2D DINAMIKA FLUIDA

Projekat iz predmeta “Numerički algoritmi i numerički softver”

UVOD

-U ovom projektu bavićemo se jednostavnom implementacijom programa za rešavanje dinamike fluida u realnom vremenu, u našem slučaju u 2D prostoru. Algoritmi za rešavanje dinamike fluida svode se na korišćenje ‘Navier-Stokes Jednačina’ koje opisuju kretnju samog fluida. Jednačine same po sebi su teške za rešavanje kada nam je od velikog značaja preciznost simulacije, ali pošto ćemo se samo baviti vizuelnom reprezentacijom kretanja fluida to mnogo pojednostavljuje problem. Takođe algoritam se bazira na stabilnoj simulaciji i ne dolazi do prekoračenja i pucanja programa.Projekat realizovan u Python-u.

-Fluidi se javljaju svuda oko nas u realnom svetu: reke koje teku, vazduh u konstantnom strujanju, krv u telu, okeani…. Rad koji sam pratio u izradi ovog projekta je papir ‘Real-Time Fluid Dynamics for Games’-Jos Stam, 2003. koji se u to vreme bavio rešavanjem vizuelne reprezentacije za video igre. Revolucionarna stvar u vezi ovog papira je ta što su oni po prvi put koristili fiziku iz vremena Ojlera, Naviera i Stokesa. Navier-Stokes jednačine su precizne matematičke jednačine koje modeluju kretanje fluida. Analitičko rešavanje ovih jednačina bilo je moguće samo u vrlo jednostavnim slučajevima i zbog toga nisu korišćene do pojave računara i njihove implementacije pomoću numeričkih algoritama. Pošto za kompleksna izračunavanja (otpor vazduha kod aviona, sile koje deluju na most) treba dosta vremena i resursa mi se ovde nećemo bazirati na to nego na vizuelni prikaz i brzo (Real-Time) računanje.

FIZIKA FLUIDA

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} + \nu \nabla^2 \mathbf{u} + \mathbf{f}$$

$$\frac{\partial \rho}{\partial t} = -(\mathbf{u} \cdot \nabla)\rho + \kappa \nabla^2 \rho + S$$

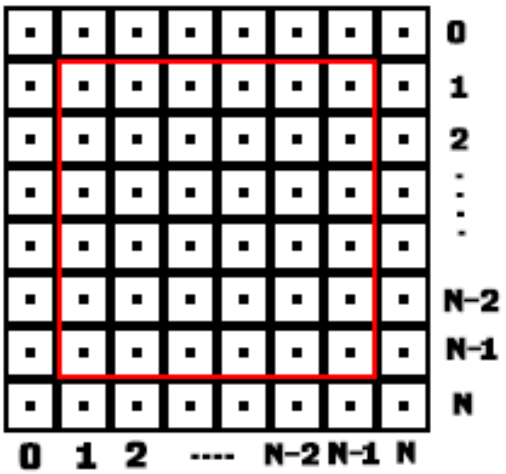
-Posmatrajući ove dve jednačine vidimo da je u gornjoj opisana promena brzine tokom male promene vremena (izvod po vremenu) dok je u drugoj jednačini opisana promena gustine po vremenu tečnosti. Kao što vidite jednačine mnogo liče jedna na drugu i obe zavise od 3 stvari koje se nalaze sa desne strane jednačine. Možemo da zamislimo neki fluid/tečnost kao vazduh u nekoj sobi ili na nekoj površini, na njega će prirodno uticati izvori toplote, kretanje vazduha koje već ima neku brzinu.

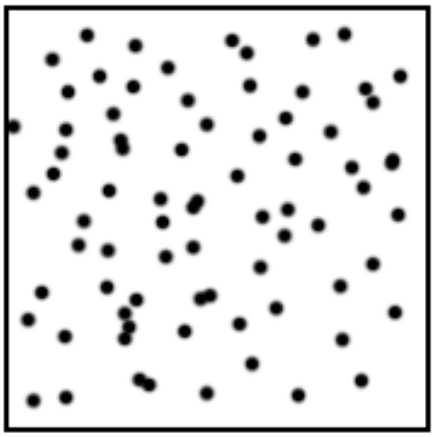
-Pošto su u realnom svetu svi fluidi (pa i čvrste materije) stišljive mi ćemo za naš fluid smatrati da se ponaša kao nestišljiv (na primer voda ili ulje). Iako su voda i ulje stišljivi, kao i dijamant koji je 500 puta manje stišljiv od vode (ali ipak jeste) mi ćemo ih smatrati nestišljivima. Modelovanjem kretanja i interakcija stišljivih fluida kao što je na primer vazduh, koji u flašu od jedne litre možete nagurati više od litre vazduha za razliku od vode koje stane tačno jedan litar, nećemo se baviti u obimu ovog projekta jer je za to potreban matematički aparat koji daleko prevazilazi obime ovog projekta i kursa i koji sa sobom takođe donosi duža i obimnija računanja te naš program više ne bi bio u realnom vremenu nego bi morali da čekamo na rezultate računanja! Iz ovoga sledi da nam je zapremina pa tako i gustina u svakom delu fluida ista!

-Brzinu koju fluid poseduje mi ćemo da predstavimo kao niz tj. u našem 2D slučaju kao matricu vektora. Gde ćemo imati posebnu matricu za X i za Y koordinate i tako zapravo razdovjiti i vrišiti računanja. Pošto smo iz prethodnog pasusa zaključili da ćemo da radimo sa nestišljivim fluidima i da će nam gustina biti ista u celom fluidu mi ćemo da uvedemo ‘DRUGU GUSTINU’ koju možemo da smatramo kao boju koju dodamo na tanjir vode. Koristeći ove dve formule mi ćemo da modelujemo kretnju vektora brzina kroz same sebe kao i tu drugu gustinu boje koja se kreće kroz vektore brzina i pomoću koje zapravo vidimo ponašanje fluida.

FLUID U KUTIJI

-Za reprezentaciju fluida kojim se bavimo u ovom projektu su moguća dva pristupa. Prvi pristup je da se fluid modeluje kao skup čestica od kojih je fluid izgrađen i on zapravo predstavlja atome i molekule u fluidu i njih modeluje i posmatra njihovo kretanje i međusobnu interakciju čestica.





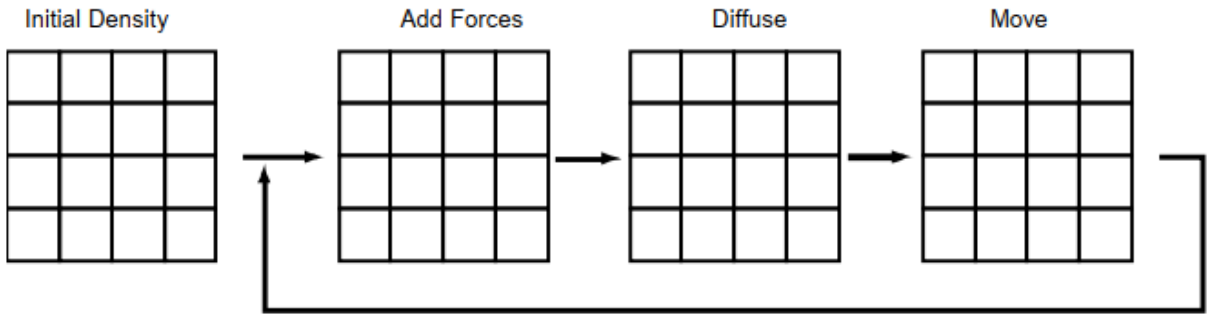
-Drugi pristup koji ćemo mi zapravo koristiti i koji je bolji po performanse programa i zbog jednostavnosti shvatanja i implementacije posmatra oblast kao neku konačnu ravan (ili kocku ako je u pitanju 3D) koju delimo na N podjednakih delova. Zbog specifičnosti problema za slučaje na granicama ravni moraćemo posebno da računamo slučajeve i zbog toga i dodajemo dva dodatna polja za X i dva za Y kooridnatu. Naš fluid će se zapravo kretati samo unutar crvene ograničene oblasti dok se u graničnim slučajevima računaju vrednosti koje ćemo kasnije objasniti. Svaku ćeliju ćemo smatrati česticom za sebe i svakoj ćemo pridodati X i Y komponente brzine kao i GUSTINU BOJE koje se smatraju da se nalaze u centru svake ćelije. U daljem tekstu smatraće se da se GUSTINA odnosi na (=) GUSTINA BOJE.

```
#MATRIXES NEEDED FOR REPRESENTATION AND CALCULATION
self.densityZERO = np.zeros(shape=(self.n,self.n))
self.density = np.zeros(shape=(self.n,self.n))

self.Vx = np.zeros(shape=(self.n,self.n))
self.Vy = np.zeros(shape=(self.n,self.n))

self.Vx0 = np.zeros(shape=(self.n,self.n))
self.Vy0 = np.zeros(shape=(self.n,self.n))
```

-Za rešavanje i računanje trebaće nam pored trenutnih i naredne vrednosti X,Y komponenti brzine kao i gustine.

PROMENA GUSTINE

-Za početak da razjasnimo kako sam ja zamislio svoj program. Tokom svake iteracije u sredini ekrana ću dodavati novu boju i u zavisnosti od pozicije miša upravljaću gde će boja biti ispaljena tj usmeravaću njeno polje brzine. Što se tiče drugih implementacije moguće je takođe napraviti da se desnim klikom dodaje boja a levim da se pomera, ili da se u igrama dim pomera kako igrač prolazi kroz njega i razne kombinacije i varijante a ja sam se odlučio za ovu. Vrednosti se menjaju tokom svake iteracije.



-Da bi objasnili sliku od gore vratimo se na prvu sliku koja prikazuje Navier-Stokes jednačine, konkretno donju za gustinu. Jednačina nalaže da promena gustine tokom vremena zavisi od 3 stvari sa desne strane jednačine.
Prvi član (‘ -(u*∇) ρ ’) nam govori da gustina prati gradijent brzine tj. brzinu.
Drugi član (’ k ∇**2 ρ ’) nam govori da gustina moze da difuzira tj. da

može da se rasipa. Ovo možemo zamisliti u primeru ako kanemo na primer neku boju na tanjir vode, voda iako nužno nema neku kretnju boja će se sama po sebi širiti tj. difuzirati/rasipati jer teži da se izjednači u celoj tečnosti. Treći član (' S ') nam govori da se gustina povećava u zavisnosti od izvora gustine. To će u mom slučaju zapravo biti svaka tokom svake iteracije u centru jer jelte tako sam zamislio i implementirao program. Rešavanje ove jednačine zapravo ćemo obaviti u obrnutom redosledu od navedenih članova u jednačini pa ćemo tako prvo dodati izvore, pa ćemo izvršiti rasipanje i na kraju ćemo da pomeramo kroz polje brzine kao što je i prikazano na gornjoj slici sled događaja!

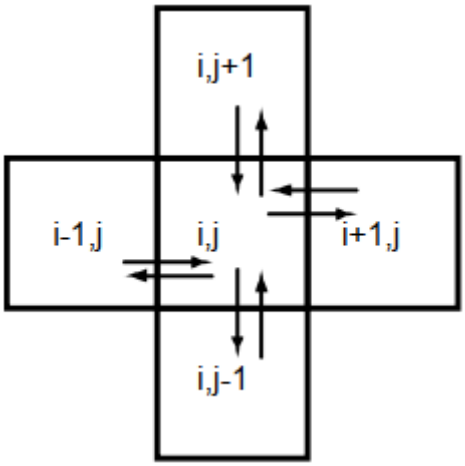
DODAVANJE GUSTINE I BRZINE

-Pošto su ove dve funkcije vrlo jednostavne i intuitivne obe metode ćemo da ostavimo ovde u dokumentaciji a posle možete i sami da pogledate u fajlovima.

```
def addDye(self,x,y,kolicina):
    self.density[x,y]+=kolicina

def addVelocity(self,x,y,kolX,kolY):
    self.Vx[x,y]+=kolX
    self.Vy[x,y]+=kolY
```

DIFUZIJA

-Drugi korak posle dodavanja vrednosti jeste rasipanje vrednosti. Ovo se radi i za gustinu i za brzinu. Vrednost koja se koristi za gustinu jeste diff dok je za brzinu visc. Kad su vrednosti vece od (' > ') 0 vrednosti će se rasipati. Pogledaćemo slučaj pojedinačne ćelije pa posle logiku primeniti na celo polje. Vrednosti za ćeliju koju gledamo će se menjati u zavisnosti od 4 najbliže ćelije toj posmatranoj. Vrednost posmatrane će se smanjivati ako su okolne vrednosti manje od njene ili povećavati ako su veće. Vrednost unutrašnje ćelije će težiti prosečnoj vrednosti susednih ćelija i na osnovu toga će menjati vrednost.



-Jedno rešenje za implementaciju jeste da se naredna vrednost posmatrane ćelije računa tako što se prosek susednih ćelija doda na vrednost trenutne ćelije.

```
x[i,j] = x0[i,j] + a*(x0[i-1,j]+x0[i+1,j]+x0[i,j-1]+x0[i,j+1]-4*x0[i,j])
```

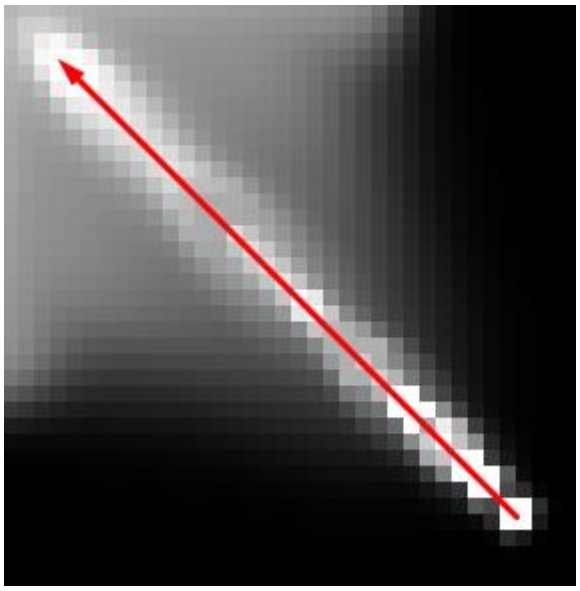
-Iako ovo rešenje deluje intuitivno za velike vrednosti množitelja ‘a’ vrednosti počinju da osciluju i postaju negativne i konačno divergiraju čime simulacija postaje beskorisna.To se dešava jer su vrednosti linearno zavisne od prethodnih. Ovo je bio problem koji je u vreme Jos Stama bio čest i zbog čega su metode tog doba bile nestabilne. Revolucionarna stvar u njegovorm radu jeste ta što su Jos i njegov tim zaključili i došli do jednostavne ideje napravivši hiperboličnu zavisnost tako što su pronalazi trenutne vrednosti koje pri rasipanju u nazad u vremenu daju početne vrednosti! Vrednosti ćemo da računamo preko Gauss-Seidel metode.

```
x[i,j] = (x0[i,j] + a*(x[i-1,j]+x[i+1,j]+x[i,j-1]+x[i,j+1]))*cRecip
```

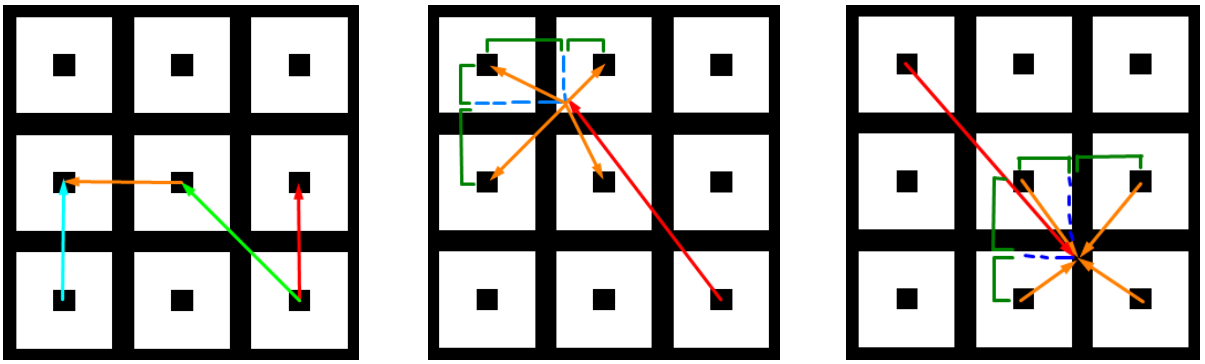
-Lepota ove metode prema Jos-u je ta što iako je jednostavna kao i prethodna simulacija će biti stabilna za bilo koje vrednosti: diff, dt i N. Gde je diff zamenjeno sa visc za viskoznost kod brzine. Dok su nam vrednosti a,c i cRecip računate kao:

```
a = dt * diff * (n-2) * (n-2)
c=1+4*a
cRecip=1.0/c
```

ADVEKCIJA



-Razmotrimo sada treći i poslednji član u našoj Navier-Stokes jednačina. Treći član se tiče pomeranja atributa jedne konkretne ćelije u zavistnosti od toga gde se u polju nalazi i kakva je tu struja tj. vektor brzina. Prva zamisao (‘ a) ’) i mnogo jednostavniji pristup je da vektor brzine pokazuje sa centra jedne ćelije direkt u centar druge. Iako je ovaj pristup jednostavan u praksi vektori brzine nikad ne pokazuju tako egzaktno i precizno nego su mnogo izmešaniji i neregularniji kao u realnim sistemima. Druga zamisao (‘ b) ’)koja nam pada na pamet jeste da bi iz sredine svake ćelije prateći vektor brzine mogli da sračunamo gde će gustina ili brzina te ćelije koju posmatramo da završi i da onda iz te tačke interpoliramo u 4 najbliže tačke i tako rasporedimo gustinu ili brzinu.



a) b) c)

-Iako metoda pod (‘ b) ‘) deluje intuitivno i jeste pravi smer razmišljanja sa ovim pristupom ćemo da naletimo na problem jer će nam u realnom svetu u

jednoj tački završiti više ćelija prateći vektor brzine te ćemo više puta računati vektore jedne ćelije što nam nije optimalno. Pravi i najefektivniji način da se reši ovaj problem jeste sledeći (‘ c) ‘). Prateći vektor brzine ali unazad mi ćemo naći tačku iz koje trenutna ćelija zapravo dobija vrednost a njena vrednost će zapravo biti interpolacija 4 njoj najbliže tačke. Tako smo efektivno za svaku ćeliju našeg polja/matrice računali vrednosti u jednom prolazu što nam je daleko optimalnije!

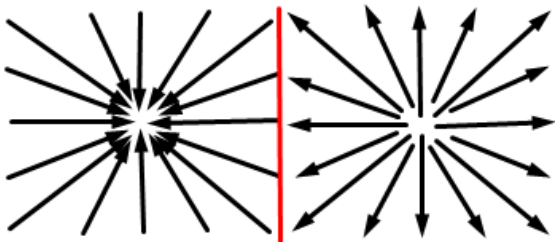
EVOLUCIJA VREDNOSTI

```
def Step(self,iter):
    difuse(1, self.Vx0, self.Vx, self.visc, self.dt, iter, self.n)      #DIFUSES VELOCITY VECTORS
    difuse(2, self.Vy0, self.Vy, self.visc, self.dt, iter, self.n)

    project(self.Vx0, self.Vy0, self.Vx, self.Vy, iter, self.n)         #PROJECTS THE VELOCITY

    advect(1, self.Vx, self.Vx0, self.Vx0, self.Vy0, self.dt, self.n)   #ADVECTS THE VELOCITY
    advect(2, self.Vy, self.Vy0, self.Vx0, self.Vy0, self.dt, self.n)

    project(self.Vx, self.Vy, self.Vx0, self.Vy0, iter, self.n)         #PROJECTS THE VELOCITY AGAIN

    difuse(0, self.densityZERO, self.density, self.diff, self.dt, iter, self.n)     #DIFUSES THE DYE
    advect(0, self.density, self.densityZERO, self.Vx, self.Vy, self.dt, self.n)    #ADVECTS THE DYE
```

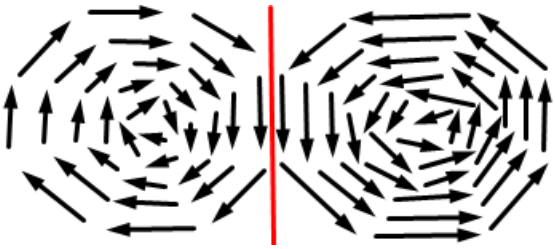
-Sa ova tri koraka mi smo završili osnovne koncepte za dodavanje, difuziju i pomeranje vrednosti, gde prvo radimo pre metode ‘Step’ dodavanje vrednosti pomoću metoda ‘addDye’ i ‘addVelocity’. Posle pozivamo metodu ‘Step’ koja vrši prvo difuziju za brzinu pa pomeranje brzine pa potom raspršuje gustinu i onda je pomera. Kao što vidite u sklopu metode ‘Step’ imamo I fju ‘Project’, nju ćemo naredno da objasnimo.

PROJEKCIJA

-Projekcija jeste bitan i najkomplikovaniji deo programa za simulaciju fluida. Ova funkcija nam omogućava da brzina tj. sistem u celosti očuvava masu. Ovo je veoma bitna osobina realnih fluida koje bi se trebali pridržavati. Ova funkcija nam mnogo čemu doprinos jer će pored očuvanja mase takođe terati fluid da ima mnogo vrtloga i tokova koji će biti kružni.



DIVERGENCIJA



VRTLOG

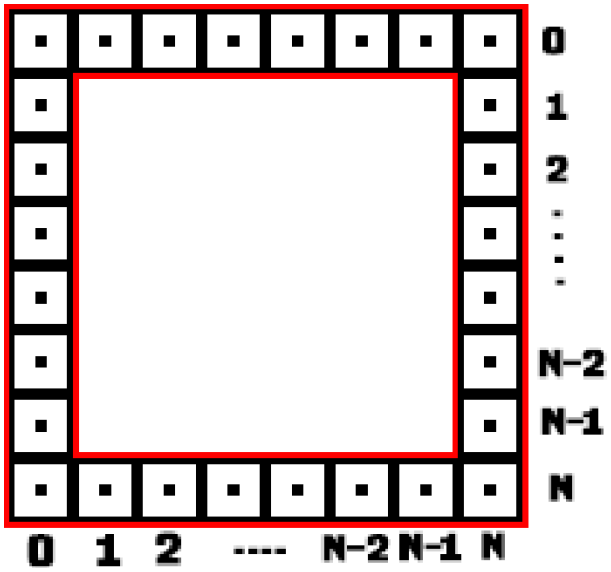
-Da bi smo shvatili kako projekcija funkcioniše prvo moramo da shvatimo osnovne pojmove Divergencije i Vrtloga. Vrtlog je kao što znate kada se vektori kreću kružno, dok pojam Divergencije odlikuje to što vektori upiru jedan u drugog ili jedan od drugog. Ove dve osobine su svojstvene za svaki vektorsko polje. Koje god vektorsko polje da imamo mi možemo da nađemo njegove vrtloge i njegovu divergenciju.

- Razmislimo sada o tečnostima (fluidima). Tečnosti imaju mnogo vrtloga ali za razliku od njih nemaju divergenciju jer da ona postoji to bi moralo da znači da nam se materijal stvara ni iz čega i da materijal isto tako nestaje, što nije moguće! Zbog toga nam je cilj da nam vektorsko polje brzina naše tečnosti ima vrtloge ali da nema divergenciju. Ali posle računanja difuzije i advekcije naše tečnosti dobijamo polje brzina koje poseduje i vrtloge i divergenciju. Mi zapravo želimo da dobijemo polje bez divergencije, ovo se radi pomoću 'Helmholtzove Dekompozicije'.

-Naše prvobitno originalno izračunato polje posle difuzije i advekcije po Helmholtzu je zapravo suma dva polja. Prvo koje je bez divergencije i sadrži samo vrtloge i drugog koje je bez vrtloga i sadrži samo divergenciju (suštinski gradijentno polje). Pošto ne postoji način da se direktno izračuna polje bez divergencije mi ćemo zapravo da izračunamo polje bez vrtloga i oduzmemo ga od našeg početnog polja faktički dobijajući polje bez divergencija tj. samo

vrtloge što nam je bio i cilj! Ovde ćemo takođe ponovo koristiti Gauss-Seidel metodu.

GRANIČNI SLUČAJEVI



-Na kraju dolazimo do komentarisanja graničnih slučajeva. Na početku prezentacije/dokumentacije smo ustanovili da ćemo fluid posmatrati na nekoj ograničenoj oblasti, samim tim će nam 0 i N biti granice našeg polja/oblasti. Najjednostavnija implementacija ovoga jeste ta da nam horizontalna komponenta na vertikalnim zidovima treba biti 0, a da nam vertikalna komponenta na horizontalnim zidovima takođe treba biti 0. Samim tim će nam vrednosti brzina u toj graničnoj oblasti varirati u zavisnosti od ćelija odma do nje koje se smatraju unutrašnjima. Moguće su i druge implementacije granica kao na primer: da fluid koji izađe na levu stranu da se pojavi na desnoj ili da onaj koji izađe gore se pojavljuje dole i obrnuto, da napravimo da nam sa jedne strane ulazi tečnost a na drugu izlazi i napravimo neki način ‘Wind Tunnel-a’. Ovo ostavljamo da čitaocu sam po potrebi implementira.

STUDENT

Luka Ličina, 08.02.2022.

‘Fakultet Tehničkih Nauka’ , Novi Sad

Predmet: Numerički Algoritmi i Numerički Softver

REFERENCE

1. Real-Time Fluid Dynamics for Games, Jos Stam, 2003. https://www.autodesk.com/research/publications/real-time-fluid-dynamics
2. Fluid Simulation for Dummies, Mike Ash, 2006. https://mikeash.com/pyblog/fluid-simulation-for-dummies.html
3. GitHub Thread- Lattice Boltzmann Methods for Fluid Simulation, Daniel Shiffman, 2019. https://github.com/CodingTrain/Rainbow-Topics/issues/178
4. Lattice-Boltzmann Fluid Dynamics, Weber State University, 2012. https://physics.weber.edu/schroeder/javacourse/LatticeBoltzmann.pdf
5. But How DO Fluid Simulations Work, Gonkee Youtube, 2020. https://www.youtube.com/watch?v=qsYE1wMEMPA
6. Divergence and curl: The language of Maxwell’s equations, fluid flow and more, 3Blue1Brown Youtube, 2018. https://www.youtube.com/watch?v=rB83DpBJQsE&t=661s