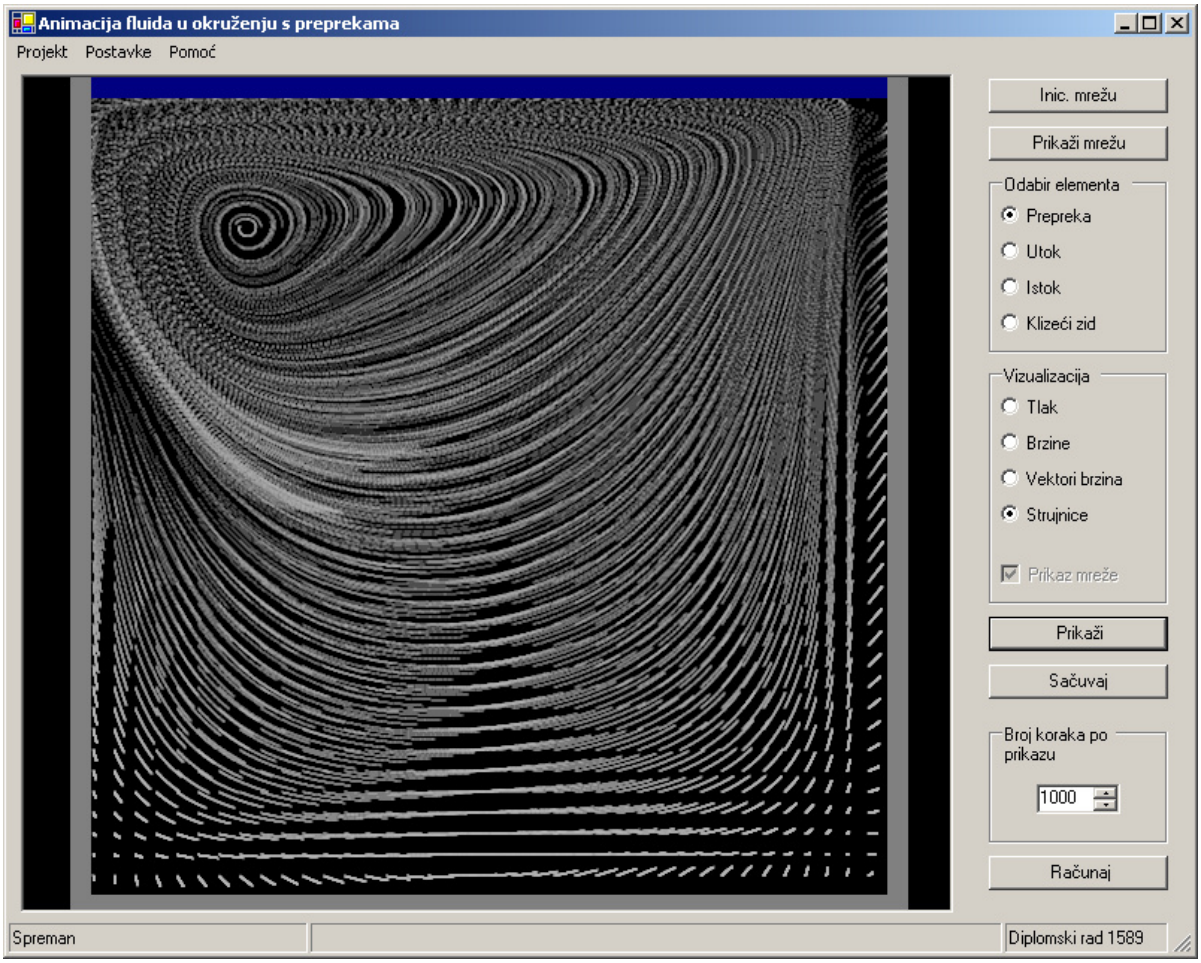
Klasa Visualizer je pomoćni dio programa. Sve bitno za simulaciju fluida je sadržano u klasi *Solver*. Međutim, rezultati ne znače puno ako stoje u memoriji računala. Neki oblik vizualizacije je u svakom slučaju potreban. Algoritam SIMPLE kako je objašnjen funkcionira jednako dobro za dvodimenzionalni i trodimenzionalni sustav. Metode klase *Solver* se mogu vrlo lako promijeniti tako da rade s poljima bogatijima za jednu dimenziju. Međutim, specifične potrebe vizualizacije strujanja trodimenzionalnih fluida su razlog zašto tako nešto nije napravljeno. Naime, vizualizacija dvodimenzionalnog sustava je manje više trivijalna, s obzirom da su svi elementi u svakom trenutku vidljivi. U trodimenzionalnom slučaju elementi zaklanjaju jedan drugoga. Za prikaz takvoga sustava potrebno je odabrati zanimljive elemente (npr. one koji imaju određenu brzinu, ili su dijelovi vrtloga i slično), od njih stvoriti trodimenzionalni objekt, zatim tome objektu

detektirati površinu i na kraju je iscrtati. To je proces bitno drukčiji od dvodimenzionalnoga.

U ovoj su klasi implementirane metode koje iscrtavaju određene veličine ili izvedenice veličina dobivenih primjenom metoda klase *Solver*. Iz toga razloga metode ove klase imaju direktan pristup tim ključnim podacima. Mogućnosti vizualizacije koje daju poznate vrijednosti polja brzine i tlaka su mnogobrojne. Ova klasa podržava samo neke najočitije mogućnosti. Ideje koje se još nameću su detekcija i specijalizirano iscrtavanje laminarnog toka, prikaz iznosa vrtložnosti i funkcije toka, gradijenta brzina i tlaka, povezanosti polja brzina i tlaka i slično.

2.3 Korisničko sučelje

Korisničko sučelje se sastoji od dva osnovna dijela kako se vidi iz slike 10. Najveći dio prozora aplikacije zauzima *pictureBox* korišten za vizualizaciju rezultata rada *Solver*-a. Drugi element sučelja nalazi se uz desni rub prozora i sadrži elemente namijenjene osnovnom upravljanju radom.



Slika 10. Izgled glavnog prozora programa. Prozorom dominira područje iscrtavanja rezultata simulacije. S desne strane su dostupne glavne kontrole programa, dok su dodatne dostupne putem glavnog izbornika. Na dnu prozora se ispisuju poruke o stanju simulacije.

Klikom na tipku "Inic. mreže" pokreće se postupak inicijalizacije simulacijske mreže. Korisniku se nudi mogućnost određivanja širine i visine mreže. Ove će mjere poslužiti za instanciranje novih objekata tipa *Solver* i *Visualizer*.

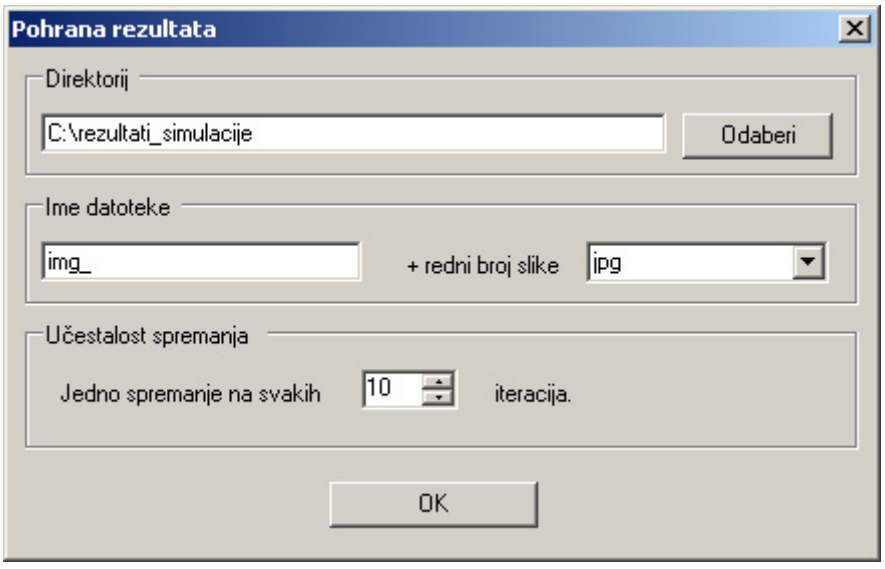
Korištenjem miša moguće je iscrtavati prepreke na području simulacije pri čemu je zbog olakšanog rada prisutan i prikaz mreže. Klikom na desnu tipku miša briše se određeni element, tj. označava se kao element ispunjen fluidom.

Funkcija lijeve tipke miša ovisi o izboru označenome pod "Odabir elementa". Moguće je postavljanje prepreke, utoka fluida u sustav, istoka iz njega ili postavljanje klizećeg (pomičnog) zida za rješavanje problema kao na slici.

Pod grupom opcija "Vizualizacija" nudi se odabir vrste prikaza, pri čemu se može birati između prikaza tlaka, brzina, vektora brzina ili strujnica. Za prve dvije mogućnosti, također je moguće odabrati da li će se ujedno prikazivati i mreža ili ne. Pri iscrtavanju vektora brzina i strujnica iscrtavanje mreže se ne nudi jer se radi o načinima prikaza čiji elementi prelaze granice elemenata mreže.

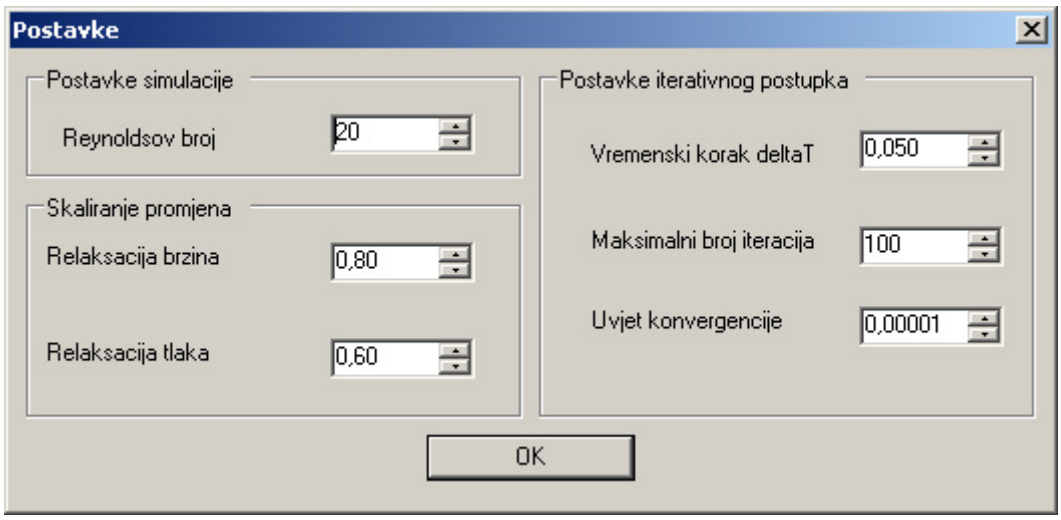
Pritiskom na tipku "Računaj" pokrenuti će se simulacija i dok traje, na donjem dijelu prozora ispisivati će se poruke o statusu programa. Kada završi, poruka će glasiti "Spreman". Ako je odabrano spremanje rezultata, ono će se vršiti automatski, a nakon izvršenog proračuna iscrtati će se dobiveni rezultat na način koji je odabran pod "Vizualizacija".

S obzirom da vizualizacija ima relativno velike potrebe za računalnim resursima, a iscrtavanje strujnica i iznimno velike, moguće je odabrati odnos broja vremenskih koraka simulacije i broja osvježavanja prikaza rezultata. Odabir broja deset bi, na primjer, značio da će između svaka dva osvježavanja prikaza biti odrađeno deset vremenskih koraka simulacije.



Slika 12. Izgled prozora za odabir postavki automatske pohrane rezultata

Dodatna tipka "Sačuvaj" nudi korisniku prozor u kojem može odabrati mjesto na tvrdom disku gdje će biti sačuvane četiri slike odabranoga formata i pod odabranim imenom (a s nastavcima "pressure", "velocities", "streamlines" i "streams") koje odgovaraju svim četirima načinima vizualizacije rezultata. Ukoliko je potrebno rezultate spremati u svakom koraku (ili jednom u određenom broju koraka), ovo se može odrediti u glavnom izborniku na Postavke→Pohrana rezultata gdje je moguće odrediti put do datoteke, ime (kojemu će biti dodani redni brojevi koraka u kojima je vršeno spremanje) te učestalost spremanja (broj koraka simulacije među svaka dva spremanja. Izgled ovoga izbornika se može vidjeti na slici 12.



Slika 13. Izgled prozora za odabir postavki simulacije

Dodatne postavke programa vezane za samu simulaciju se mogu prilagoditi u izborniku "Postavke" pod opcijom "Osnovne postavke". Izgled ovoga prozora je prikazan na slici 13. Moguće je promijeniti Reynoldsov broj, koeficijente skaliranja brzina i tlaka, vremenski korak simulacije, maksimalni broj iteracija koje će simulacija izvršiti u pokušaju da dođe do konvergencije i na kraju, iznos samog uvjeta konvergencije.

2.4 Zaključak o implementaciji

Izbor C#-a za programski jezik se na kraju pokazao pozitivnim. Jednostavnost poznate sintakse je najveća pomoć, iako, kako je već rečeno, neka poboljšanja koja jezik donosi zapravo otežavaju rad s njim u posebnim situacijama. Za primjer, može se spomenuti rad s grafičkim elementima. Podržan je veliki broj grafičkih elemenata, kao što su razni geometrijski likovi, linije, krivulje i slično. Da bi se olakšao problem prilagođavanja veličine u slučaju promjene prostora na kojem se odvija iscrtavanje, proces je u velikoj mjeri automatiziran. Zato je većina koordinata dana u relativnim iznosima. Posljedica ovakvog olakšavanja programiranja je da standardne biblioteke, apsurdno, ne podržavaju iscrtavanje jednog jedinog *piksela*. Najbolji prijedlog koji se može pronaći na *Internet*-u je stvaranje objekta tipa *Bitmap* dimenzija 1x1, koji će se puniti potrebnom bojom i konstantno kopirati na potrebne koordinate u području iscrtavanja. Usporavanje programa u ovakvom načinu rada može se naslutiti.

Ipak, prednosti nadjačavaju mane.

Cilj je bio izraditi razumljiv, izrazito modularan program s maksimalno specijaliziranim zadaćama svakoga dijela. U tome je izbor objektno orijentiranog jezika bio od velike pomoći. Nadalje, po prirodi iterativnog postupka je bilo moguće podijeliti glavninu posla na čvrsto uokvirene korake. Zbog toga je moguće pratiti rad algoritma iz koraka u korak i iz iteracije u iteraciju. U tom smislu, program je ostvario postavljeni cilj.

2.5 Rezultati

U nastavku je prikazano nekoliko primjera korištenja izrađenoga programa. Za sve je probleme prikazana vizualizacija tlaka, brzina te vektora brzina i strujnica.

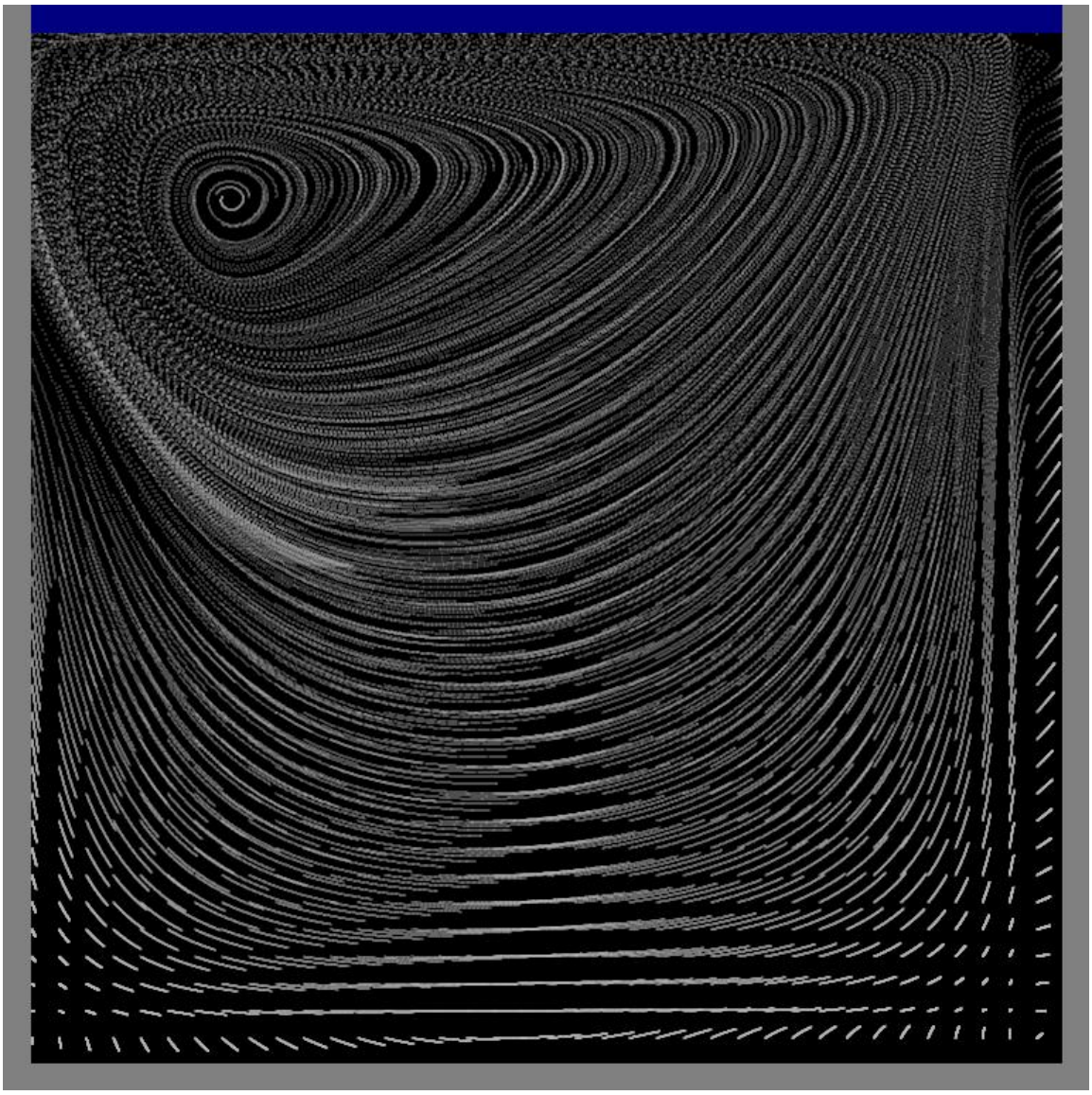
Kod vizualizacije tlaka svjetlija nijansa plave boje znači veće odstupanje od srednje vrijednosti tlaka. Naime, tlakovi mogu biti i pozitivni i negativni, kako je već raspravljeno, ali su za momentnu i Poissonovu jednadžbu važne razlike tlakova, a ne njihove apsolutne brzine.

Vizualizacija polja brzina se vrši kroz prikaz intenziteta brzina u x i y smjeru. Komponenta u x smjeru je prikazana crvenom, a u y smjeru zelenom bojom. Tamo gdje su obje komponente prisutne u podjednakom iznosu boja prelazi u žutu. Svjetlija nijansa znači veću brzinu.

Kod prikaza vektora brzina iscrtane linije pokazuju smjer kretanja fluida u točki koja je točno na polovici linije, dok njena duljina ovisi o iznosu brzine.

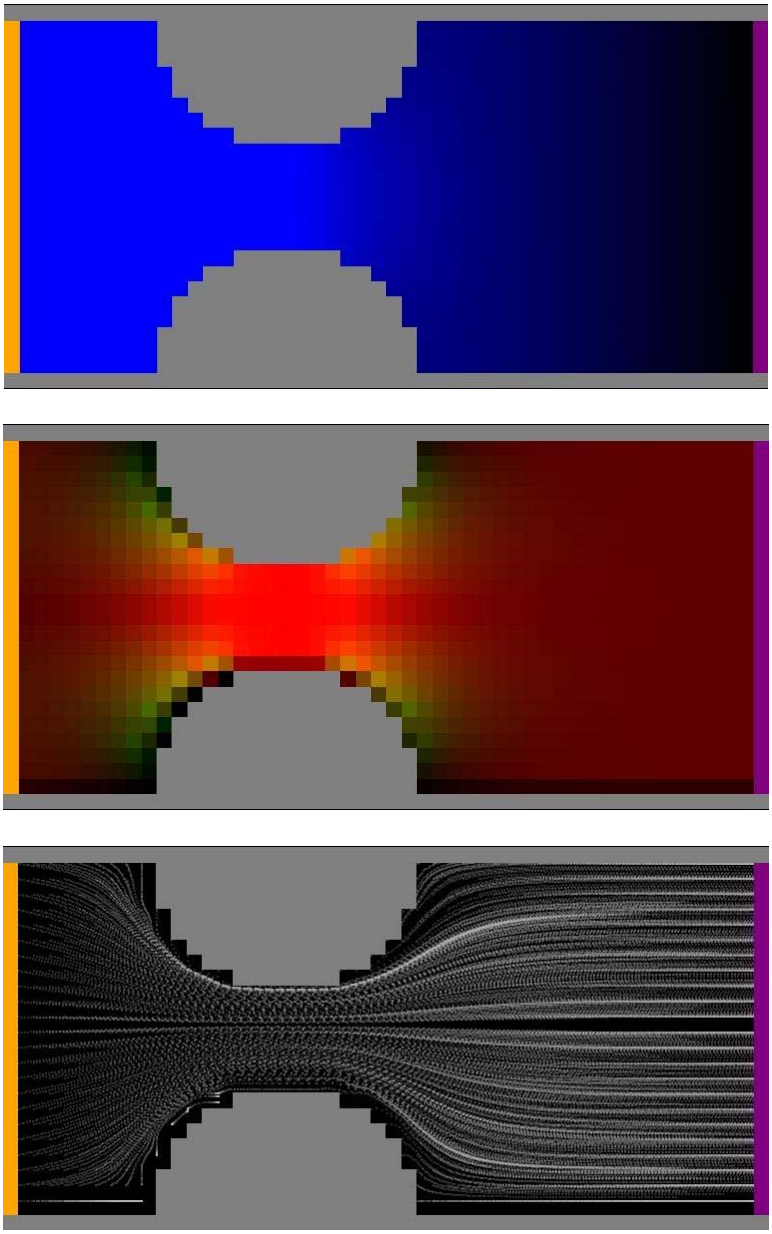
Prikaz strujnica se dobija puštanjem imaginarnih čestica niz tok fluida. Iako nesavršene, zbog mogućih loših interpolacija u blizini rubova i prepreka, daju glađu sliku od one koju prikazuju vektori brzina i u većini slučajeva predstavljaju najzanimljiviji prikaz.

2.5.1 Šupljina s pomičnim zidom (eng. *driven lid cavity*)



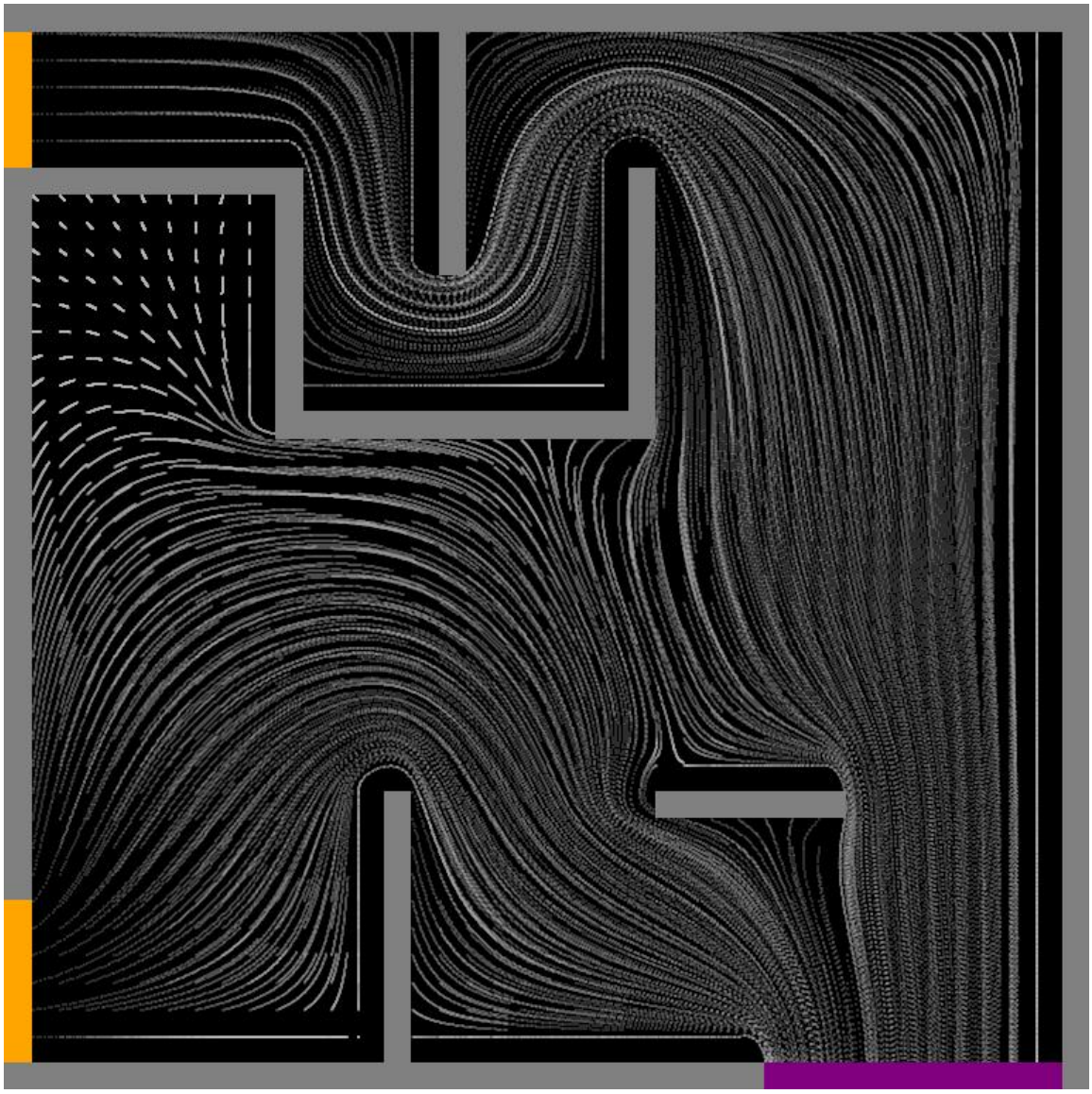
Ovo je jedan od tipičnih problema računalne simulacije fluida. Gornja granica sustava se jednoliko giba ulijevo, povlačeći za sobom rubne slojeve fluida, koji dalje povlače sljedeće slojeve. U cijeloj šupljini nastaje kruženje koje se na iscrtavanju strujnica najbolje vidi. Centar rotacije se pomiče u smjeru gibanja pomičnoga zida.

2.5.2 Tok između dvije kugle



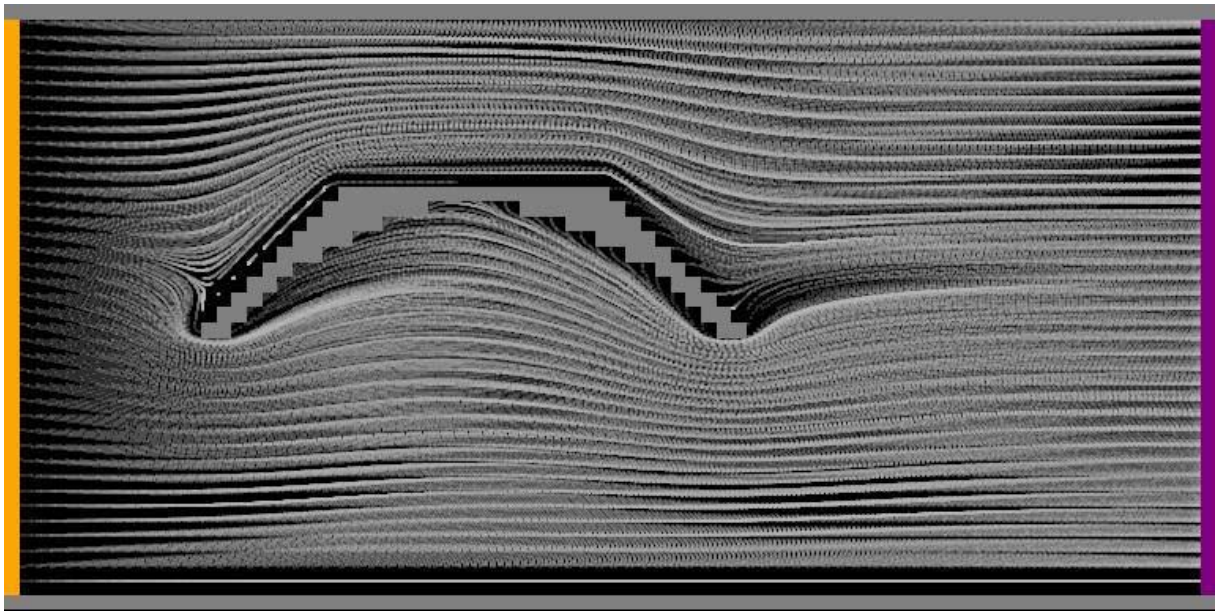
U ovom primjeru fluid ulazi u sustav s lijeve strane, a izlazi s desne. Prepreka je u obliku dvije nasuprot postavljene polukugle. Na prikazu tlaka se vidi opadanje vrijednosti nakon prolaska kroz tjesnac. Prikaz brzina pokazuje najveću brzinu točno između kugli, kako je i očekivano. Strujanje fluida između dvije kugle stvara privlačnu silu među njima kao posljedicu visoke brzine i zbog toga niskoga statičkoga tlaka. Na prikazu strujnica se vidi glatko prilagođavanje toka fluida preprekama.

2.5.3 Primjer 3. – Tok kroz kompleksne prepreke



Ovaj primjer služi za prikaz rada algoritma u slučaju zahtjevnijih sustava. Utoci fluida su na lijevoj gornjoj i lijevoj donjoj strani, označeni narančastim granicama, a istok na donjoj desnoj, označen ljubičastom bojom. Na prikazu strujnica treba obratiti pažnju na tri detalja. Prvo, gladak tok kroz kanal u gornjem dijelu slike gdje su razlike brzina kao posljedica uskoga prostora toka velike. Drugo, tok oko okomite prepreke u donjem lijevom dijelu sustava i na kraju, razdvajanje i ponovno spajanje toka ispred i iza slobodne prepreke u donjem desnom dijelu sustava.

2.5.4 Udubljena prepreka (frizbi)



Gornja slika prikazuje tok fluida oko stiliziranog frizbija. Primjer je zanimljiv utoliko što se jasno vidi razdvajanje toka fluida ispred i ponovno spajanje iza prepreke. Također, na cijelom donjem dijelu sustava može se pratiti reakcija strujnica na udubljeni dio prepreke. Strujnice se međusobno udaljavaju, pokazujući sporiji tok i time veći statički tlak. S gornje strane strujnice su gušće, što ukazuje na veću brzinu i manji statički tlak. Ukupan efekt je postojanje uzgonske sile na prepreku uzrokovane razlikom tlakova.

Dodatak A:

Operator ∇je standardni matematički operator koji stoji kao kratica za

$$\nabla = \frac{\delta}{\delta x}\vec{i} + \frac{\delta}{\delta y}\vec{j} + \frac{\delta}{\delta z}\vec{k} \qquad \text{(A.1)}$$

Djelovanje operatora je dvojako s obzirom da sadrži operatore deriviranja, ali je prvenstveno vektor. Deriviranje, ako se vrši, vrši se tek pošto su obavljene vektorske operacije.

U ovisnosti na što i kako djeluje, moguće su tri različite operacije. Sve tri proizlaze iz vektorske naravi operatora. Djelovanje može biti na skalar, na vektor skalarno ili na vektor vektorski. U prvom slučaju, za neki skalar φ rezultat će biti

$$\nabla\varphi = \frac{\delta\varphi}{\delta x}\vec{i} + \frac{\delta\varphi}{\delta y}\vec{j} + \frac{\delta\varphi}{\delta z}\vec{k} \qquad \text{(A.2)}$$

U slučaju skalarnog umnoška s vektorom rezultat operacije će u slučaju nekog vektora **v** biti

$$\nabla \bullet \vec{v} = \frac{\delta v_x}{\delta x} + \frac{\delta v_y}{\delta y} + \frac{\delta v_z}{\delta z} \qquad \text{(A.3)}$$

Na kraju, u slučaju vektorskog umnoška s vektorom **v** rezultat operacije će biti

$$\nabla \times \vec{v} = \left(\frac{\delta v_z}{\delta y} - \frac{\delta v_y}{\delta z}\right)\vec{i} - \left(\frac{\delta v_z}{\delta x} - \frac{\delta v_x}{\delta z}\right)\vec{j} + \left(\frac{\delta v_y}{\delta x} - \frac{\delta v_x}{\delta y}\right)\vec{k} \qquad \text{(A.4)}$$

Ovo posljednje se ne koristi u Navier Stokesovim jednadžbama, ali je ovdje dano zbog potpunosti prikaza operatora. Također, treći oblik ima smisla isključivo u trodimenzionalnom sustavu zbog naravi vektorskog umnoška, dok prva dva vrijede za bilo koji sustav.

Dodatak B:

Izuzme li se iz klasičnog oblika Navier Stokesovih jednadžbi kako su ispisane u (1.1.1) utjecaj vanjske sile, u ovom slučaju gravitacije, dobije se:

$$\frac{\delta \vec{u}}{\delta t} = -(\vec{u}\nabla)\vec{u} - \frac{1}{\rho}\nabla p + \frac{\nu}{\rho}\nabla^2 \vec{u} \qquad \text{(B.1)}$$

Ovaj zapis se može pretvoriti u bezdimenzionalni oblik uzmu li se u obzir sljedeće supstitucije:

$x' \equiv x / L$

$y' \equiv y / L$

$\mathbf{u}' \equiv \mathbf{u} / U$

$p' \equiv p / \rho U^2$

$$\nabla' \equiv \vec{i}\frac{\delta}{\delta x'} + \vec{j}\frac{\delta}{\delta y'} \equiv L\nabla$$

$t' \equiv t * U / L$

gdje su x i y koordinate dvodimenzionalnog koordinatnog sustava, **u** vektor brzine, p tlak i t vrijeme. L i U su karakteristična duljina i brzina sustava. Ove dvije veličine mogu biti po volji odabrane konstante, ali je bitno da se u dva usporediva modela mjere na potpuno jednak način i na jednakim mjestima (npr. ukupna duljina krila i brzina na samom kraju krila). Uvrste li se ove veličine u jednadžbu (B.1), dobija se:

$$\frac{\delta(U\vec{u}')}{\delta(\frac{L}{U}t')} = -\left(U\vec{u}'\cdot\frac{1}{L}\nabla'\right)U\vec{u}' - \frac{1}{\rho}\frac{1}{L}\nabla'\rho U^2 p' + \frac{\nu}{\rho}\frac{1}{L^2}\nabla'^2(U\vec{u}') \qquad \text{(B.2)}$$

Pretpostavljajući konstantnu gustoću i množenjem ove jednadžbe s L/U^2 dobija se

$$\frac{\delta \vec{u}'}{\delta t'} = -(\vec{u}'\cdot\nabla')\vec{u}' - \nabla' p' + \frac{\nu}{\rho}\frac{1}{LU}\nabla'^2 \vec{u}' \qquad \text{(B.3)}$$

Izraz uz posljednji element jednadžbe je dobiven iz raznih konstanti, pa se jednostavnije zapisuje kao bezdimenzionalni Reynoldsov broj. Ovaj broj pokazuje omjer inercijskog i viskoznog djelovanja na fluid:

$$\mathrm{Re} = \frac{Inercija}{Viskoznost} = \frac{\rho LU}{\nu} \qquad \text{(B.4)}$$

Potrebno je napomenuti da se u literaturi pojavljuju i izrazi Reynoldsovog broja u kojima se upotrebljava dinamička viskoznost. Kako su kinematička i dinamička viskoznost međusobno proporcionalne s koeficijentom proporcionalnosti jednakim gustoći fluida, svejedno je koji će se od dva oblika upotrebljavati. Gornji izraz je korišten zbog direktne povezanosti s jednadžbom (1.1.1).

Nakon uvrštavanja (B.4) u (B.3) i odbacivanja apostrofa zbog jednostavnijeg zapisa dobija se bezdimenzionalni oblik Navier Stokesovih jednadžbi:

$$\frac{\delta \vec{u}}{\delta t} = -(\vec{u} \cdot \nabla)\vec{u} - \nabla p + \frac{1}{\mathrm{Re}} \nabla^2 \vec{u} \qquad \text{(B.5)}$$

Ovaj je oblik jednak jednadžbi (1.1.3).

Dodatak C:

Ime SIMPLE algoritma znači "djelomično implicitna metoda za tlakom povezane jednadžbe". S obzirom da algoritam naizmjence rješava momentnu i Poissonovu jednadžbu, ti se utjecaji isprepliću i dovode do povećanja kompleksnosti algoritma. Da bi se točno izračunala korekcija tlaka u bilo kojem elementu, potrebno je koristiti vrijednosti korekcija na susjednim lokacijama. S obzirom da ovo vrijedi za svaki element, vidljivo je da bi takav pristup rezultirao potrebom korištenja svih korekcija tlaka na cijeloj mreži za proračun korekcija svakoga elementa. Ovakva metoda vodi prema direktnoj metodi rješavanja sustava jednadžbi, što se zbog veličine sustava svakako želi izbjeći.

Pokazuje se, međutim, da način rješavanja Poissonove jednadžbe ne utječe na konačno rješenje, jer je cilj dobiti polje tlaka bez dodatnih potrebnih korekcija. Na taj način se može zaključiti da korekcije tlaka susjednih elemenata neće utjecati na korekciju tekućeg elementa jednostavno zato što će u trenutku konvergencije rezultata sve korekcije biti nula. Iz ovog razloga se sav utjecaj susjednih korekcija jednostavno izbacuje iz jednadžbi, uzrokujući pogreške u prvim koracima. Takve pogreške u iterativnim metodama ne samo da nisu problem, nego su i očekivane, dok god se na kraju ipak postiže konvergencija.

Na ovaj se način algoritam pojednostavljuje i cijeli postupak rješavanja se skraćuje, pa tako i cijela simulacija ubrzava. U svemu tome zbog naravi Poissonove jednadžbe ne dolazi do pogrešaka u krajnjem rezultatu, tj. iznosima brzina i tlaka.

Iako je metoda rješavanja Poissonove jednadžbe dakle nebitna dok se postiže konvergencija, o njenom pravilnom izboru ovisi brzina dolaska do konvergencije. Naime, iz jednadžbe se može izbaciti bilo koji utjecaj i još uvijek će vrijediti činjenica da konačne korekcije ionako moraju biti nula i da zanemarivanje bilo kojeg izraza neće utjecati na točnost rješenja. Međutim ovo vrijedi samo uz pretpostavku da se do rješenja zapravo i dođe.

Zbog zanemarivanja utjecaja susjednih elemenata na korekciju tlaka, metoda nosi naziv "djelomično implicitna".

Zaključak

U ovom su radu iznesene osnovne ideje modeliranja sustava, matematička podloga modela ponašanja fluida i tehnike diskretizacije diferencijalnih jednadžbi. Riješeni su neki problemi koji su se pokazali ključnima, a zatim je izložen jedan od algoritama rješavanja sustava diferencijalnih jednadžbi koje opisuju model toka nestlačivih fluida. Na kraju je prikazana programska implementacija prikazanoga algoritma. Na taj način zatvoren je čitav krug od osnovnih zamisli modeliranja i simuliranja i fundamentalnih zakona fizike do ostvarenja gotovog proizvoda.

Područje simulacije fluida vrlo je živo područje na kojem radi veliki broj znanstvenika u raznim granama znanosti. Ovaj rad je pokrio najosnovnije pojmove i probleme područja.

Logičan nastavak bi bio proširiti problem na tri dimenzije i uvesti detekciju granica fluida. Posebno zanimljivo područje predstavlja simulacija fluida drugom filozofijom, koristeći čestični model. U ovom slučaju na jednostavan način se mogu dobiti efekti viskoznosti, elastičnosti i plastičnosti, a rekonstrukcija površine je daleko jednostavniji problem. Određeni hibridni modeli su također mogući.

Literatura:

[1] Ben Albahari, Peter Drayton, Brad Merrill, 2002, "C# essentials, 2nd edition"

[2] Marek Behr, 2001, "Introduction to simulation"

[3] Peter Birtles, 2000, "Fluid simulation"

[4] T. J. Chung, 2002, "Computational fluid dynamics"

[5] Simon Clavet, Phillipe Beaudoin, Pierre Poulin, 2005, "Particle-based viscoelastic fluid simulation"

[6] J. H. Ferziger, M. Perić, 2002, "Computational methods for fluid dynamics"

[7] Nick Foster, Ronald Fedkiw, "Practical animation of liquids"

[8] Petar Javor, 2000, "Matematička analiza 2"

[9] Maciey Matyka, 2004, "Solution to two-dimensional incompressible Navier-Stokes equations with SIMPLE, SIMPLER and Vorticity-Stream function approaches. Driven-lid cavity: Solution and visualization"

[10] Suhas V Patankar, 1980, "Numerical heat transfer and fluid flow"

[11] Freddy Perez, Wilson Rivera, 1999, "An object oriented framework for computational fluid dynamics simulations"

[12] Simon Premože, Tolga Tasdizen, James Bigler, Aaron Lefohn, Ross T. Whitaker, 2003, "Particle based simulation of fluids"

[13] Dennis C. Prieve, 2000, "A course in fluid mechanics with vector field theory"

[14] Ljiljana Pilić Rabadan, 1993, "Mehanika fluida"

[15] Carlos Eduardo Scheidegger, João Luiz Dihl Comba, Rudnei Dias sa Cunha, 2004, "Navier-Stokes on programmable graphics hardware using SMAC"

[16] Franz Vesely, 2001, "Computational physics – An introduction"

Sažetak:

Računalna simulacija fluida je znanstveno područje koje se jako brzo razvija. Zbog potreba simulacije raznovrsnih problema postoji veliki broj specijaliziranih algoritama za specifične namjene. Za potrebe razumijevanja područja, međutim, potrebno je krenuti od osnova. Upravo to je cilj ovoga rada. Izloženi su osnovni principi modeliranja, predstavljene prednosti i mane dva temeljna principa modeliranja fluida i izrađen matematički model jednadžbi koje određuju ponašanje fluida. Dan je pregled principa diskretizacije diferencijalnih jednadžbi i posebno, formulacija Navier Stokesovih jednadžbi u diskretnom obliku, prilagođenih primjeni na računalu. Opisan je algoritam SIMPLE kao jedan od osnovnih algoritama rješavanja ovih jednadžbi. Algoritam je implementiran na računalu i prikazani su rezultati rada programa.

Ključne riječi: računalna simulacija fluida, Navier Stokesove jednadžbe, algoritam SIMPLE

Abstract:

Computational fluid simulation is a fast growing area of scientific research. Simulating a large number of different problems induces development of many specialized algorithms for specific purposes. Understanding the area, however, means building from the ground. This is the exact goal of this thesis. Basic principles of modeling where described, strengths and weaknesses of two elemental approaches to fluid modeling where given and mathematical model of governing equations was presented. Furthermore, basic principles of derivation of discretized equations where shown and especially used to formulate computer friendly form of Navier Stokes equations. SIMPLE algorithm, as one of basic Navier Stokes solver algorithms, was presented and implemented in a computer program.

Keywords: computational fluid dynamics, simulation of fluids, Navier Stokes equations, SIMPLE algorithm