SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 2114

SIMULACIJA FLUIDA I INTERAKCIJA S OBJEKTIMA

Marin Banožić

Zagreb, lipanj 2025.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 2114

SIMULACIJA FLUIDA I INTERAKCIJA S OBJEKTIMA

Marin Banožić

Zagreb, lipanj 2025.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 3. ožujka 2025.

ZAVRŠNI ZADATAK br. 2114

Pristupnik:	**Marin Banožić (0036530551)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentorica:	prof. dr. sc. Željka Mihajlović

Zadatak: **Simulacija fluida i interakcija s objektima**

Opis zadatka:

Proučiti osnove fizikalnog ponašanja fluida. Razraditi problematiku simulacije fluida u grafičkom programskom pogonu Unity. Posebice obratiti pažnju na interakciju fluida s drugim objektima. Omogućiti interaktivan rad korisnika u navedenom okruženju. Načiniti testiranje i usporedbu na nizu primjera. Analizirati i ocijeniti ostvarene rezultate. Diskutirati upotrebljivost ostvarenih rezultata kao i moguća proširenja. Izraditi odgovarajući programski proizvod. Rezultate rada načiniti dostupne putem Interneta. Radu priložiti algoritme, izvorne kodove i rezultate uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 23. lipnja 2025.

Sadržaj

Uvod

Simulacija fluida i interakcija fluida s krutim tijelima ima široku primjenu u računalnoj grafici.

Postoji više načina na koje se mogu simulirati fluidi, a svaki od njih ima svoje prednosti i mane. Razvijeniji modeli mogu precizno simulirati gotovo sve fizičke pojave, no takve simulacije dolaze uz znatne vremenske ustupke.

Cilj ovog rada je prikazati simulaciju fluida i objekata u stvarnom vremenu, stoga neke pojave nije moguće prikazati u potpunosti. Uz simulaciju, cilj je napraviti i grafičko korisničko sučelje za interaktivan rad u okruženju.

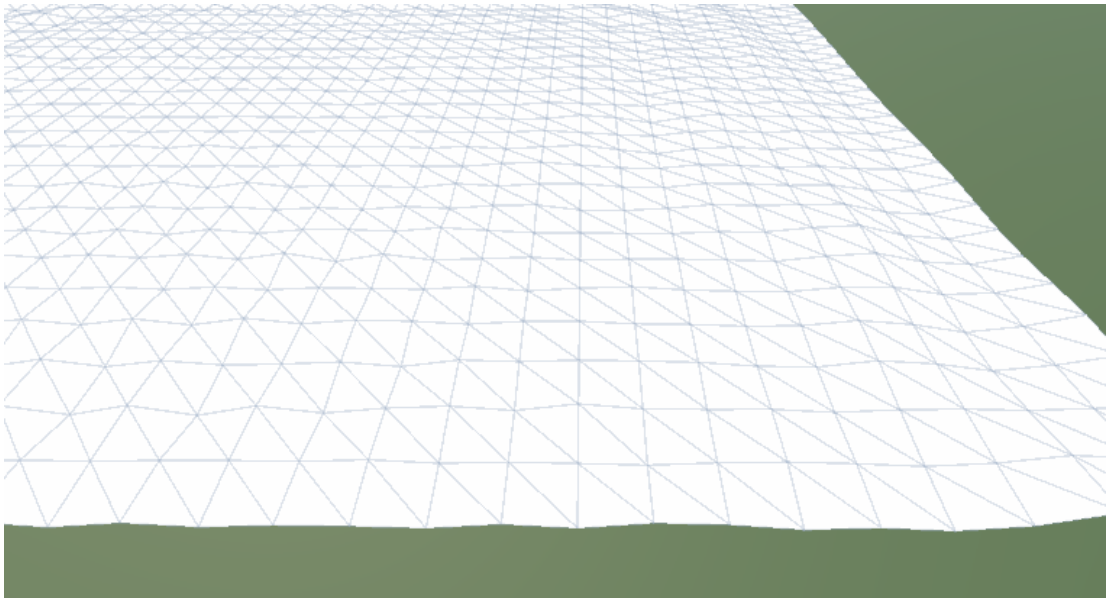
Rad je napravljen u grafičkom pogonu Unity. Unity je alat koji omogućuje izradu velikog broja aplikacija, od video igara do animacija i realističnih simulacija za gotovo sve platforme. Aplikacije mogu biti 2D ili 3D, a u ovom radu korišteno je 3D okruženje.

Kroz ovaj rad ću opisati rad simulacije fluida i interakciju s objektima te priložiti odgovarajući kod u programskom jeziku C# za grafički pogon Unity. Na kraju ću prikazati ostvarene rezultate i performanse, napraviti analizu i opisati upotrebljivost rezultata te moguća proširenja.

1. Simulacija fluida

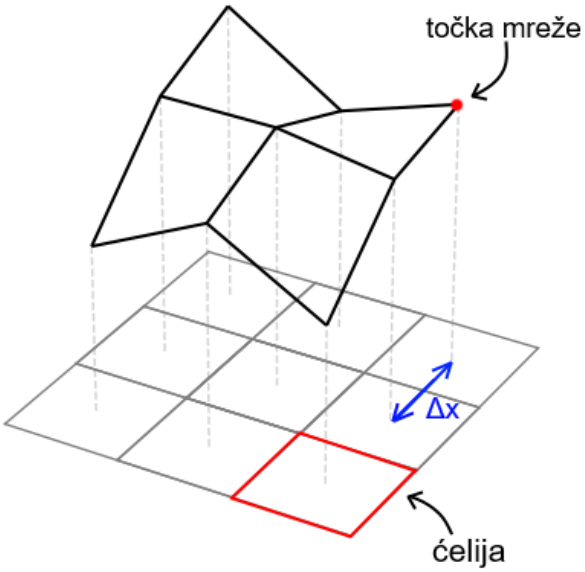
1.1. Metoda visinske mape

Jedna od najjednostavnijih metoda simulacije fluida je korištenje visinske mape (*Heightfield* ili *Heightmap*). U ovom načinu simulacije simulira se površina fluida prikazana kao 2D mreža jednako udaljenih točaka u kojoj se jedino mijenjaju visine točaka (Slika 1.1). Ovim pristupom gube se neke fizičke pojave kao što su lomljenje valova, mjehuri zraka ispod fluida, vodopadi itd., no izračun te brzina simulacije i prikazivanja su velika prednost ovog pristupa.



Slika 1.1 – Izgled mreže visinske mape

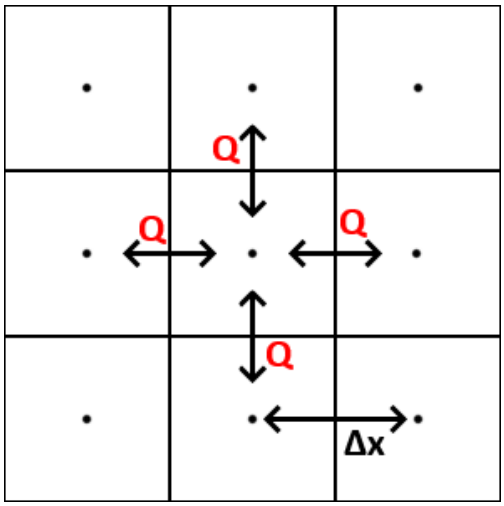
Površina fluida koja se sastoji od mreže točaka (vrhova) podijeljena je na ćelije jednakih veličina (Slika 1.2). Svaka ćelija predstavlja jedan stupac fluida. Sredina svake ćelije nalazi se na svakoj točki mreže, pa je prema tome dužina i širina svake ćelije jednaka udaljenosti dviju susjednih točaka mreže. Udaljenost između susjednih ćelija označuje se s Δx.



Slika 1.2 – 3D pogled na mrežu 3x3 točaka

Svaka ćelija ima do četiri susjedne ćelije i četiri cijevi, po jednu prema svakoj susjednoj ćeliji.

Između svakog stupca fluida, može se zamisliti cijev koja u svakom ažuriranju prenosi količinu fluida u jednom smjeru (prema trenutačnoj ćeliji, ili izvan trenutačne ćelije, tj. prema susjednoj ćeliji). Prenesena količina fluida se zatim zbraja u svakoj ćeliji i prema tome se visina ćelije (točka u sredini ćelije) mijenja. Količina fluida u cijevima označuje se s Q (Slika 1.3). [1]



Slika 1.3 – 2D pogled na ćelije 3x3 odozgo

Simulacija površine fluida se događa u dva koraka: ažuriranje količine toka fluida u cijevima i ažuriranje visina svake točke u mreži.

1.1.1. Metoda UpdateFlow

Promjena toka (*flow, Q*) u cijevi između dvije susjedne ćelije računa se prema formuli u izrazu (1).

$$Q += A\frac{g}{\Delta x}\Delta h \Delta t \tag{1}$$

Pri čemu A predstavlja poprečni presjek cijevi (tj. površinu strane između dva stupca fluida), g gravitaciju u simulaciji, Δx udaljenost između susjednih ćelija, Δh razliku između visina ćelija, a Δt vremenski razmak od zadnjeg ažuriranja.

Prilikom ažuriranja toka u metodi (Kod 1.1), koristi se prigušivanje (*damping*) kako bi se simulacija s vremenom smirila, pošto se i u stvarnosti kinetička energija u sustavu gubi s vremenom. Bez prigušivanja, simulacija bi postala savršeno elastična i valovi bi se kretali zauvijek bez gubljenja jačine.

Vrijednost *damping* je obično konstantna, no ovdje je izmijenjena u *dampingPerSecond* koji ovisi o vremenskom razmaku između prošlog i trenutačnog ažuriranja. Niže vrijednosti ove varijable puno će brže smiriti sustav, ali to može izgledati nerealistično. U mojoj simulaciji, vrijednost *damping* je obično 0.6, što dovodi do dobre ravnoteže.

```
void UpdateFlow(float dt)
{
    int numVertices = MeshResolution + 1;

    float dampingPerSecond = Mathf.Pow(1 - dampingFactor, dt);

    // Horizontal pipes
    for (int z = 0; z < numVertices; z++)
    {
        for (int x = 0; x < numVertices - 1; x++)
        {
            int cellLeftIdx = z * numVertices + x;
            int cellRightIdx = cellLeftIdx + 1;

            float hLeft = vertices[cellLeftIdx].y;
            float hRight = vertices[cellRightIdx].y;

            float A = Mathf.Max(hLeft, hRight) * dx;
            float dh = hRight - hLeft;

            horizontalPipes[x, z] += A * (Gravity / dx) * dh * dt;
            horizontalPipes[x, z] *= dampingPerSecond;
        }
    }

    // Vertical pipes
    for (int z = 0; z < numVertices - 1; z++)
    {
        for (int x = 0; x < numVertices; x++)
        {
            int cellBottomIdx = z * numVertices + x;
            int cellTopIdx = cellBottomIdx + numVertices;

            float hBottom = vertices[cellBottomIdx].y;
            float hTop = vertices[cellTopIdx].y;

            float A = Mathf.Max(hTop, hBottom) * dx;
            float dh = hTop - hBottom;

            verticalPipes[x, z] += A * (Gravity / dx) * dh * dt;
            verticalPipes[x, z] *= dampingPerSecond;
        }
    }
}
```

Kod 1.1 – Metoda *UpdateFlow*

1.1.2. Metoda UpdateHeights

Kako bi se ažurirala visina u ćeliji, potrebno je izračunati koliko fluida ulazi u, tj. izlazi iz nje. To se računa zbrajanjem toka fluida iz susjednih ćelija. Vrijednosti toka iz ćelija koje su „ispod“ ili „lijevo“ se oduzimaju jer se u metodi *UpdateFlow* vrijednosti spremaju u suprotnom smjeru.

Promjena visine (*height, h*) u ćeliji se računa formulom u izrazu (2).

$$h+= \ -\Delta t * \frac{\sum Q}{\Delta x^2} \tag{2}$$

Pri čemu $\sum Q$ predstavlja sumu toka iz susjednih ćelija, a Δx^2 površinu gornje strane ćelije.

Metoda *UpdateHeights* (Kod 1.2) ažurira visine prema formuli u izrazu (2). Metoda prima vremenski razmak Δt kao ulaz.

```
void UpdateHeights(float dt)
{
    Mesh mesh = GetComponent<MeshFilter>().mesh;

    int numVertices = MeshResolution + 1;

    for (int z = 0; z < numVertices; z++)
    {
        for (int x = 0; x < numVertices; x++)
        {
            int index = z * numVertices + x;
            float Qsum = 0;
            if (x > 0)
            {
                Qsum += horizontalPipes[x - 1, z];
            }
            if (x + 1 < numVertices)
            {
                Qsum += -horizontalPipes[x, z];
            }
            if (z > 0)
            {
                Qsum += verticalPipes[x, z - 1];
            }
            if (z + 1 < numVertices)
            {
                Qsum += -verticalPipes[x, z];
            }

            float y = vertices[index].y;
            float change = -dt * Qsum / (dx * dx);

            vertices[index].y += change;
        }
    }
    mesh.vertices = vertices;

    GetComponent<MeshFilter>().mesh = mesh;

    UpdateSidesMesh();
}
```

Kod 1.2 – Metoda *UpdateHeights*

1.2. Stabilnost simulacije

Metode *UpdateFlow* i *UpdateHeights* su dovoljne za simulaciju fluida metodom visinske mape. Kako računala ne mogu raditi s kontinuiranim vrijednostima, simulaciju dijelimo na diskretne pomake Δt. Kako bi simulacija ostala stabilna, ovi pomaci moraju biti dovoljno mali. Ako bi metodama *UpdateFlow* i *UpdateHeights* predali veliki vremenski razmak, npr. 10 sekundi, najmanje razlike u visinama bi proizvele ekstremne valove te simulacija ne bi imala smisla.

Zbog toga je potrebno prije svakog ažuriranja toka i visina izračunati maksimalan vremenski razmak za koji bi simulacija ostala stabilna.

Vrijednost stabilnog vremenskog razmaka *t* računa se formulom u izrazu (3).

$$t = 0.9 \cdot \frac{\Delta x}{g \cdot h_{max}} \tag{3}$$

Pri čemu h_{max} predstavlja visinu najviše ćelije u mreži, g gravitaciju u simulaciji, a Δx udaljenost dviju susjednih ćelija. Zbog dodatne stabilnosti, formula se množi proizvoljnim faktorom 0.9.

Konačna vrijednost vremenskog ustupka koja se predaje metodama za ažuriranje vrijednosti je manja vrijednost između stvarnog vremenskog razmaka od posljednjeg ažuriranja i izračunatog stabilnog vremenskog razmaka (Kod 1.3).

Ovime se osigurava stabilnost simulacije, čak i ako je program usporen i vremenski razmaci između ažuriranja postanu veliki.

```
var verticesHeight = vertices.Select(v => v.y);
float MaxVertexHeight = verticesHeight.Max();

float stableTimestep = 0.9f * dx / (Gravity * MaxVertexHeight);

float dt = Mathf.Min(Time.deltaTime, stableTimestep);

UpdateFlow(dt);
UpdateHeights(dt);
```

Kod 1.3 – Dio metode *FixedUpdate*

2. Interakcija fluida i objekata

Nakon simulacije fluida, potrebno je dodati simulaciju tijela. Može se koristiti Unityev ugrađeni model za kruta tijela *Rigidbody*. Ovaj model podržava osnovnu fiziku krutih tijela, kao što su gravitacija, međusobna interakciju krutih tijela itd. Uz to, sadrži neke ključne vrijednosti poput mase tijela, otpora i kutnog otpora koje se mogu mijenjati kako bi se postigao željeni rezultat.
No samo korištenje ovog modela nije dovoljno za simulaciju interakcije fluida i krutih tijela, jer se fluidi ne ponašaju kao kruta tijela.
Stoga je potrebno implementirati dodatna ponašanja koja će sva kruta tijela imati u interakciji s fluidima. U ovom radu, implementirane su dva osnovna ponašanja: sila uzgona i potiskivanje fluida. Ova ponašanja opisuje Arhimedov zakon. Arhimedov zakon glasi: Kada tijelo pluta na površini tekućine, težina mu je jednaka težini tekućine što je istisnuta onim dijelom tijela koji se nalazi ispod razine tekućine. [9]

2.1. Sila uzgona

Uzgon je sila kojom fluid djeluje na uronjeno tijelo, i djeluje suprotno smjeru gravitacije. [2]

Formula za statički uzgon F_u dana je u izrazu (4).

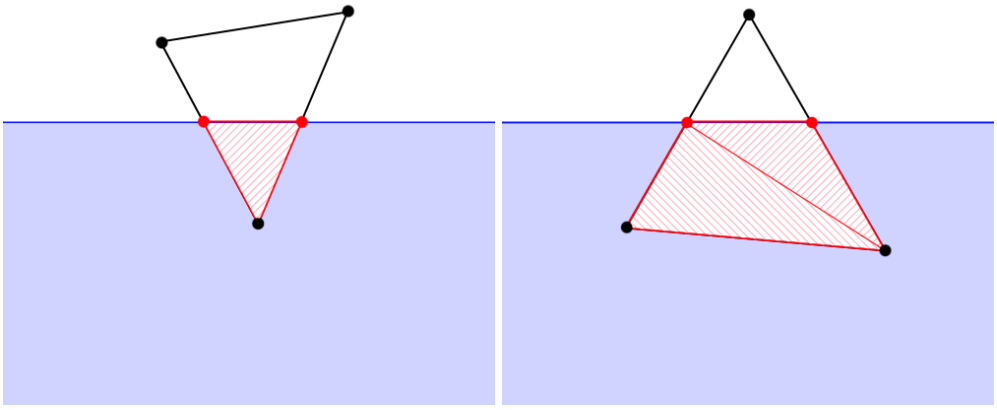
$$F_u = \rho \cdot g \cdot V \tag{4}$$

Pri čemu ρ predstavlja gustoću fluida, g gravitaciju u simulaciji, a V volumen tijela ispod površine fluida. Ukratko, sila uzgona omogućuje da tijela s gustoćom manjom od gustoće fluida plutaju, a tijela većih gustoća tonu na dno.

2.1.1. Komponenta ObjectPhysics

Svakom krutom tijelu koje se stavlja u simulaciju dodaje se komponenta *ObjectPhysics* koja tijekom svakog ažuriranja računa silu uzgona i djeluje na tijela tom silom.
Pošto je mreža tijela sastavljena od trokuta, komponenta *ObjectPhysics* za svaki trokut koji se nalazi u potpunosti ispod površine fluida računa silu uzgona. Trokuti koji su u potpunosti iznad površine fluida su zanemareni i na njih ne djeluje uzgon. Trokuti koji su djelomično ispod površine fluida moraju se najprije podijeliti na trokute koji su ili potpuno iznad površine ili potpuno ispod površine fluida. Postoje

dva takva slučaja, gdje su jedan ili dva vrha trokuta iznad površine, te se onda takav trokut dijeli na dva, odnosno jedan trokut ispod površine (Slika 2.1). [8]



Slika 2.1 – Dijeljenje trokuta koji su djelomično pod površinom

Nakon što se dobiju svi trokuti ispod površine, sila uzgona se prema formuli dodaje na sredinu svakog trokuta ispod površine fluida (Kod 2.1). Sila djeluje obrnuto od smjera gravitacije, prema gore, s pozitivnom y vrijednosti.

```
1 reference
void AddBuoyancy()
{
    foreach (Triangle tr in modify.underwaterTriangles)
    {
        Vector3 buoyancy = BuoyancyForce(tr, fluidManager.FluidDensity);
        rigidbody.AddForceAtPosition(buoyancy, tr.center);
    }
}

1 reference
private Vector3 BuoyancyForce(Triangle triangleData, float density)
{
    Vector3 buoyancyForce = density * Physics.gravity.y * triangleData.underwaterVolume;

    buoyancyForce.x = 0f;
    buoyancyForce.z = 0f;

    return buoyancyForce;
}
```

Kod 2.1 – Dodavanje sile uzgona na svaki trokut

2.2. Potiskivanje fluida

Tijela koja se nalaze ispod površine fluida (djelomično ili u cijelosti) potiskuju fluid izvan prostora koje zauzima tijelo. U ovoj simulaciji, kruta tijela ne potiskuju fluid doslovno, već samo mijenjaju tok fluida kako bi se simulirali valovi. Površina fluida (mreža) prolazi kroz kruta tijela, no to ne predstavlja znatan vizualni problem.

2.2.1. Metoda CalculateObjectDisplacement

Metoda *CalculateObjectDisplacement* (Kod 2.2) za svako tijelo u sceni računa ukupni volumen ispod površine fluida, zatim taj volumen koristi za promjenu količinu toka fluida u cijevima.

Metoda *GetSubmergedHeight* računa visinu (h) tijela u ćeliji ispod površine slanjem dviju zraka od dna prema gore te od površine fluida prema dolje.

Nakon što se izračuna visina tijela ispod površine tijela, volumen tijela u ćeliji se može izračunati formulom $V = h \cdot \Delta x^2$. Bitno je napomenuti da će za manje Δx ovaj izračun biti precizniji.

Ovdje se, kao i u metodi *UpdateFlow*, koristi prigušivanje. Tijela koja plutaju na površini fluida će se neznatno pomicati i time neprestano mijenjati količinu volumena u ćeliji (zbog npr. nakupljenih grešaka u izračunu sile uzgona, nedovoljno malog Δx itd.). Zbog toga je uvedena varijabla *velocityDamping* koja znatno prigušuje stvaranje novih valova naizgled mirnih objekata na površini fluida, a vrijednosti u izračunu same varijable dobivene su empirijski.

Prigušenje također pomaže vizualnom učinku. Naime, objekti koji padaju u fluid s visine će stvoriti veće valove, a plutajuća tijela koja gurnemo uz površinu fluida će stvoriti manje valove.

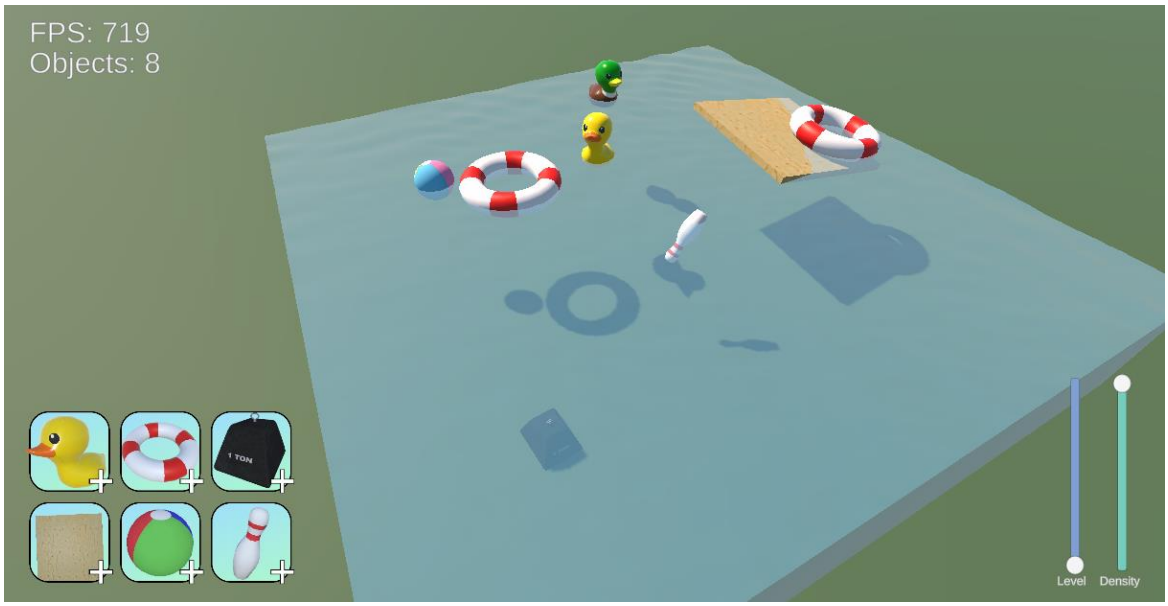
Nakon izračuna ukupnog volumena svih tijela u svakoj ćeliji, ažurira se količina toka fluida u cijevima. Ova količina se može jednostavno izračunati kao $Q = \frac{\Delta V}{\Delta t}$, a potom se jednaka količina toka fluida prenosi u svaku susjednu ćeliju.

```
private float[] prevSubmergedVolumes;

1 reference
void CalculateObjectDisplacement(float dt)
{
    int numVertices = MeshResolution + 1;

    float[] currSubmergedVolumes = new float[prevSubmergedVolumes.Length];

    foreach (GameObject obj in GameObject.FindGameObjectsWithTag("FluidObject"))
    {
        Collider collider = obj.GetComponent<Collider>();
        Rigidbody rigidbody = obj.GetComponent<Rigidbody>();
        Bounds b = obj.GetComponent<Renderer>().bounds;

        for (int z = 0; z < numVertices; z++)
        {
            for (int x = 0; x < numVertices; x++)
            {
                int index = z * numVertices + x;
                Vector3 surface = vertices[index];

                if (surface.x < b.min.x || surface.x > b.max.x || surface.z < b.min.z || surface.z > b.max.z)
                {
                    continue;
                }

                float submergedHeight = GetSubmergedHeight(collider, surface);

                if (submergedHeight > 0)
                {
                    float velocityY = rigidbody.linearVelocity.y;
                    float velocityDamping = Mathf.Clamp(Mathf.Abs(velocityY) / 5, 0.05f, 1f);

                    float submergedVolume = submergedHeight * dx * dx * velocityDamping;

                    currSubmergedVolumes[index] += submergedVolume;
                }
            }
        }
    }


    for (int z = 0; z < numVertices; z++)
    {
        for (int x = 0; x < numVertices; x++)
        {
            int index = z * numVertices + x;

            float dV = currSubmergedVolumes[index] - prevSubmergedVolumes[index];
            if (Mathf.Abs(dV) > 0)
            {
                float flowChange = dV / dt;

                if (x > 0)
                {
                    horizontalPipes[x - 1, z] += flowChange * 0.25f;
                }
                if (x + 1 < numVertices)
                {
                    horizontalPipes[x, z] -= flowChange * 0.25f;
                }

                if (z > 0)
                {
                    verticalPipes[x, z - 1] += flowChange * 0.25f;
                }
                if (z + 1 < numVertices)
                {
                    verticalPipes[x, z] -= flowChange * 0.25f;
                }
            }
        }
    }

    prevSubmergedVolumes = currSubmergedVolumes;
```

Kod 2.2 – Metoda za računanje potiskivanje fluida

3. Korisničko sučelje

Za izradu korisničkog sučelja korišten je *Canvas* objekt i *EventSystem* za obradu unosa. U *Canvas* komponenti korišteni su UI elementi poput *Text* (tekst), *Button* (gumb) i *Slider* (klizač).



Slika 3.1 – Prikaz korisničkog sučelja u programu

Na slici 3.1 se mogu vidjeti elementi korisničkog sučelja. U donjem lijevom kutu se mogu stvoriti novi objekti koji padaju u fluid. U donjem desnom kutu pomicanjem klizača se može mijenjati visina, odnosno razina fluida (*Level*) i gustoća fluida (*Density*). U gornjem lijevom kutu može se vidjeti FPS simulacije i broj tijela u sceni.

Lijevim i desnim klikom tijela u fluidu se mogu pomicati, i može se promatrati sila uzgona i valovi nastali potiskivanjem fluida.

Kamera se može pomicati tipkama W, A, S i D te rotirati držanjem desnog klika i pomicanjem miša.

Tijela u simulaciji su besplatni resursi (modeli i teksture) iz trgovine Unity Asset Store. [3][4][5][6][7]
Ikone u donjem lijevom kutu te ikona programa su izrađene u grafičkom alatu Paint.NET.

4. Rezultati

4.1. Performanse

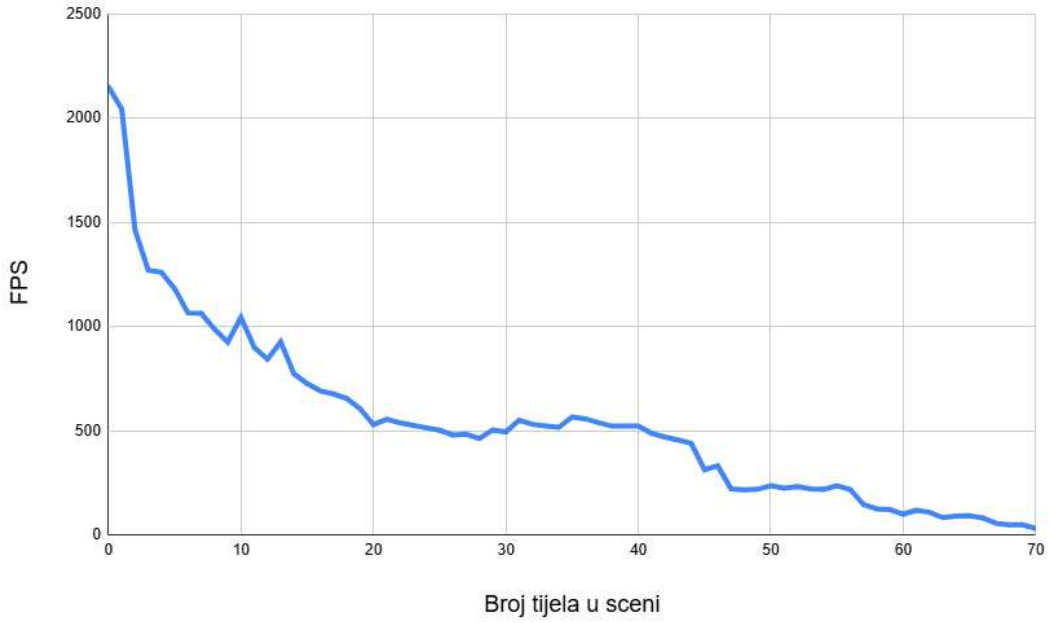
Komponente računala korištenog za testiranje:

- CPU: AMD Ryzen 5600x @ 3,70 GHz
- GPU: AMD Radeon RX 6600
- RAM: 32 GB @ 3200 MHz
- OS: Windows 10

Korištena je mreža veličine 64x64 točaka.

Bez ikakvih objekata, mirna simulacija fluida se izvršava u 2000 FPS. Za očekivati je da će FPS postepeno padati dodavanjem novih tijela, jer se za svaki objekt treba računati nekoliko sila, te njihova međusobna interakcija, kao i promjene u mreži fluida.

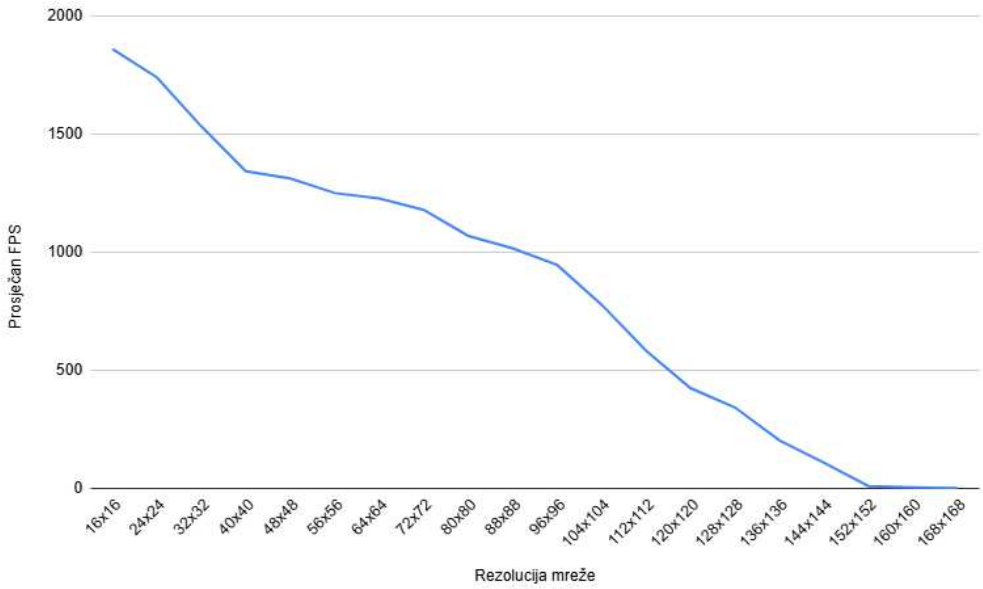
FPS ovisi o veličini tijela kao i o broju trokuta svakog tijela. Veća tijela će zauzimati više prostora i zbog toga će se na puno više vrhova mreže morati računati potiskivanje. Tijela s više trokuta su zahtjevna zbog podjele trokuta u komponenti *ObjectPyhsics*. Poželjno je koristiti tijela sa što manje trokuta.



Graf 4.1 – Graf odnosa FPS-a i broja tijela, u mreži rezolucije 64x64

Iz grafa 4.1. je vidljivo da FPS pada kako dodajemo tijela. Ovo je i očekivano, jer veći broj simuliranih tijela zahtjeva veći broj izračuna. Nakon 100 tijela, simulacija znatno usporava.

Slična rezultat možemo dobiti mijenjanjem rezolucije mreže. Za očekivati je da će se mreže veće rezolucije izvoditi s manje FPS-a, jer simulacija mora računati više nad svakim tijelom, te prikazivati na kvadrat više točaka mreže.

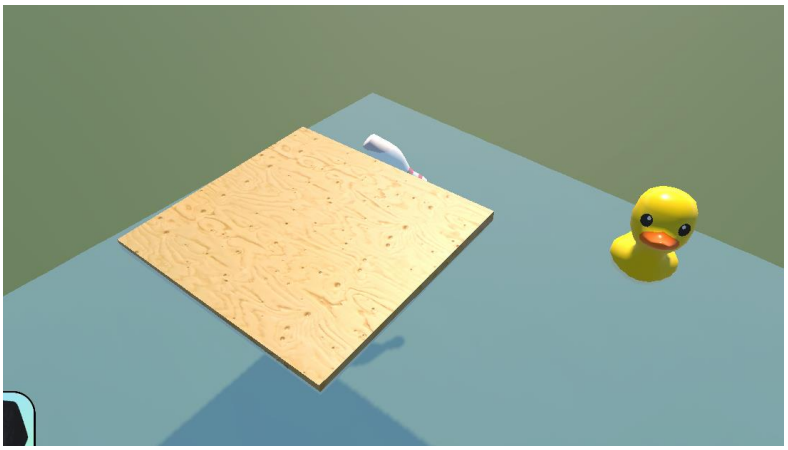


Graf 4.2 – Graf odnosa FPS-a i rezolucije mreže, s 25 tijela

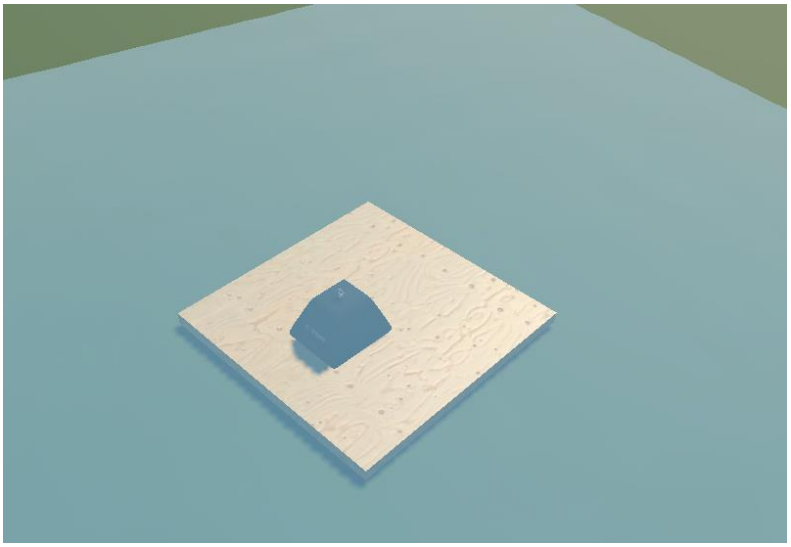
Iz grafa 4.2. je vidljivo da FPS pada kako povećavamo rezoluciju mreže. U svakoj simulaciji korišteno je istih 25 tijela. Rezolucije mreže do 128x128 (oko 15000 točaka) su upotrebljive, a nakon toga simulacija znatno usporava.

4.2. Analiza i primjenjivost rezultata

Pokretanjem programa možemo proučiti kako se ponašaju razna tijela. Gustoća fluida postavljena je na oko 1000 kg/m^3, što odgovara gustoći vode. Pritiskom na tijela male gustoće – npr. gumena patka, čunj i slično, može se primijetiti kako će ova tijela plutati na površini vode (Slika 4.2). Tijela velikih gustoća (npr. uteg) će tonuti (Slika 4.3). Također se mogu primijetiti valovi koji nastaju potiskivanjem vode.

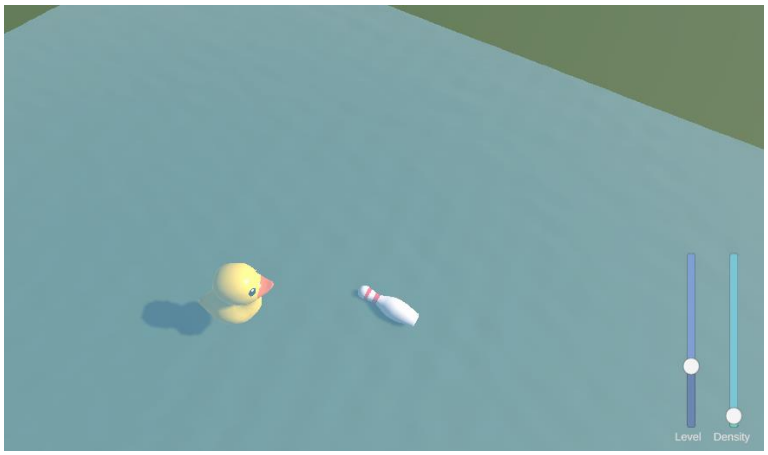


Slika 4.2 – Plutajuća tijela



Slika 4.3 – Potonuli uteg drži dasku ispod površine

Smanjivanjem gustoće fluida na oko 200 kg/m^3, na slici 4.4 može se uočiti kako plutajuća tijela počinju tonuti, što odgovara formulama.



Slika 4.4 – Smanjivanjem gustoće fluida, tijela tonu

Iz prikazanih performansi se može vidjeti da je ova simulacija prikladna za manji broj dodanih tijela, oko 50, te manjom rezolucijom mreže, oko 64x64 točaka.

Jedna od primjena ove simulacije je u video igrama gdje je bitna jedino površina fluida. Mogu se simulirati valovi te jednostavne interakcije s objektima poput brodova i drugih plutajućih tijela. Simulacija je prikladna i namijenjena za manje količine vode poput bazena, no moguće je simulirati i druge tipove toka poput rijeka ili oceana.

5. Moguća proširenja

Metoda visinske mape nije savršeni model za simulaciju fluida jer ne može simulirati sve fizičke pojave. Takve pojave bi se mogle dodatno zasebno simulirati korištenjem drugih metoda. Ako nam je potrebna veća preciznost, cijeli fluid se može modelirati sustavom čestica (npr. *smoothed particle hydrodynamics)* koji će točnije simulirati odnose između čestica fluida i krutih tijela, gdje bi tijela stvarno potiskivala čestice.

Vizualno poboljšanje fluida možemo dobiti tako da se zadatak prikazivanja (*renderanja*) mreže prebaci na grafičku karticu s većom rezolucijom mreže. Tada se za izračun fizike (interakcija fluida i tijela) na procesoru može koristiti manja mreža, npr. veličine 48x48. Ove dvije simulacije bi radile odvojeno, ali bi bile sinkronizirane.

Zadatak izračuna sile uzgona i potiskivanja fluida bi se mogao prebaciti na više dretvi kako bi simulacija podržavala nekoliko stotina krutih tijela od jednom.

Također, uporabom raznih sjenčara (*shadera*) može se dobiti bolji izgled fluida, kao i realistične sjene, pruge na dnu uzrokovane refrakcijom fluida itd.

Zaključak

Interakcija fluida s objektima je bitan problem sa širokom primjenom u računalnoj grafici. Ovaj rad prikazuje dio mogućnosti simulacije. Fluidi su simulirani metodom visinske mape koja ima prednost jednostavne implementacije i velike brzine izvođenja. Interakcija s objektima je postignuta uz pomoć modela Rigidbody te implementaciju sile uzgona i potiskivanja vode. Ostale sile se mogu simulirati zasebno. Performanse simulacije su zadovoljavajuće za manji broj objekata i odgovarajućom veličinom mreže.

Literatura

[1] Kellomäki, T. Large-Scale Water Simulation in Games. Doktorski rad. Tampere University of Technology, 2015. https://trepo.tuni.fi//handle/10024/115052, pristupljeno 27. travnja 2025.

[2] Wikipedia, Uzgon. https://hr.wikipedia.org/wiki/Uzgon

[3] Samer Khatib, *Super Rubber Duck Pack,* Unity Asset Store, 2021. https://assetstore.unity.com/packages/3d/props/super-rubber-duck-pack-34781

[4] Loafbrr, *Inflatables*, Unity Asset Store, 2025. https://assetstore.unity.com/packages/3d/props/inflatables-316990

[5] Samer Khatib, *Super Beach Pack*, Unity Asset Store, 2021. https://assetstore.unity.com/packages/3d/props/exterior/super-beach-pack-39084

[6] Egor Ilin, *Wood Products*, Unity Asset Store, 2018. https://assetstore.unity.com/packages/3d/characters/wood-products-80094

[7] Happy Zig Games, LLC, *Cartoon Heavy Weights*, Unity Asset Store, 2016. https://assetstore.unity.com/packages/3d/props/weapons/cartoon-heavy-weights-2857

[8] Erik Nordeus, *Make a realistic boat in Unity with C#*, 2020. https://www.habrador.com/tutorials/unity-boat-tutorial/3-buoyancy/, pristupljeno 12. svibnja 2025.

[9] Wikipedia, Arhimedov zakon. https://hr.wikipedia.org/wiki/Arhimedov_zakon

Sažetak

U ovom radu obrađena je teorija simulacije fluida korištenjem visinske mape. Dodana je podrška za kruta tijela te je simulirana sila uzgona i potiskivanje fluida. Za izradu rada korišten je alat Unity. Kreirano je korisničko sučelje za interaktivan rad u simulaciji. Zatim su prikazane implementacije metoda i njihov opis u programskom jeziku C#. Opisane su performanse simulacije te su objašnjena moguća proširenja.

Ključne riječi: visinska mapa; simulacija fluida; interakcija fluida i krutih tijela; računalna grafika; interaktivno sučelje

Summary

This final paper described fluid simulation using heightmaps. Support was added for rigid bodies which have buoyancy force and create fluid displacement. The program was made in game engine Unity. There is an user interface for interactive play in the simulation. This paper then shows method implementations and their description in programming language C#. It describes simulation performance and possible additions and expansions to the simulation.

Keywords: heightmap; fluid simulation; fluid-object interaction; computer graphics; interactive user interface

Privitak

Izvorni kod i datoteke rada, kao i izvršna datoteka programa za operacijski sustav Windows dostupne su na Github repozitoriju: https://github.com/774marin/Fluidi. Upute za korištenje i demonstracija se također mogu pronaći na Github repozitoriju.