SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1451

SIMULACIJA I VIZUALIZACIJA FLUIDA

Marin Lovrinović

Zagreb, lipanj 2026.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1451

SIMULACIJA I VIZUALIZACIJA FLUIDA

Marin Lovrinović

Zagreb, lipanj 2026.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 2. ožujka 2026.

DIPLOMSKI ZADATAK br. 1451

Pristupnik:	**Marin Lovrinović (0036540934)**
Studij:	Računarstvo
Profil:	Programsko inženjerstvo i informacijski sustavi
Mentorica:	prof. dr. sc. Željka Mihajlović

Zadatak: **Simulacija i vizualizacija fluida**

Opis zadatka:

Simulacija fluida predstavlja jedan od najzahtjevnijih izazova u modernoj računalnoj grafici zbog kompleksnosti fizikalnog ponašanja. Cilj ovog rada je proučiti teorijsku fizikalnu podlogu, posebice Navier?Stokesove jednadžbe koje predstavljaju osnovu dinamike fizikalnog ponašanja stlačivih i nestlačivih fluida. Napraviti pregled raznih koncepata koji se koriste pri implementaciji ovog fizikalnog modela, odnosno metode temeljene na česticama (Lagrangeov pristup), metode temeljene na rešetci (Eulerov pristup), te hibridne metode. Proučiti i potrebne matematičke metode za rješavanje parcijalnih diferencijalnih jednadžbi kakve se pojavljuju u ovom kontekstu. Ostvariti programsku implementaciju odabranog modela te pripadnu vizualizaciju. Analizirati utjecaj raznih parametara na kvalitetu ostvarenih rezultata te dati kritički osvrt na dobivene performanse. Razmotriti mogućnost korištenja GPU za navedenu problematiku. Diskutirati upotrebljivost rješenja i dati smjernice za buduća poboljšanja.

Rok za predaju rada: 26. lipnja 2026.

Sažetak

Simulacija i vizualizacija fluida

Obrađena je teorija fizikalnog modeliranja fluida, s naglaskom na Navier-Stokes jednadžbe. Pokrivene su čestične (Lagrangeove) metode simuliranja fluida i one temeljene na rešetci (Eulerove), kao i hibridne metode. Navedene su sile koje se mogu uračunati tijekom simulacije fluida metodom SPH, kao i njihove formule. Obrađene su neke metode vizualizacije fluida. Istražena je implementacija metode SPH koristeći Verlet integraciju uz dodatak bliske gustoće i optimizaciju podjelom prostora na ćelije. Objašnjene su funkcionalnosti najbitnijih metoda. Razmotreno je ponašanje simuliranog fluida u ovisnosti o raznim parametrima simulacije. Performanse dvodimenzionalne i trodimenzionalne simulacije su sagledane. Razmotrena su potencijalna poboljšanja simulacije.

Ključne riječi: simulacija, vizualizacija, simulacija fluida, računalna grafika, računalno modeliranje, C++, OpenGL

Summary

Simulation and Visualization of Fluids

The theory of physical modeling of fluids was presented, with an emphasis on the Navier–Stokes equations. Particle-based (Lagrangian) methods for fluid simulation and grid-based (Eulerian) methods, as well as hybrid approaches, were covered. The forces that can be taken into account in fluid simulation using the SPH method, along with their formulas, were listed. Several methods for fluid visualization were also discussed. An implementation of the SPH method using Verlet integration was examined, with the addition of density estimation and optimization through spatial partitioning. The functionality of the most important methods was explained. The behavior of the simulated fluid under various simulation parameters is analyzed. The performance of both two-dimensional and three-dimensional simulations is evaluated. Potential improvements to the simulation are also considered.

Keywords: simulation, visualization, fluid simulation, computer graphics, computer modelling, C++, OpenGL

Sadržaj

Uvod

Simulacija fluida je područje široke primjene, od interaktivnih simulacija u stvarnom vremenu do simulacija koje nisu u stvarnom vremenu i koriste se za inženjerstvo i znanstvene svrhe. U ovom radu će biti objašnjene teorijske osnove fizikalnog modeliranja fluida, uključujući Navier-Stokes jednadžbe. Biti će obrađene glavne metode simulacije fluida; čestične, one temeljene na rešetci, kao i hibridne. Glavni pristupi vizualizaciji fluida u virtualnom okruženju bit će sagledani. Nakon toga će biti objašnjena implementacija čestične metode SPH (engl. smoothed particle hydrodynamics) u C++-u i OpenGL-u, s naglaskom na neke dodatke i optimizacije koje su potrebne za implementaciju takve simulacije u stvarnom vremenu na zadovoljavajućoj razini detalja. Nakon toga će rezultati biti analizirani i daljnje mogućnosti razvoja projekta bit će razmotrene.

1. Teorija

U sljedećem poglavlju bit će objašnjene Navier-Stokes jednadžbe, koje su osnova modeliranja fluida u fizici. Nakon toga će biti istražena teorijska osnova glavnih metoda simulacije fluida, koje se dijele na čestične (Lagrangeov pristup), one temeljene na rešetci (Eulerov pristup) i hibridne. Bit će sagledane mogućnosti vizualizacije rezultata dobivenih tim metodama simulacije.

1.1. Navier-Stokesove jednadžbe

Navier-Stokesove jednadžbe su osnova fizičkog modeliranja fluida [1]. Jednadžba (1) govori o očuvanju mase, gdje je ρ gustoća fluida, t vrijeme, a v brzina fluida.

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho v) = 0 \tag{1}$$

Intuitivno, ta jednadžba nam govori da kada fluid odlazi s neke točke, gustoća fluida u toj točki se smanjuje, i obratno. Alternativno, za nestlačive fluide jednadžba se pojednostavljuje. Kod nestlačivih fluida gustoća svugdje mora biti jednaka, što znači da tok fluida u neku točku mora biti jednak toku fluida iz te točke. Upravo to nam govori jednadžba (2). Divergencija brzine fluida mora biti jednaka nuli.

$$\nabla \cdot v = 0 \tag{2}$$

Jednadžba (3) je kompleksnija i govori o očuvanju količine gibanja.

$$\rho\left(\frac{\partial v}{\partial t} + v \cdot \nabla v\right) = -\nabla p + \rho g + \mu \nabla^2 v \tag{3}$$

Ova jednadžba analogna je Newtonovom drugom zakonu ($ma = F$). Gustoća ρ analogna je masi, a izraz $\left(\frac{\partial v}{\partial t} + v \cdot \nabla v\right)$ analogan je akceleraciji. Sastoji se od promjene polja brzine u vremenu $\frac{\partial v}{\partial t}$ i promjene u polju brzine u smjeru kretanja, $v \cdot \nabla v$. S desne strane nalaze se izrazi za unutarnje sile i vanjsku silu. Prva unutarnja sila je sila pritiska. Izraz $-\nabla p$ označava da fluid ubrzava iz područja višeg pritiska u područje nižeg pritiska. Druga unutarnja sila je sila viskoznosti. Izraz $\mu \nabla^2 v$ označava da se točka u fluidu teži kretati jednako kao okolni fluid, ovisno o konstanti viskoznosti μ. Izraz ρg označava vanjsku silu poput gravitacije.

Sada će biti obrađeni neki pristupi numeričkom rješavanju tih jednadžbi i simuliranju fluida na računalu.

1.2. Lagrangeove metode

Lagrangeove metode ili čestične metode jedan su od glavnih pristupa simulaciji fluida na računalu. Te metode predstavljaju fluid kao skup čestica koje međusobno utječu jedna na drugu. Svaka čestica ima svojstva poput mase, brzine i gustoće. Korištenje čestica značajno pojednostavljuje Navier-Stokesove jednadžbe. Jednadžbu za očuvanje mase (1) možemo u potpunosti izostaviti s obzirom da svaka čestica ima konstantnu masu, a izraz $\left(\frac{\partial v}{\partial t} + v \cdot \nabla v\right)$ u jednadžbi za očuvanje količine gibanja (3) možemo zamijeniti s promjenom brzine čestice u vremenskom koraku. Samim time, da bismo dobili promjenu brzine čestice, prema izrazu (3) moramo izračunate sile na česticu podijeliti s izračunatom gustoćom čestice [1]. Najčešće korištena čestična metoda za simulaciju fluida je metoda zaglađenih čestica (engl. smoothed particle hydrodynamics, SPH). Metoda SPH je prvi put predstavljena u radovima [2] i [3] u području astrofizike, a kasnije je prilagođena za računalnu grafiku, na primjer [1].

1.2.1. Osnovni principi

Metoda zaglađenih čestica nam omogućuje da veličinu bilo kojeg polja koje je definirano samo na poznatim lokacijama čestica interpoliramo na bilo koju drugu točku u prostoru. Za tu svrhu se koriste radijalno simetrične jezgre zaglađivanja koje raspoređuju neku veličinu u susjedstvo čestice [1]. Općenito, skalarnu veličinu A možemo interpolirati na lokaciji r koristeći težinsku sumu doprinosa od svih čestica (4).

$$A_s(r) = \sum_j A_j \frac{m_j}{\rho_j} W(r - r_j, h) \tag{4}$$

U ovoj formuli je m_j masa čestice, r_j njena pozicija, ρ_j njena gustoća i A_j je veličina skalarnog polja u točki r_j. Funkcija W(r, h) je jezgra zaglađivanja s radijusom zaglađivanja h. Primijetimo da doprinos čestice množimo s $\frac{m_j}{\rho_j}$, što je izraz za volumen koji ta čestica predstavlja. Masa čestice je konstantna, a gustoća svake čestice morat će biti izračunata prije nego se može primijeniti formula (4) za interpolaciju drugih veličina [1].

1.2.2. Izračun gustoće

Formula za gustoću može se dobiti supstitucijom u formulu (4), kako slijedi (5).

$$\rho_s(r) = \sum_j \rho_j \frac{m_j}{\rho_j} W(r - r_j, h) = \sum_j m_j W(r - r_j, h) \tag{5}$$

1.2.3. Sila pritiska

Koristeći jednadžbu (4) može se dobiti izraz za silu pritiska:

$$f_i^{pressure} = -\nabla p(r_i) = -\sum_j p_j \frac{m_j}{\rho_j} \nabla W(r_i - r_j, h) \tag{6}$$

Ovdje će se morati izračunati gradijent jezgre zaglađivanja. Nažalost, ova sila nije simetrična, što bi kršilo Newtonov treći zakon i time zakon očuvanja energije. Jednostavan način za dobiti simetričnu silu pritiska je korištenje aritmetičke sredine pritiska trenutne čestice i susjedne čestice (7) [1]

$$f_i^{pressure} = -\sum_j \frac{p_i + p_j}{2} \frac{m_j}{\rho_j} \nabla W(r_i - r_j, h) \tag{7}$$

Pritisak na nekoj čestici može se dobiti korištenjem izraza (8), gdje je k konstanta plina koja ovisi o temperaturi, ρ gustoća fluida na čestici, a ρ_0 ciljana gustoća fluida.

$$p = k(\rho - \rho_0) \tag{8}$$

1.2.4. Sila viskoznosti

Uvrštavanjem u jednadžbu (4) može se dobiti izraz za silu viskoznosti (9).

$$f_i^{viscosity} = \mu \nabla^2 v(r_i) = \mu \sum_j v_j \frac{m_j}{\rho_j} \nabla^2 W(r_i - r_j, h) \tag{9}$$

Ova sila nije simetrična. S obzirom da sile viskoznosti ovise samo o razlikama brzine, a ne o apsolutnim brzinama, postoji jednostavan način za dobiti simetričnu silu koristeći razlike brzina (10) [1].

$$f_i^{viscosity} = \mu \sum_j (v_j - v_i) \frac{m_j}{\rho_j} \nabla^2 W(r_i - r_j, h) \tag{10}$$

1.2.5. Vanjske sile

Vanjske sile poput gravitacije ili interakcije s objektima mogu se primjenjivati direktno na čestice.

1.2.6. Sudari

Sudaranje s objektima može se postići izbacivanjem čestice iz objekta i reflektiranjem vektora brzine čestice od normale objekta.

1.2.7. Integracija

Integracija pozicije čestice može se izračunati raznim metodama. Eulerova metoda koristi formulu (11) za integraciju pozicije, a formulu (12) za integraciju brzine.

$$x(t + \Delta t) = x(t) + v(t) * \Delta t \tag{11}$$

$$v(t + \Delta t) = v(t) + a(t) * \Delta t \tag{12}$$

Kinematička jednadžba koristi formulu (13) za integraciju pozicije, a za integraciju brzine također koristi formulu (12).

$$x(t + \Delta t) = x(t) + v(t) * \Delta t + \frac{a(t) * \Delta t^2}{2} \tag{13}$$

Verlet integracija koristi formulu (14) za integraciju pozicije.

$$x(t + \Delta t) = 2x(t) - x(t - \Delta t) + a(t) * \Delta t^2 \tag{14}$$

Za ovu metodu nije potrebno pamtiti brzinu čestice, već samo prošlu poziciju čestice.

1.3. Eulerove metode

Eulerove metode još jedan su od glavnih pristupa simulaciji fluida na računalu. Zasnivaju se na korištenju rešetke za spremanje svojstava fluida. Čest pristup je podjela prostora na ćelije, spremanje vrijednosti pritiska za svaku ćeliju i za spremanje vrijednosti brzine fluida za sve granice između ćelija [4]. Potrebno je spremati samo komponentu brzine fluida koja je okomita na granicu između ćelija. Da bi se te brzine mijenjale prema izrazu (3), potrebno je izračunati gradijent pritiska, ∇p. Taj gradijent može se dobiti korištenjem metode konačnih razlika (eng. finite difference) s vrijednostima pritiska u susjednim ćelijama. Za nestlačive

fluide također je potrebno osigurati da je količina fluida koja ulazi u ćeliju jednaka količini koja izlazi. Kombiniranjem tih dvaju uvjeta mogu se dobiti jednadžbe za izračunavanje pritisaka i brzina u sljedećem vremenskom koraku. Bitan fenomen koji je potrebno razmotriti u simulaciji Eulerovim metodama je advekcija, odnosno kretanje fluida u smjeru polja brzine. Advekcija je trivijalna u Lagrangeovim metodama s obzirom da brzina samih čestica predstavlja brzinu fluida i fluid se stoga kreće zajedno s česticama u smjeru brzine. U Eulerovim metodama, s druge strane, potrebno je prenositi vrijednost polja brzine na sljedeće ćelije u smjeru brzine. Jedan od pristupa razmatra zamišljene čestice fluida i interpolira polja brzine ovisno o razmotrenim brzinama tih čestica. Taj pristup se zove polu-Lagrangeov (eng. semi-Lagrangian) [5]. Uključivanje Eulerovih metoda u simulaciju fluida može dovesti do još više razine preciznosti i omogućiti simuliranje šireg broja fenomena.

1.4. Hibridne metode

Hibridne metode kombiniraju Lagrangeove metode (čestične) i Eulerove metode (temeljene na rešetci) da bi se osigurala zadovoljavajuća simulacija najšireg broja scenarija.

1.4.1. FLIP metoda

Vrlo popularna hibridna metoda je FLIP, koja je varijanta metoda čestica u ćeliji (engl. particle-in-cell, PIC) [6]. PIC metode simuliraju čestice, zatim prenose vrijednosti brzina na rubove ćelija, nakon toga izračunavaju ograničenje nestlačivosti Eulerovim pristupom i potom prenose novu brzinu nazad na čestice. U ovim metodama su čestice odgovorne za advekciju kao u Lagrangeovim metodama, tako da taj dodatni korak nije potreban kao što je potreban u čistim Eulerovim metodama. Nažalost, prilikom prijenosa brzine s rešetke nazad na čestice, gubimo određenu razinu detalja s obzirom da su čestice unutar ćelije prije prijenosa brzine na rešetku mogle imati međusobno vrlo različite vrijednosti brzine. Drugim riječima, metoda sama po sebi uvodi visoku razinu viskoznosti. Ključna inovacija FLIP metode je ta da ne izračunava brzine čestica izravno iz vrijednosti na rešetci, nego samo mijenja brzine čestica ovisno o izračunatim promjenama brzina na rešetci [6]. Hibridne metode poput FLIP metode donose prednosti obiju pristupa (Eulerovog i Lagrangeovog), ali zato povećavaju kompleksnost implementacije.

1.5. Vizualizacija fluida

Vizualizacija fluida bavi se prikazivanjem željenih fluida iz podataka o simulaciji. U ovom poglavlju bit će sagledane neke glavne metode.

1.5.1. Algoritam pokretnih kocki

Algoritam pokretnih kocki jedan je od najčešće korištenih algoritama za stvaranje trodimenzionalne mreže za prikazivanje fluida i drugih prostornih podataka. Zasniva se na podjeli prostora na kocke, identificiranje graničnih kocaka i, ovisno o tome koji kutovi kocke su unutar volumena a koji izvan, generiranje odgovarajućeg dijela mreže unutar te kocke [7]. Algoritam može dati vrlo dobre rezultate, ali je izračunavanje može biti skupo.

1.5.2. Postupak praćenja puta

Prostorni podatci dobiveni iz simulacije fluida mogu se izravno prikazivati koristeći postupak praćenja puta [8]. Ta metoda može olakšati implementaciju raznih kompleksnih efekata poput refleksije i refrakcije, ali također može biti vrlo računalno zahtjevna.

1.5.3. Metode u prostoru ekrana

Metode za renderiranje fluida u prostoru ekrana (engl. screen-space methods) mogu pružiti zadovoljavajuće rezultate kada je brzo izvođenje ključno (na primjer u videoigrama). Ove metode zasnivaju se na aproksimaciji svojstava fluida poput normala površine i dubine vode uz pomoć nekih alata kao što su tekstura dubine kamere, filteri zamućivanja i aditivno renderiranje [9]. Uz pomoć tih dobivenih svojstava mogu se napraviti dovoljno dobri izračuni fenomena poput refleksije i refrakcije, uz dobre performanse.

1.5.4. Prskanje, pjena i balončići

Dodatak finih detalja poput prskanja, pjene i balončića može značajno doprinijeti vizualnom rezultatu renderiranog fluida, pogotovo ako se vizualizira nešto poput površine mora. Jedan mogući pristup tom problemu je generiranje novih čestica na mjestima velikih razlika u brzini kretanja fluida i velike kinetičke energije fluida [10]. Ovisno o gustoći okolnog fluida, te novostvorene čestice mogu spadati u 3 kategorije:

- Prskanje (niska gustoća fluida): Čestice lete kroz zrak i padaju.

- Balončići (visoka gustoća fluida): Čestice su ispod površine fluida, kretanje fluida ih ubrzava i dižu se prema površini zbog sile uzgona.
- Pjena (srednja gustoća fluida): Čestice su na površini fluida i kreću se zajedno s fluidom.

Kombinacija tih triju vrsta čestica može doprinijeti ostvarivanju realističnih vizualnih rezultata u simulaciji nekih vrsta fluida.

2. Implementacija

U ovom poglavlju bit će analizirana implementacija simulacije i vizualizacije fluida u C++-u i OpenGL-u. Odabran je čestični model zaglađenih čestica (SPH). Omogućeno je micanje kamere i interakcija s fluidom s tipkovnicom i mišem za vrijeme izvršavanja simulacije. Implementirana je vrlo jednostavna vizualizacija; svaka čestica u simulaciji prikazana je ikosaedrom u grafičkom pogonu. Prvo će biti objašnjene neke metode optimizacije i poboljšanja modela koje su korištene za postizanje zadovoljavajuće simulacije u stvarnom vremenu, a nakon toga će biti sagledani najbitniji dijelovi koda.

2.1. Podjela prostora na ćelije

Glavna prepreka pri postizanju dobrih performansi s velikim brojem čestica krije se u tome što je potrebno iterirati kroz sve susjedne čestice prilikom računanja gustoće, sile pritiska i sile viskoznosti za jednu česticu. Složenost je $O(n^2)$, što znači da vrijeme izvršavanja drastično raste s porastom broja čestica. Ključni uvid koji je potreban za rješavanje ovog problema je taj da su jedine relevantne čestice prilikom računanja gustoće, sile pritiska i sile viskoznosti one koje su unutar radijusa zaglađivanja od ciljane čestice. Iz toga slijedi, ako je prostor podijeljen na kvadratne ćelije čije su stranice jednake radijusu zaglađivanja, bit će potrebno gledati jedino čestice unutar ćelije u kojoj se ciljana čestica nalazi i 26 ćelija koje su susjedne toj ćeliji. Svaka ćelija mogla bi sadržavati listu čestica koje su u njoj, ali taj pristup doveo bi do puno dinamičkih alokacija memorije, što nije optimalno za performanse. Zato je korišten alternativni pristup [11]. Svakoj ćeliji pridružen je jedan ključ (hash od koordinata ćelije). Korištena je lista čiji elementi sadrže ključ ćelije i indeks čestice koja se nalazi u toj ćeliji. Ta lista se zove lista za prostorno pretraživanje (engl. spatial lookup). Ta lista će biti sortirana po ključevima ćelija. Time će indeksi svih čestica koje se nalaze u istoj ćeliji biti neprekinuto spremljeni jedan pored drugog u toj listi. Drugim riječima, lista će biti grupirana po ćelijama. Nakon toga će biti izrađena mapa koja za svaki ključ ćelije može dobiti indeks početka te ćelije u listi za prostorno pretraživanje. Ta mapa se zove mapa početnih indeksa (engl. starting indices). Može se dobiti jednostavnim prolaskom kroz listu za prostorno pretraživanje i bilježenjem svake promjene ključa ćelije među susjednim elementima. Jednom kada je takva struktura izrađena, lako je pristupiti svim česticama u nekoj ćeliji tako što se izračuna ključ ćelije, pogleda gdje je početak te ćelije u mapi početnih

indeksa i iterira po listi za prostorno pretraživanje dok god je ključ ćelije onaj s kojim je iteriranje započeto. Ta mogućnost može se koristiti za pronalazak svih susjednih čestica nekoj čestici tako što se iz pozicije čestice izračuna njena ćelije, iz toga se izračuna još 26 susjednih relevantnih ćelija i onda se iterira kroz sve čestice u tim relevantnim ćelijama.

2.2. Dodatak bliske gustoće

U velikom broju situacija čestice mogu zadovoljiti ograničenje ciljane gustoće tako da se odvoje od ostalih čestica i podosta približe jednoj ili dvjema drugim česticama. Zbog tog fenomena tekućina ponekad izgleda previše „mrvičasto“. Skupina čestica u letu raspada se na puno malih grumena umjesto da ostanu zajedno. Radi sprječavanja tog fenomena i povećanja kohezije tekućine, moguće je uvesti blisku gustoću (engl. near-density) [12]. To je dodatna vrijednost gustoće koja se računa prilikom simulacije, ali koja koristi oštriju (bližu) jezgru zaglađivanja. Sila pritiska koju ta gustoća stvara treba spriječiti vezanje čestica u male skupine, tako da je ta sila isključivo odbojna, neovisno o udaljenosti čestica. Ovaj dodatak je povećao koheziju tekućine i uveo nekakvu primitivnu verziju površinske napetosti. Doveo je do boljih vizualnih rezultata, osobito kod tekućine koja pada ili se brzo kreće.

2.3. Paralelizacija

Većina simulacije može se trivijalno paralelizirati jer je se uglavnom sastoji od akumuliranja vrijednosti gustoća i sila nezavisno za svaku česticu. Većina `for` petlji zamijenjeno je sa `std::foreach` pozivima uz politiku izvršavanja (eng. execution policy) `std::execution::par`. To je dovelo do ubrzanja simulacije od otprilike 4 puta.

2.4. Razred `Fluid`

Klasa Fluid sadrži stanje svih čestica i logiku za simulaciju fluida:

```
class Fluid
{
private:
    struct SpatialLookupEntry
    {
        int particleIndex;
```

```
        unsigned int cellKey;
        bool operator<(const SpatialLookupEntry& other) const
        {
            return cellKey < other.cellKey;
        }
    };

    int particleCount;
    float smoothingRadius;
    float targetDensity;
    float pressureMultiplier;
    float nearPressureMultiplier;
    float viscosityStrength;
    vector<glm::vec3> previousPositions;
    vector<glm::vec3> currentPositions;
    vector<glm::vec3> predictedPositions;
    vector<int> particleIndices;
    vector<glm::vec3> velocities;
    vector<glm::vec3> forces;
    vector<glm::vec2> densities;
    vector<SpatialLookupEntry> spatialLookup;
    vector<int> startIndices;
    Object* object;
    glm::vec3 corner1, corner2;

    [[nodiscard]] float SmoothingKernelFlat(float radius,
float distance) const;
    [[nodiscard]] float SmoothingKernelSpikyPow2(float
radius, float distance) const;
    [[nodiscard]] float
SmoothingKernelSpikyPow2Derivative(float radius, float
distance) const;
    [[nodiscard]] float SmoothingKernelSpikyPow3(float
radius, float distance) const;
    [[nodiscard]] float
SmoothingKernelSpikyPow3Derivative(float radius, float
distance) const;

    float kernelScalingFactorFlat;
    float kernelScalingFactorSpikyPow2;
    float kernelScalingFactorSpikyPow2Derivative;
```

```
    float kernelScalingFactorSpikyPow3;
    float kernelScalingFactorSpikyPow3Derivative;
    float pi = glm::pi<float>();

    [[nodiscard]] glm::vec2 CalculateDensities(glm::vec3
samplePoint) const;
    [[nodiscard]] float DensityToPressure(float density)
const;
    [[nodiscard]] float NearDensityToNearPressure(float
nearDensity) const;
    [[nodiscard]] glm::vec3 CalculatePressureForce(int
particleIndex) const;
    [[nodiscard]] glm::vec3 CalculateViscosityForce(int
particleIndex) const;
    [[nodiscard]] glm::vec3
CalculateInteractionForce(glm::vec3 inputPos, float radius,
float strength, int particleIndex) const;
    static glm::ivec3 PositionToCellCoord(glm::vec3 point,
float radius);
    static unsigned int Fluid::HashCell(glm::ivec3 cell);
    [[nodiscard]] unsigned int GetKeyFromHash(unsigned int
hash) const;
    void UpdateSpatialLookup();
    template<typename F>
    void ForeachPointWithinRadius(glm::vec3 samplePoint, F&&
f) const;
public:
    Fluid(
        glm::ivec3 particleGridDimensions,
        float smoothingRadius,
        float targetDensity,
        float pressureMultiplier,
        float nearPressureMultiplier,
        float viscosityStrength,
        glm::vec3 startingPosition,
        float particleSize,
        float particleSpacing,
        glm::vec3 corner1,
        glm::vec3 corner2,
        Object* object);
```

```
    void Update(float dt, glm::vec3 gravity,
optional<glm::vec4> interaction);
};
```

Kod 2.4 Razred *Fluid*

Korisnik razreda može specificirati broj čestica, radijus zaglađivanja, ciljanu gustoću fluida, konstantu sile pritiska, konstantu bliske sile pritiska, konstantu viskoznosti, inicijalnu raspodjelu čestica, veličinu čestica i granice prostora simulacije. Spremnik `currentPositions` sadrži pozicije svih čestica, dok spremnik `previousPositions` sadrži prethodne pozicije koje su nam potrebne za Verlet integraciju. Uz to je korišten dodatni spremnik `predictedPositions`. Naime, na početku koraka simulacije predviđene su pozicije čestica na osnovu dosadašnjeg kretanja i sile su izračunate na osnovu tih predviđenih pozicija [13] umjesto na pozicijama u spremniku `currentPositions`. To je značajno poboljšalo stabilnost simulacije. Unatoč tome što je korištena Verlet integracija, potreban je spremnik `velocities` koji sadrži brzine čestica zato da bi se mogla izračunati sila viskoznosti. U spremnik `forces` akumulira se vrijednost sile na svaku česticu tijekom koraka simulacije. U spremnik `densities` sprema se izračun gustoće i bliske gustoće na poziciji svake čestice. Spremnici `spatialLookup` i `startIndices` služe za implementaciju podjele prostora na ćelije koja je objašnjena u poglavlju 2.1. Član `object` služi za vizualizaciju fluida u grafičkom pogonu.

2.4.1. Inicijalizacija

Ovako se inicijaliziraju podatci o fluidu:

```
Fluid::Fluid(
    const glm::ivec3 particleGridDimensions,
    const float smoothingRadius,
    const float targetDensity,
    const float pressureMultiplier,
    const float nearPressureMultiplier,
    const float viscosityStrength,
    const glm::vec3 startingPosition,
    const float particleSize,
    const float particleSpacing,
    const glm::vec3 corner1,
    const glm::vec3 corner2,
    Object* object) :
```

```
particleCount(particleGridDimensions.x *
particleGridDimensions.y * particleGridDimensions.z),
smoothingRadius(smoothingRadius),
targetDensity(targetDensity),
pressureMultiplier(pressureMultiplier),
nearPressureMultiplier(nearPressureMultiplier),
viscosityStrength(viscosityStrength),
object(object),
corner1(corner1),
corner2(corner2)
{
    previousPositions.resize(particleCount);
    currentPositions.resize(particleCount);
    predictedPositions.resize(particleCount);
    particleIndices.resize(particleCount);
    velocities.resize(particleCount);
    forces.resize(particleCount);
    densities.resize(particleCount);
    spatialLookup.resize(particleCount);
    startIndices.resize(particleCount);
    object->transforms.resize(particleCount);

    for (int x = 0; x < particleGridDimensions.x; ++x)
    {
        for (int y = 0; y < particleGridDimensions.y; ++y)
        {
            for (int z = 0; z < particleGridDimensions.z;
++z)
            {
                const int index = z *
particleGridDimensions.x * particleGridDimensions.y + y *
particleGridDimensions.x + x;
                const glm::vec3 particlePosition =
startingPosition + glm::vec3(particleSpacing * x,
particleSpacing * y, particleSpacing * z);
                previousPositions[index] = particlePosition;
                currentPositions[index] = particlePosition;
            }
        }
    }
```

```
    for (int i = 0; i < particleCount; i++)
    {
        object-
>transforms[i].SetPosition(currentPositions[i]);
        object-
>transforms[i].SetScale(glm::vec3(particleSize));
        particleIndices[i] = i;
    }

    // pre-calculate kernel scaling factors
    kernelScalingFactorFlat = 315 / (64 * pi *
glm::pow(smoothingRadius, 9));
    kernelScalingFactorSpikyPow3 = 15 / (pi *
glm::pow(smoothingRadius, 6));
    kernelScalingFactorSpikyPow2 = 15 / (2 * pi *
glm::pow(smoothingRadius, 5));
    kernelScalingFactorSpikyPow3Derivative = 45 / (pi *
glm::pow(smoothingRadius, 6));
    kernelScalingFactorSpikyPow2Derivative = 15 / (pi *
glm::pow(smoothingRadius, 5));
}
```

Kod 2.4.1 Konstruktor razreda *Fluid*

Potrebni spremnici su alocirani i zadani broj čestica stvoren je u zadanoj konfiguraciji. Također su unaprijed izračunati faktori skaliranja za jezgre zaglađivanja. Naime, jezgre je potrebno normalizirati, a faktor skaliranja ovisi samo o radijusu zaglađivanja, tako da su ti faktori izračunati odmah i koriste se kasnije.

2.4.2. Metoda `CalculateDensities`

```
glm::vec2 Fluid::CalculateDensities(const glm::vec3
samplePoint) const
{
    float density = 0;
    float nearDensity = 0;

    ForeachPointWithinRadius(samplePoint, [&](const int
pointIndex)
    {
```

```
        const glm::vec3 position =
predictedPositions[pointIndex];
        const float distance = glm::length(position -
samplePoint);
        density += SmoothingKernelSpikyPow2(smoothingRadius,
distance);
        nearDensity +=
SmoothingKernelSpikyPow3(smoothingRadius, distance);
    });
    return {density, nearDensity};
}
float Fluid::SmoothingKernelSpikyPow2(const float radius,
const float distance) const
{
    if (distance >= radius) return 0;

    const float value = radius - distance;
    return value * value * kernelScalingFactorSpikyPow2;
}
float Fluid::SmoothingKernelSpikyPow3(const float radius,
const float distance) const
{
    if (distance >= radius) return 0;

    const float value = radius - distance;
    return value * value * value *
kernelScalingFactorSpikyPow3;
}
```

Kod 2.4.2 Metoda *CalculateDensities i jegre zaglađivanja*

U ovoj metodi akumuliraju se gustoća i bliska gustoća za zadanu točku u prostoru. Pretpostavljena masa čestica je 1, tako da se mogu samo zbrojiti vrijednosti jezgri zaglađivanja za gustoću i blisku gustoću, što onda odgovara formuli (5). Korištene su jednostavne kvadratne i kubične šiljaste jezgre [12]. Pozicije čestica čitaju se iz spremnika `predictedPositions`, kao što je navedeno na početku poglavlja 2.4. Za efikasan prolazak kroz čestice u radijusu zaglađivanja korištena je pomoćna metoda `ForeachPointWithinRadius`, koja je jedan dio implementacije podjele prostora na ćelije. U nastavku će biti objašnjena ta implementacija.

2.4.3. Metoda `UpdateSpatialLookup`

```
void Fluid::UpdateSpatialLookup()
{
    // create unordered spatial lookup
    for_each(executionPolicy, particleIndices.begin(),
particleIndices.end(), [&](auto i)
    {
        const glm::ivec3 cell =
PositionToCellCoord(predictedPositions[i], smoothingRadius);
        const unsigned int cellKey =
GetKeyFromHash(HashCell(cell));
        spatialLookup[i] = SpatialLookupEntry {
            i,
            cellKey
        };
        startIndices[i] = std::numeric_limits<int>::max();
    });

    // sort by cell key
    std::sort(executionPolicy, spatialLookup.begin(),
spatialLookup.end());

    // calculate start indices of each unique cell key in the
spatial lookup
    for_each(executionPolicy, particleIndices.begin(),
particleIndices.end(), [&](auto i)
    {
        const unsigned int key = spatialLookup[i].cellKey;
        if (i == 0 || spatialLookup[i - 1].cellKey != key)
        {
            startIndices[key] = i;
        }
    });
}
```

Kod 2.4.3 Metoda *UpdateSpatialLookup*

Ovu metoda mora biti pozvana na početku izračunavanja koraka simulacije da bi se omogućilo brzo iteriranje kroz čestice u bilo kojoj ćeliji. Prvo je za svaku česticu izračunat

ključ njene ćelije, potom je lista za prostorno pretraživanje sortirana po ključevima ćelija, zatim je pronađen i zabilježen početni indeks u toj listi za svaki prisutni ključ ćelije.

2.4.4. Metoda `ForeachPointWithinRadius`

Ova metoda koristi prethodno pripremljenu strukturu da bi efikasno iterirala kroz sve točke unutar radijusa zaglađivanja od zadane točke:

```
template <typename F>
void Fluid::ForeachPointWithinRadius(const glm::vec3
samplePoint, F&& f) const
{
    const glm::ivec3 center =
PositionToCellCoord(samplePoint, smoothingRadius);
    const float sqrRadius = smoothingRadius *
smoothingRadius;
    for (int offsetX = -1; offsetX <= 1; offsetX++)
    {
        for (int offsetY = -1; offsetY <= 1; offsetY++)
        {
            for (int offsetZ = -1; offsetZ <= 1; offsetZ++)
            {
                const glm::ivec3 cell = {center.x + offsetX,
center.y + offsetY, center.z + offsetZ};
                const unsigned int key =
GetKeyFromHash(HashCell(cell));
                const int cellStartIndex = startIndices[key];

                for (int i = cellStartIndex; i <
particleCount; i++)
                {
                    const SpatialLookupEntry
spatialLookupEntry = spatialLookup[i];
                    // exit if we're no longer at the correct
cell
                    if (spatialLookupEntry.cellKey != key)
break;

                    const int particleIndex =
spatialLookupEntry.particleIndex;
```

```
                    const auto offset =
predictedPositions[particleIndex] - samplePoint;
                    const float sqrDistance =
glm::dot(offset, offset);

                    // test if the point is inside the radius
                    if (sqrDistance <= sqrRadius)
                    {
                        std::forward<F>(f)(particleIndex);
                    }
                }
            }
        }
    }
}
```

Kod 2.4.4 Metoda *ForeachPointWithinRadius*

Metoda određuje ćeliju u kojoj se nalazi zadana pozicija i iterira kroz tu ćeliju i 26 relevantnih okolnih ćelija. Za svaku česticu provjerava jeli unutar radijusa zaglađivanja i ako je, poziva zadanu funkciju `f` s indeksom te čestice. Ova funkcija koristi predložak (engl. template) da bi za svaku česticu unutar radijusa vrlo efikasno izvela bilo koji kod. Za vrijeme prevođenja programa generira se nova verzija koda za svaki različiti poziv ove funkcije i zato svaka primjena može u potpunosti biti optimizirana od strane prevoditelja.

2.4.5. Metoda `CalculatePressureForce`

```
glm::vec3 Fluid::CalculatePressureForce(const int
particleIndex) const
{
    glm::vec3 pressureForce(0);
    const float density = densities[particleIndex].x;
    const float nearDensity = densities[particleIndex].y;
    const float pressure = DensityToPressure(density);
    const float nearPressure =
NearDensityToNearPressure(nearDensity);

    const glm::vec3 particlePosition =
predictedPositions[particleIndex];
    ForeachPointWithinRadius(particlePosition, [&](const int
otherParticleIndex)
```

```
    {
        if (otherParticleIndex != particleIndex)
        {
            const glm::vec3 offset =
predictedPositions[otherParticleIndex] - particlePosition;
            const float distance = glm::length(offset);
            // pick a random direction in case two particles
share the same position
            const glm::vec3 direction = distance == 0 ?
glm::sphericalRand(1.0f) : offset / distance;
            const float otherDensity =
densities[otherParticleIndex].x;
            const float otherNearDensity =
densities[otherParticleIndex].y;
            const float otherPressure =
DensityToPressure(otherDensity);
            const float otherNearPressure =
NearDensityToNearPressure(otherNearDensity);

            const float sharedPressure = (pressure +
otherPressure) * 0.5f;
            const float sharedNearPressure = (nearPressure +
otherNearPressure) * 0.5f;
            pressureForce += sharedPressure * direction
                *
SmoothingKernelSpikyPow2Derivative(smoothingRadius, distance)
/ otherDensity;
            pressureForce += sharedNearPressure * direction
                *
SmoothingKernelSpikyPow3Derivative(smoothingRadius, distance)
/ otherNearDensity;
        }
    });
    return pressureForce;
}

float Fluid::DensityToPressure(const float density) const
{
    const float densityError = density - targetDensity;
    return densityError * pressureMultiplier;
}
```

```
float Fluid::NearDensityToNearPressure(const float
nearDensity) const
{
    return nearDensity * nearPressureMultiplier;
}
float Fluid::SmoothingKernelSpikyPow2Derivative(const float
radius, const float distance) const
{
    if (distance >= radius) return 0;

    const float value = radius - distance;
    return -value * kernelScalingFactorSpikyPow2Derivative;
}
float Fluid::SmoothingKernelSpikyPow3Derivative(const float
radius, const float distance) const
{
    if (distance >= radius) return 0;

    const float value = radius - distance;
    return -value * value *
kernelScalingFactorSpikyPow3Derivative;
}
```

Kod 2.4.5 Metoda *CalculatePressureForce*, pomoćne metode i derivacije jezgri zaglađivanja

U ovoj metodi izračunava se sila pritiska na zadanu česticu. Iz pomoćnih funkcija vidljivo je da će sila bliskog pritiska uvijek biti odbojna (pozitivan pritisak), a osnovna sila pritiska može biti odbojna ili privlačna, ovisno o trenutnoj gustoći i ciljanoj gustoći (pritisak može biti pozitivan ili negativan). Za izračun ovih sila potrebne su derivacije jezgri zaglađivanja koje su korištene za izračun gustoće i bliske gustoće. Radi simetričnosti sile korišten je zajednički pritisak koji je aritmetička sredina između vrijednost pritiska za obje čestice. Također, iznos sile mora biti podijeljen s gustoćom druge čestice, kao što je navedeno u formuli (7).

2.4.6. Metoda `CalculateViscosityForce`

```
glm::vec3 Fluid::CalculateViscosityForce(const int
particleIndex) const
{
    glm::vec3 viscosityForce(0);
```

```
    const glm::vec3 particlePosition =
predictedPositions[particleIndex];

    ForeachPointWithinRadius(particlePosition, [&](const int
otherParticleIndex)
    {
        if (otherParticleIndex != particleIndex)
        {
            const float distance =
glm::length(particlePosition -
predictedPositions[otherParticleIndex]);
            const float otherDensity =
densities[otherParticleIndex].x;
            const glm::vec3 relativeVelocity =
velocities[otherParticleIndex] - velocities[particleIndex];
            viscosityForce += relativeVelocity
                * SmoothingKernelFlat(smoothingRadius,
distance) / otherDensity;
        }
    });
    return viscosityForce * viscosityStrength;
}
float Fluid::SmoothingKernelFlat(const float radius, const
float distance) const
{
    if (distance >= radius) return 0;

    const float value = radius * radius - distance *
distance;
    return value * value * value * kernelScalingFactorFlat;
}
```

Kod 2.4.6 Metoda *CalculateViscosityForce* i jezgra zaglađivanja

Slično kao dosadašnje metode, korištena je pomoćna metoda `ForeachPointWithinRadius` da bismo prošli kroz sve relevantne čestice i izračunali silu viskoznosti za zadanu česticu prema formuli (9). Korištena je glatka jezgra zaglađivanja [1]. Sila se na kraju množi s konstantom viskoznosti tekućine.

2.4.7. Metoda `CalculateInteractionForce`

```
glm::vec3 Fluid::CalculateInteractionForce(const glm::vec3
inputPos, const float radius, const float strength, const int
particleIndex) const
{
    glm::vec3 interactionForce = glm::vec3(0);
    const glm::vec3 offset = inputPos -
currentPositions[particleIndex];
    const float sqrDst = glm::dot(offset, offset);

    if (sqrDst < radius * radius)
    {
        const float dst = glm::sqrt(sqrDst);
        const glm::vec3 dirToInputPoint = dst <=
glm::epsilon<float>() ? glm::vec3(0) : offset / dst;
        // influence is 1 when particle is at input point, 0
when at the edge of input radius
        const float influence = 1 - dst / radius;
        // velocity is subtracted to slow the particle down
        interactionForce += (dirToInputPoint * strength -
velocities[particleIndex]) * influence;
    }
    return interactionForce;
}
```

Kod 2.4.7 Metoda *CalculateInteractionForce*

Ovo je jednostavna metoda za izračun sile interakcije s određenim parametrima na zadanu česticu. Cilj je dobiti silu koja usporava česticu i privlači ju prema točki interakcije (ako je parametar `strength` pozitivan) ili ju odbija od točke interakcije (ako je parametar `strength` negativan).

2.4.8. Metoda `Update`

Metoda `Update` koristi do sada objašnjene metode da izvrši cijeli jedan korak simulacije:

```
void Fluid::Update(const float dt, const glm::vec3 gravity,
const optional<glm::vec4> interaction)
{
    for_each(executionPolicy, particleIndices.begin(),
particleIndices.end(), [&](auto i)
```

```
    {
        forces[i] = gravity;
        predictedPositions[i] = currentPositions[i] +
currentPositions[i] - previousPositions[i];
    });
    if (interaction.has_value())
    {
        for_each(executionPolicy, particleIndices.begin(),
particleIndices.end(), [&](auto i)
        {
            forces[i] +=
CalculateInteractionForce(interaction.value(), 2, 200 *
interaction.value().w, i);
        });
    }

    UpdateSpatialLookup();

    for_each(executionPolicy, particleIndices.begin(),
particleIndices.end(), [&](auto i)
    {
        densities[i] =
CalculateDensities(predictedPositions[i]);
    });

    for_each(executionPolicy, particleIndices.begin(),
particleIndices.end(), [&](auto i)
    {
        forces[i] += CalculatePressureForce(i);
    });
    for_each(executionPolicy, particleIndices.begin(),
particleIndices.end(), [&](auto i)
    {
        forces[i] += CalculateViscosityForce(i);
    });

    for_each(executionPolicy, particleIndices.begin(),
particleIndices.end(), [&](auto i)
    {
        glm::vec3 acceleration = forces[i] / densities[i].x;
        glm::vec3 currentPosition = currentPositions[i];
```

```
        glm::vec3 nextPosition = 2.0f * currentPosition -
previousPositions[i] + acceleration * dt * dt;

        // calculate collisions
        const float bounceFactor = 0.05f;
        const glm::vec3 minPos = glm::vec3(min(corner1.x,
corner2.x), min(corner1.y, corner2.y), min(corner1.z,
corner2.z));
        const glm::vec3 maxPos = glm::vec3(max(corner1.x,
corner2.x), max(corner1.y, corner2.y), max(corner1.z,
corner2.z));
        if (nextPosition.x < minPos.x)
        {
            nextPosition = Mirror(nextPosition, minPos,
glm::vec3(1, 0, 0));
            currentPosition = Mirror(currentPosition, minPos,
glm::vec3(1, 0, 0));
            currentPosition += (nextPosition -
currentPosition) * (1 - bounceFactor);
        } else if (maxPos.x < nextPosition.x)
        {
            nextPosition = Mirror(nextPosition, maxPos,
glm::vec3(-1, 0, 0));
            currentPosition = Mirror(currentPosition, maxPos,
glm::vec3(-1, 0, 0));
            currentPosition += (nextPosition -
currentPosition) * (1 - bounceFactor);
        }
        if (nextPosition.y < minPos.y)
        {
            nextPosition = Mirror(nextPosition, minPos,
glm::vec3(0, 1, 0));
            currentPosition = Mirror(currentPosition, minPos,
glm::vec3(0, 1, 0));
            currentPosition += (nextPosition -
currentPosition) * (1 - bounceFactor);
        } else if (maxPos.y < nextPosition.y)
        {
            nextPosition = Mirror(nextPosition, maxPos,
glm::vec3(0, -1, 0));
```

```
            currentPosition = Mirror(currentPosition, maxPos,
glm::vec3(0, -1, 0));
            currentPosition += (nextPosition -
currentPosition) * (1 - bounceFactor);
        }
        if (nextPosition.z < minPos.z)
        {
            nextPosition = Mirror(nextPosition, minPos,
glm::vec3(0, 0, 1));
            currentPosition = Mirror(currentPosition, minPos,
glm::vec3(0, 0, 1));
            currentPosition += (nextPosition -
currentPosition) * (1 - bounceFactor);
        } else if (maxPos.z < nextPosition.z)
        {
            nextPosition = Mirror(nextPosition, maxPos,
glm::vec3(0, 0, -1));
            currentPosition = Mirror(currentPosition, maxPos,
glm::vec3(0, 0, -1));
            currentPosition += (nextPosition -
currentPosition) * (1 - bounceFactor);
        }
        previousPositions[i] = currentPosition;
        currentPositions[i] = nextPosition;
        velocities[i] = (currentPositions[i] -
previousPositions[i]) / dt;
        object-
>transforms[i].SetPosition(currentPositions[i]);
    });
}
```

Kod 2.4.8 Metoda *Update*

Na početku koraka simulacije primjenjuju se vanjske sile i predviđaju pozicije čestica za izračun unutarnjih sila. Prema jednadžbi (3), volumne sile poput gravitacije trebale bi biti proporcionalne gustoći fluida na mjestu čestice, ali to je uzrokovalo značajno lošije rezultate pri simulaciji nestlačivih fluida. Na dnu fluida je gustoća bila viša, pa je to uzrokovalo dodatnu silu gravitacije, što bi dovelo do dodatnog povećanja gustoće, konačno dovodeći do vrlo neravnomjerne gustoće. Stoga je dodana jednaka sila gravitacije na sve čestice, neovisno o gustoći. Nakon toga je ažurirana lista za prostorno pretraživanje koja se koristi u izračunu gustoća i unutarnjih sila. Potom je izračunata gustoća fluida na poziciji svake

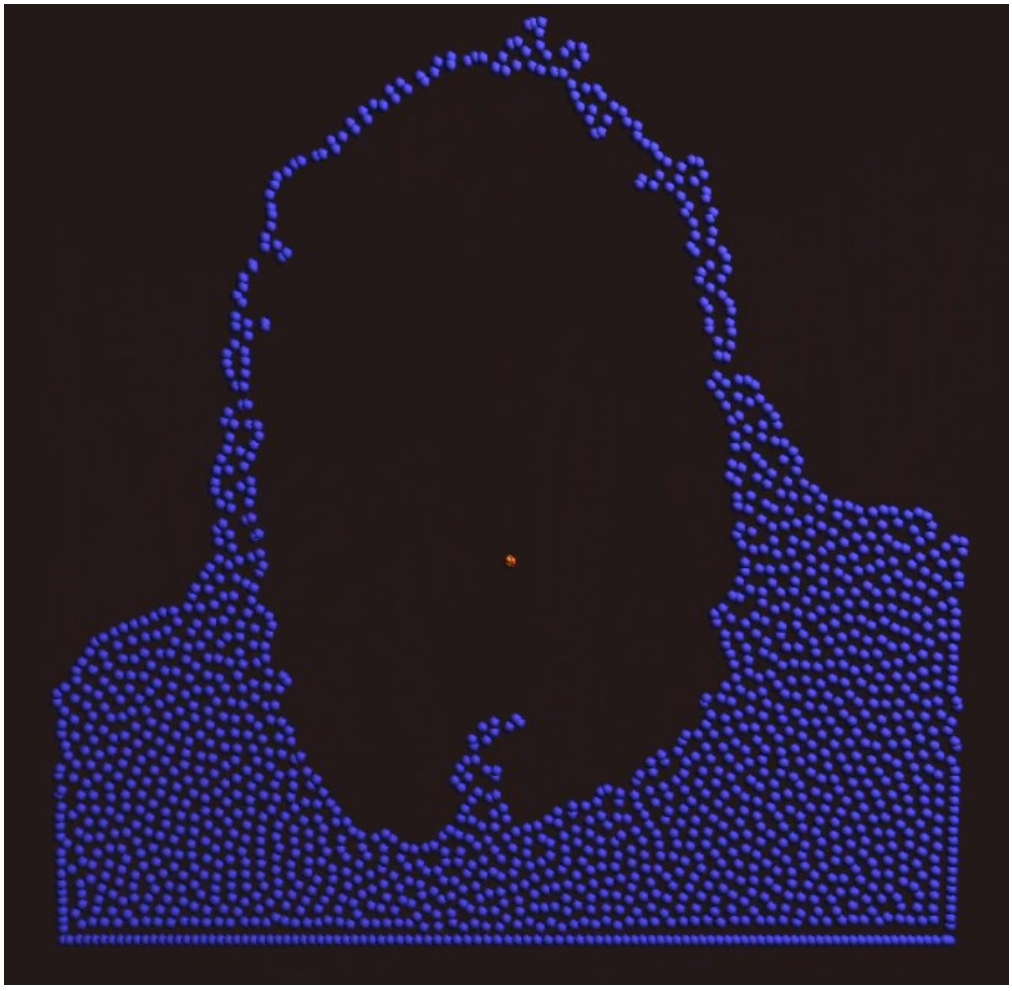
čestice, što je potrebno za daljnje izračune. Nakon toga je izračunata sila pritiska i viskoznosti kako je objašnjeno u dosadašnjim poglavljima. Konačno, pozicije čestica su integrirane koristeći Verlet integraciju. Pritom je sila podijeljena s gustoćom čestice prema jednadžbi (3) radi izračunavanja ubrzanja čestice. Konačno, čestice koje su izašle iz ograničenja simulacije odbijamo nazad s određenim gubitkom brzine.

3. Rezultati

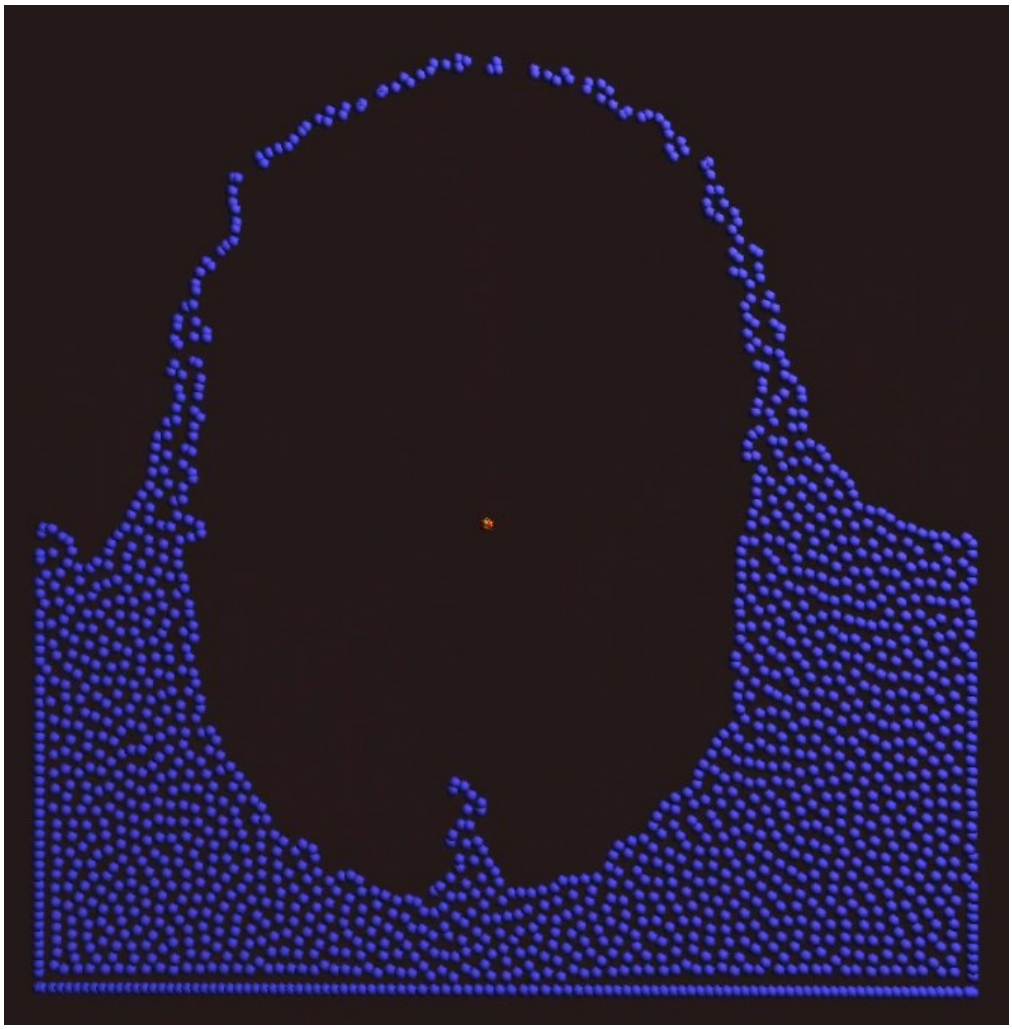
Bit će demonstriran utjecaj nekih parametara simulacije na postignute rezultate. Većinu demonstracija bit će napravljene na 2D verziji simulacije s obzirom da je u toj verziji lakše vidjeti i razumjeti što se događa. Principi se prenose i na 3D verziju simulacije.

3.1. Konstanta viskoznosti

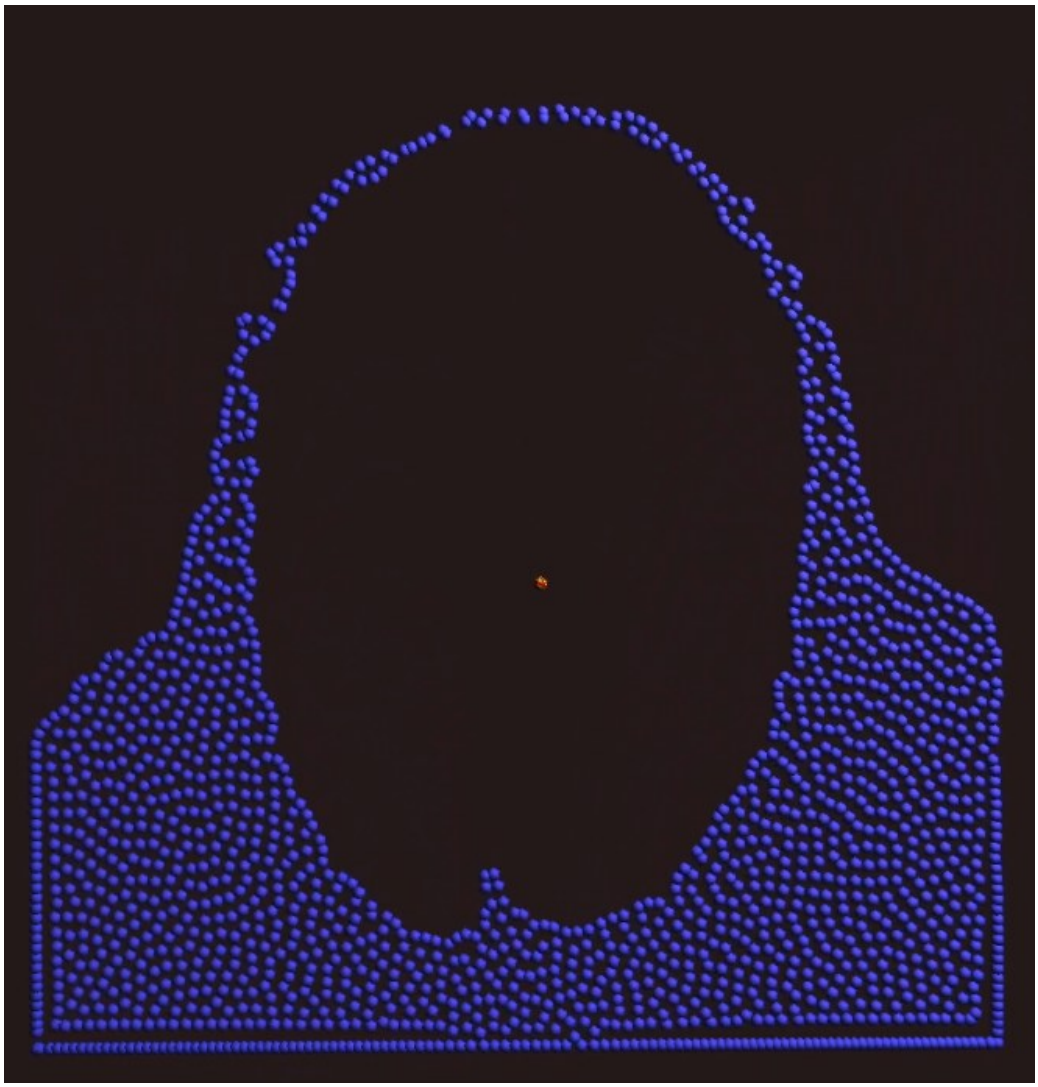
Viskoznost predstavlja unutarnje trenje fluida. Čestice unutar viskoznije tekućine težit će se kretati jednako kao okolne čestice. Promotrimo kako se fluid ponaša za razne vrijednosti konstante viskoznosti μ iz formule (10). Da bismo vidjeli razliku u viskoznosti, moramo promatrati fluid u pokretu, pa ćemo „razrezati“ kroz tekućinu prema gore koristeći odbojnu silu interakcije. Crveno-žuta točkica označava zadnje mjesto na kojem je primijenjena sila interakcije.



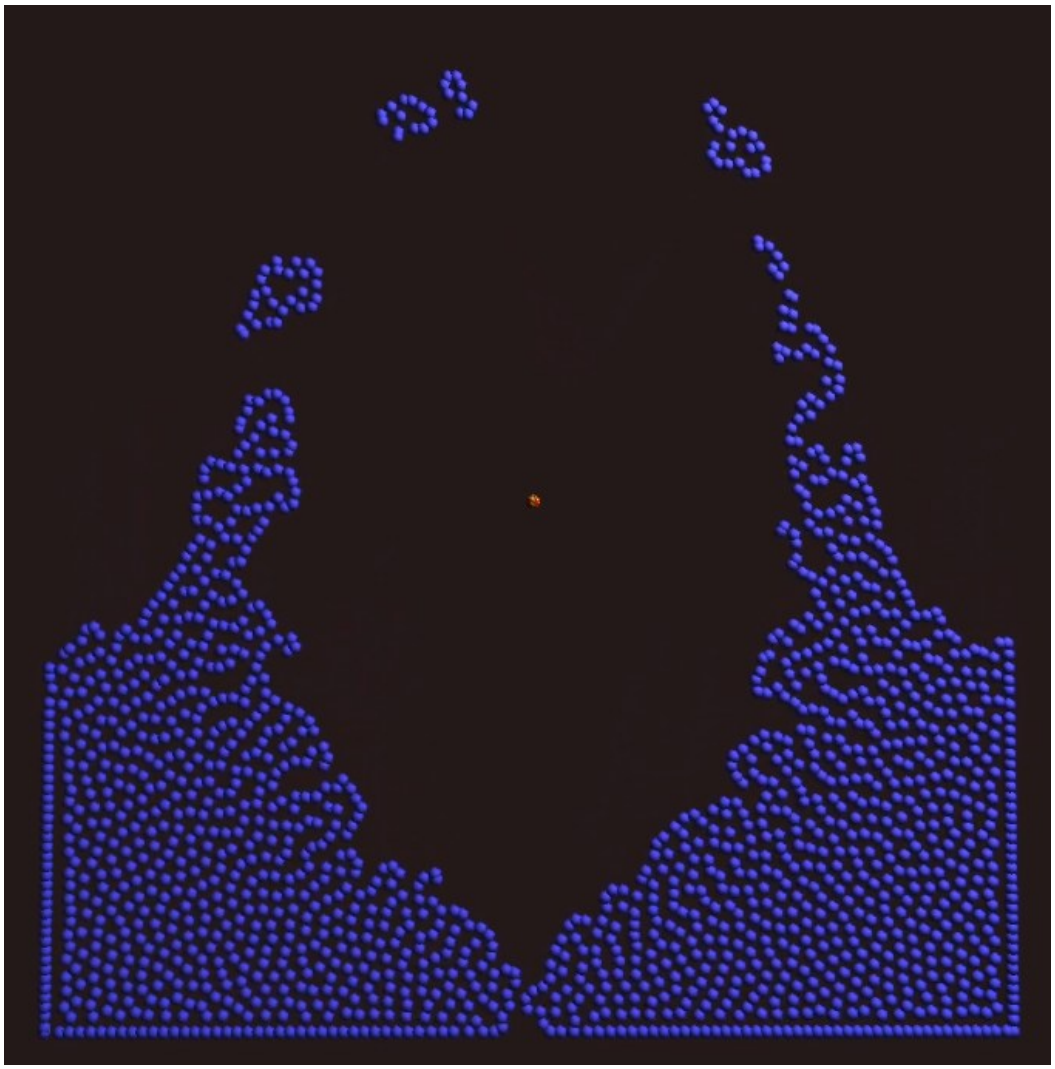
Slika 3.1.1 Konstanta viskoznosti je 0



Slika 3.1.2 Konstanta viskoznosti je 1



Slika 3.1.3 Konstanta viskoznosti je 10

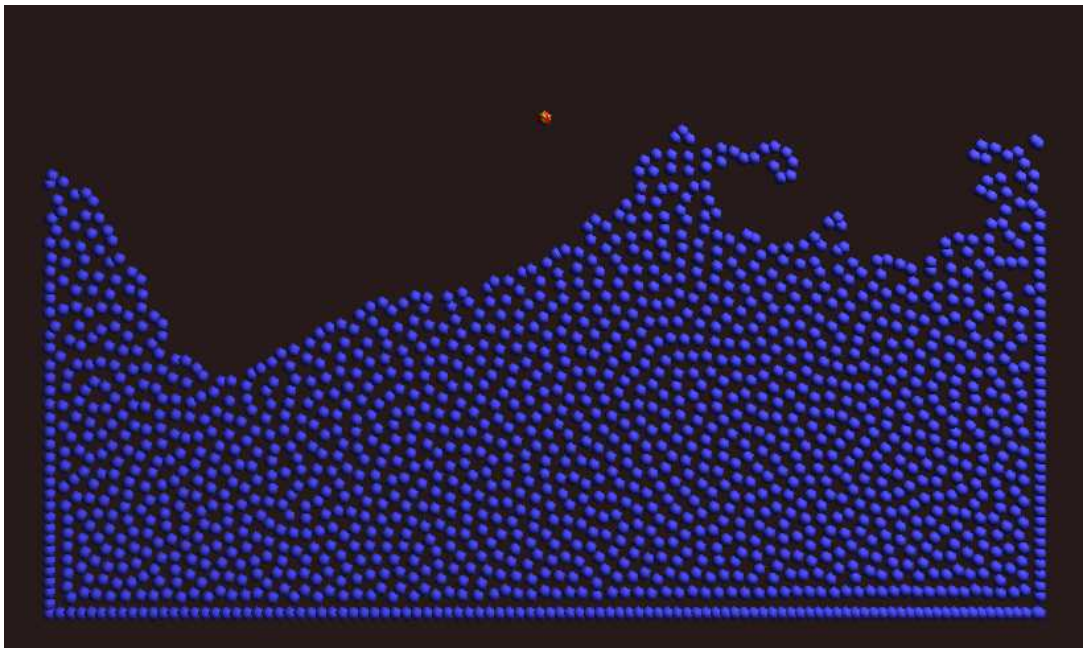


Slika 3.1.4 Konstanta viskoznosti je 100

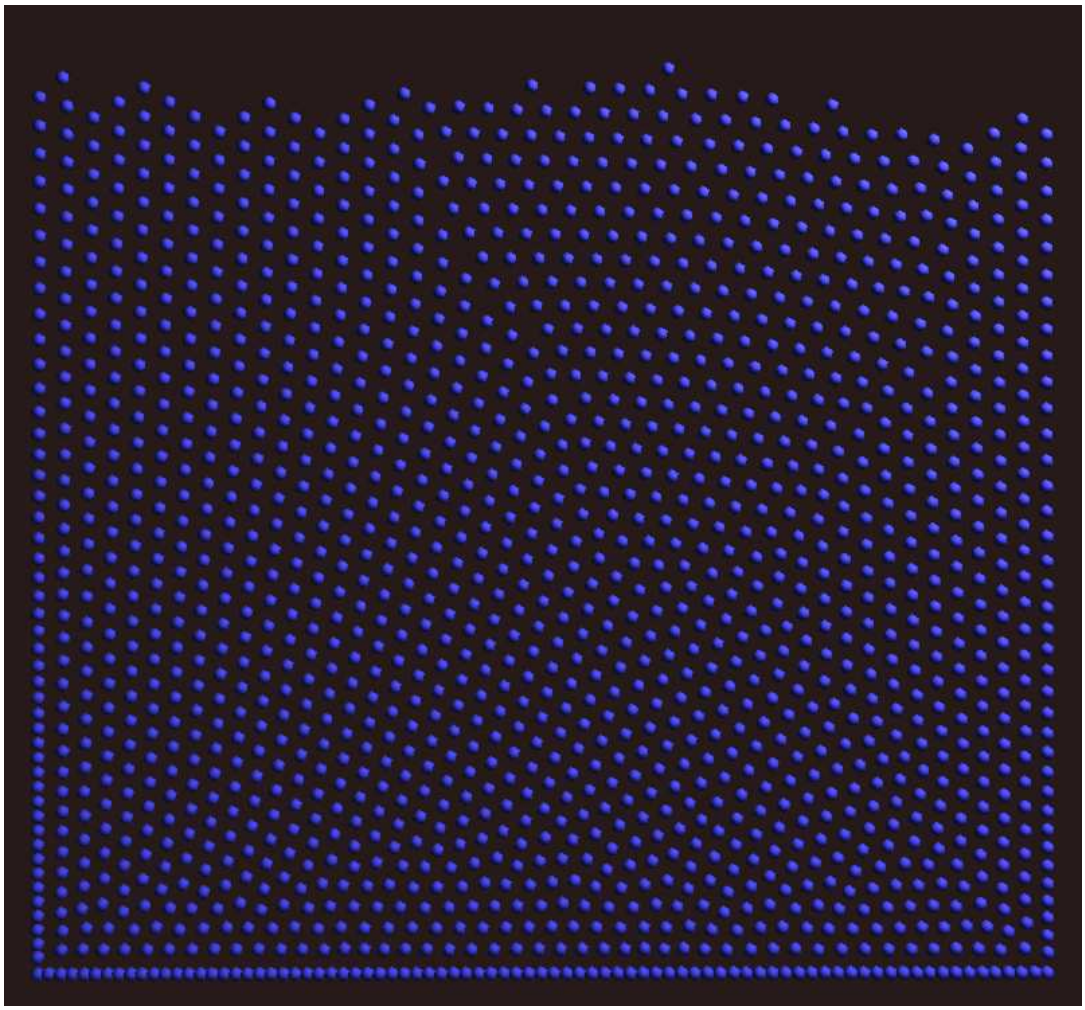
Primijetimo da je ponašanje tekućine progresivno sve više zaglađeno kako povećavamo konstantu viskoznosti. Na ekstremnim vrijednostima konstante viskoznosti kao 100, fluid prestaje nalikovati na tekućinu i počinje nalikovati nekakvom želeu. Trga se umjesto da teče. Vrijednost konstante između 0 i 1 najbolje odgovara simulaciji tipičnih tekućina poput vode.

3.2. Ciljana gustoća

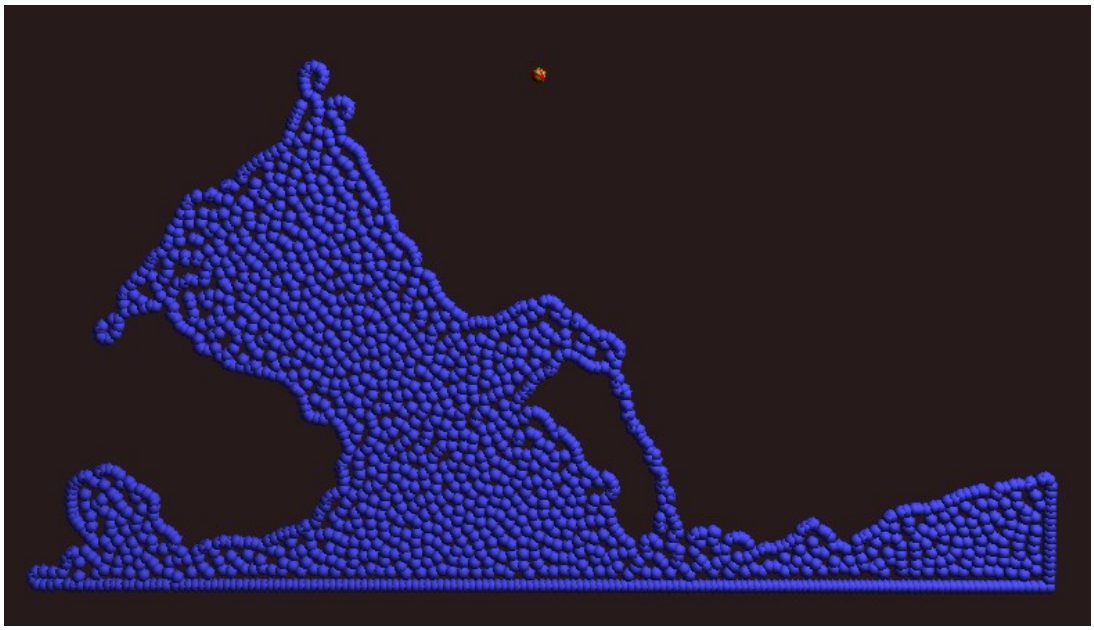
Ciljana gustoća fluida ρ_0 iz formule (9) određuje gustoću kojoj će fluid težiti. Ako je gustoća fluida na nekom mjestu viša od ciljane, čestice će se privlačiti, a ako je niža, čestice će se odbijati. Pogledajmo izgled fluida sa nekoliko različitih vrijednosti.



Slika 3.2.1 Ciljana gustoća je 30



Slika 3.2.2 Ciljana gustoća je 10



Slika 3.2.3 Ciljana gustoća je 60

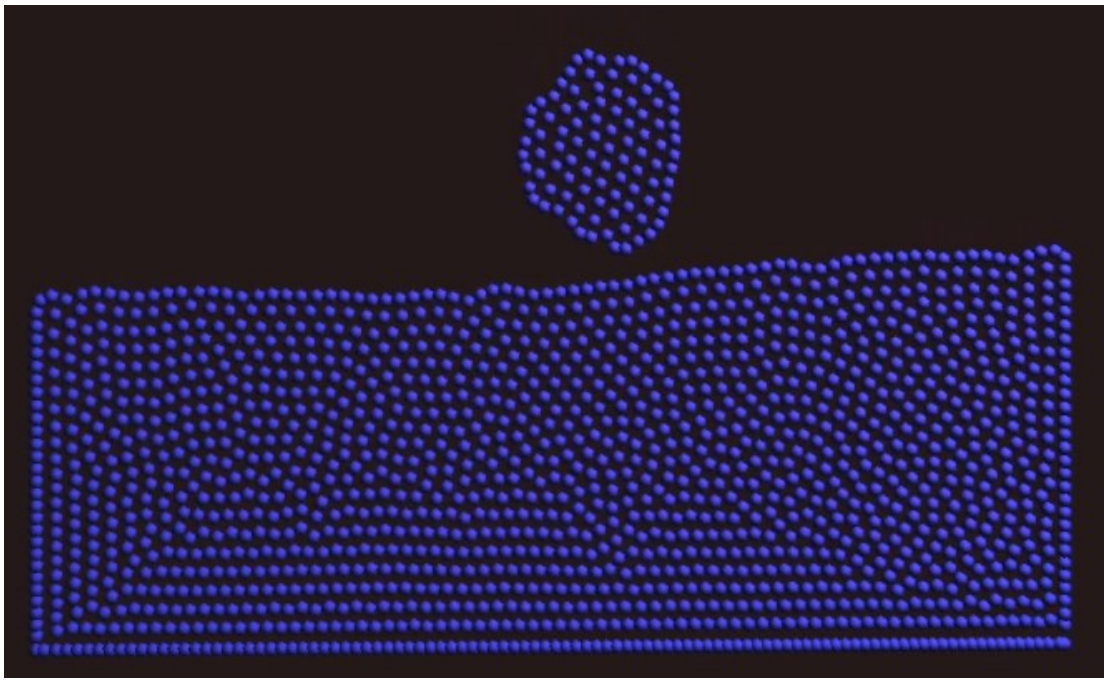
Primijetimo da su čestice međusobno bliže kada je ciljana gustoća viša, kao što je očekivano. Fluid sa većom ciljanom gustoćom također se ponaša viskoznije. To je vjerojatno zbog toga što je sila viskoznosti jača što su čestice međusobno bliže.

3.3. Konstanta sile pritiska

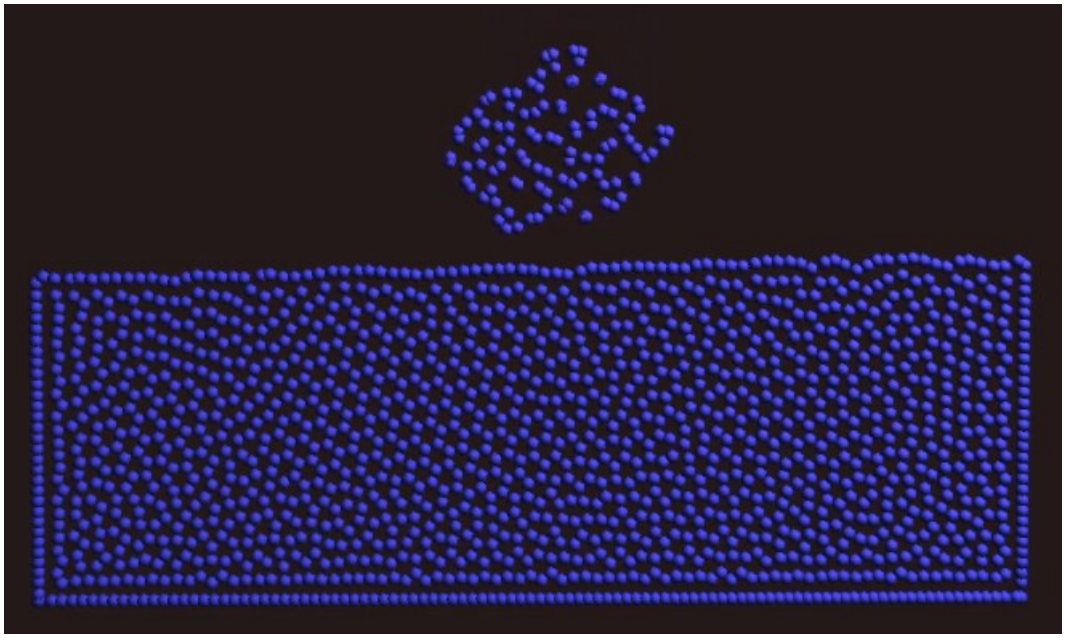
Konstanta sile pritiska (engl. stiffness) k iz formule (8) skalira iznos sile pritiska između čestica. Niže vrijednosti mogu dovesti do neujednačene gustoće i do osciliranja fluida. Za nestlačive fluide većinom je dobro koristiti višu vrijednost konstante sile pritiska, ali povećavanje te konstante povećava nestabilnost simulacije i zahtijeva korištenje manjeg vremenskog koraka.

3.4. Bliska gustoća

Kao što je objašnjeno u poglavlju 2.2, dodatak bliske gustoće i sile bliskog pritiska pomaže nam u ostvarenju višeg stupnja kohezije fluida i primitivnog efekta površinske napetosti. Pogledajmo na primjeru kapljice. Silom interakcije podignuta je skupina čestica iznad površine fluida i puštena da padne. Slike su uslikane nakon određene količine padanja, prije nego je kapljica dodirnula ostatak tekućine.



Slika 3.4.1 Konstanta bliske gustoće je 2



Slika 3.4.2 Konstanta bliske gustoće je 0

Vidimo da kapljica puno bolje drži svoj oblik kada je vrijednost konstante 2, a kada je vrijednost konstante 0, vidimo da se kapljica raspada na male skupine od dvije do tri čestice. Odbojna sila bliske gustoće sprječava takvo ponašanje. Vizualni učinak ove sile na sveukupno ponašanje fluida vrlo je pozitivan.

3.5. Broj čestica

Promotrimo kako broj čestica utječe na brzinu simulacije.

Komponente računala korištenog za testove:

- CPU: AMD Ryzen 5 5600H
- RAM: 16 GB
- GPU: NVIDIA Geforce RTX 3050 Ti Laptop GPU

3D performanse i 2D performanse trebale bi se značajno razlikovati jer čestice u 3D prostoru mogu imati puno veći broj susjeda.

3.5.1. 2D performanse

Tablica 3.5.1 performanse 2D simulacije

Inicijalna mreža	Broj čestica	Vrijeme izračunavanja koraka simulacije	Broj koraka simulacije po sekundi
10 x 10	100	0.37 ms	2715.97
30 x 30	900	1.5 ms	666.65
50 x 50	2500	3.95 ms	252.61
70 x 70	4900	6.23 ms	160.47
90 x 90	8100	10.24 ms	97.67
110 x 110	12100	14.88 ms	67.2
120 x 120	14400	18.03 ms	55.45

Dvodimenzionalna verzija simulacije je prilično široko upotrebljiva u stvarnom vremenu, čak i za simulacije na većoj skali.

3.5.2. 3D performanse

Tablica 3.5.2 performanse 3D simulacije

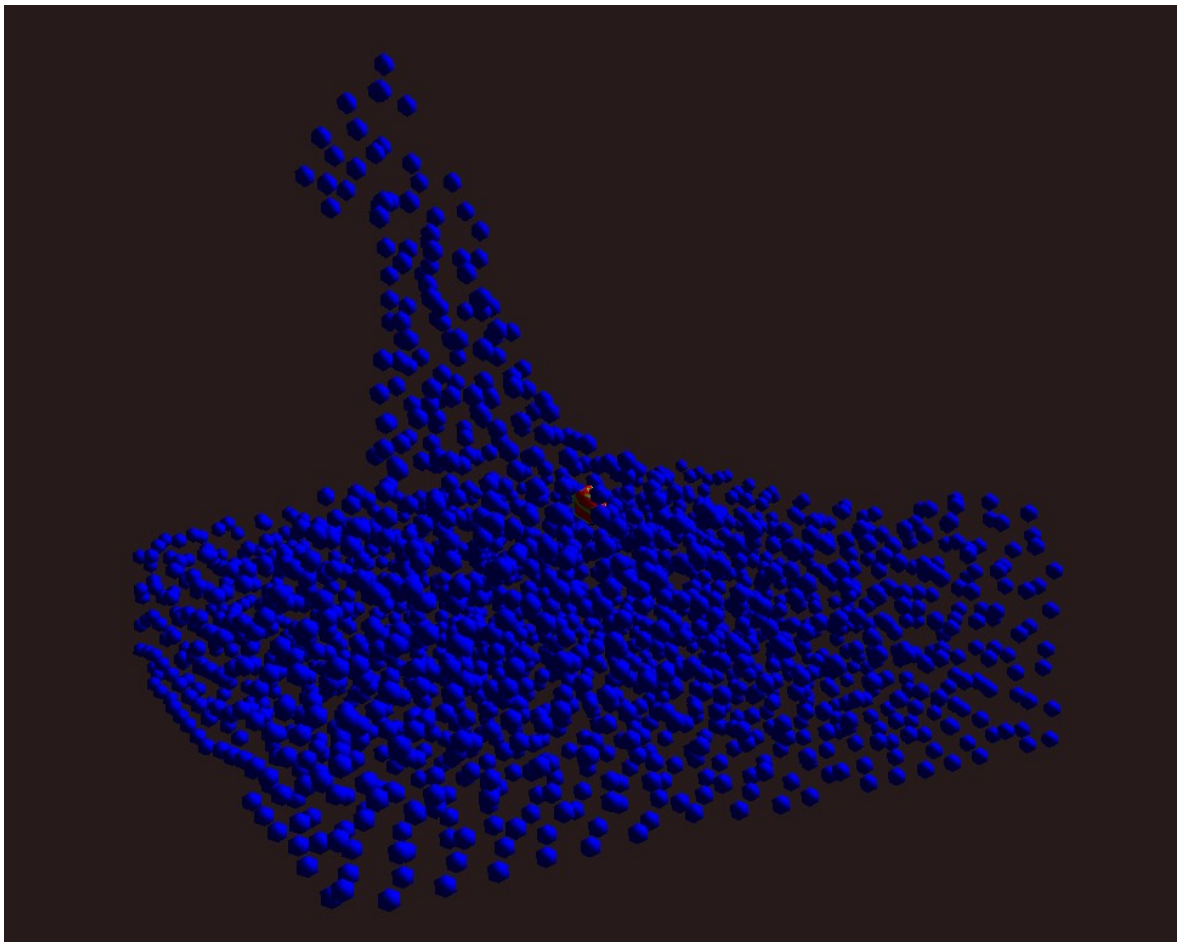
Inicijalna mreža	Broj čestica	Vrijeme izračunavanja koraka simulacije	Broj koraka simulacije po sekundi
5 x 5 x 5	125	2.12 ms	470.86
7 x 7 x 7	343	3.96 ms	252.45
9 x 9 x 9	729	6.08 ms	164.41
11 x 11 x 11	1331	8.16 ms	122.53
12 x 12 x 12	1728	10.87 ms	92.0
13 x 13 x 13	2197	14.83 ms	67.45
14 x 14 x 14	2744	19.37 ms	51.62

Trodimenzionalna verzija simulacije je vrlo ograničene upotrebljivosti u stvarnom vremenu. Potencijalno bi se mogla koristiti za lokalizirane efekte koji se sastoje od umjerenog broja čestica. Količina posla po čestici je puno viša zbog većeg potencijalnog broja susjeda svake čestice. Prebacivanje simulacije na GPU uz pomoć računskih sjenčara (engl. compute shader) omogućilo bi puno širu upotrebljivost 3D verzije simulacije.

3.6. Moguća poboljšanja

Najveća prilika za poboljšanje bila bi sofisticiranija metoda vizualizacije fluida. Algoritam pokretnih kocki vjerojatno bi bio sljedeći dobar pristup za isprobati. Takva vizualizacija dovela bi do jasnijeg ponašanja 3D verzije simulacije.

Nakon toga, prebacivanje paralelizacije simulacije sa CPU na GPU uz pomoć računskih sjenčara omogućilo bi skaliranje simulacije na puno veći broj čestica.



Slika 3.6.1 Izgled 3D simulacije

Zaključak

Iz rezultata testova smo vidjeli da simulacija fluida metodom zaglađenih čestica uz dodatak bliske gustoće i jednostavnu paralelizaciju na CPU daje široko upotrebljive rezultate za dvodimenzionalnu simulaciju fluida za primjene u stvarnom vremenu poput računalnih igara. S druge strane, trodimenzionalna verzija takve simulacije je vrlo ograničena brojem čestica i stoga je ograničene upotrebe, potencijalno za manje, lokalne efekte.

Fluidi su vrlo kompleksan fizikalni fenomen koji utječe na velik broj stvari oko nas. Ponašanje fluida može biti vrlo kompliciranо i naizgled neshvatljivo, ali uz proučavanje Navier-Stokes jednadžbi možemo shvatiti da su pravila koja fluidi slijede iznenađujuće intuitivna. Implementacija čestične simulacije fluida može nas dodatno približiti tom razumijevanju.

Literatura

[1] M. Müller, D. Charypar i M. Gross, »Particle-Based Fluid Simulation for Interactive Applications,« u *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2003.

[2] R. Gingold i J. Monaghan, »Smoothed particle hydrodynamics: theory and application to non-spherical stars,« *Monthly Notices of the Royal Astronomical Society,* 1977.

[3] L. Lucy, »A numerical approach to the testing of the fission hypothesis,« *The Astronomical Journal,* 1977.

[4] N. Foster i D. Metaxas, »Realistic Animation of Liquids,« *Graphical Models and Image Processing,* 1996.

[5] R. Bridson i M. Müller-Fischer, »Fluid Simulation,« u *ACM SIGGRAPH 2007 Courses*, 2007.

[6] J. U. Brackbill, D. B. Kothe i H. M. Ruppel, »FLIP: A Low-Dissipation, Particle-in-Cell Method for Fluid Flow,« *Computer Physics Communications,* 1988.

[7] W. E. Lorensen i H. E. Cline, »Marching Cubes: A High resolution 3D Surface Construction Algorithm,« u *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '87)*, 1987.

[8] J. Wong, »https://jamie-wong.com/2016/07/15/ray-marching-signed-distance-functions/#the-raymarching-algorithm,« 2016.

[9] S. Green, »Screen Space Fluid Rendering for Games,« Nvidia Corporation, 2010.

[10] M. Ihmsen, N. Akinci, G. Akinci i M. Teschner, »Unified Spray, Foam and Bubbles for Particle-Based Fluids,« *The Visual Computer,* svez. 28, 2012.

[11] S. Green, »Particle Simulation using CUDA,« Nvidia Corporation, 2012.

[12] S. Clavet, P. Beaudoin i P. Poulin, »Particle-based Viscoelastic Fluid Simulation,« u *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposion on Computer Animation*, 2005.

[13] M. Macklin i M. Müller, »Position Based Fluids,« *ACM Transactions on Graphics,* svez. 32, br. 4, 2013.

Izjava o korištenju umjetne inteligencije

Koristio sam ChatGPT (verzija GPT-5.3-mini).

Osvrt na korištenje umjetne inteligencije: Koristio sam ChatGPT za prijevod sažetka na engleski. Dobiveni prijevod sam doradio. Također sam koristio ChatGPT za pronalazak bitnih znanstvenih radova iz simulacije i vizualizacije fluida.

Izjava o korištenju umjetne inteligencije: Potvrđujem da sam tijekom izrade ovog završnog rada koristio umjetnu inteligenciju u skladu s Politikom primjerenog korištenja umjetne inteligencije na Fakultetu elektrotehnike i računarstva, te da umjetna inteligencija nije zamijenila moje kritičko razmišljanje niti moj doprinos.

Skraćenice

SPH	*Smoothed Particle Hydrodynamics*	metoda zaglađenih čestica
PIC	Particle-In-Cell	metoda čestica u ćeliji

Privitak

Implementaciju je moguće pronaći na poveznici:

https://github.com/MarinLovrinovic/FluidSim