# Elastični i proceduralni objekti, NPR tehnika, odšumljavanje slike te WebGPU i big data
# Tehnička dokumentacija
**Verzija 1.0**


**Studentski tim:**

Hana Ujčić

Dominik Šarić

Davor Najev

Oleksandr Ichenskyi

Fran Androić

Zoa Horvat

Marin Lovrinović

**Nastavnik:** Željka Mihajlović

# Sadržaj

# Tehnička dokumentacija

# 1. Opis razvijenog proizvoda

U projektu će se razmatrati problematika izrade elastičnih tijela. Primjena proceduralnih načina generiranja objekata u Houdiniju. Bit će razrađen problem uklanjanja šuma u iscrtanim slikama praćenjem puta i mogućnosti korištenja WebGPU tehnologije. Nadalje, proučit će se tehnika NPR sjenčanja kroz razvoj programskog rješenja za iscrtavanje i prikaz osjenčanih modela, koristeći grafičko sučelje Vulkan. Problematika vizualizacija velike količine podataka. Također, razmatrat će se problematika prikaza presjeka stošca i ravnine te mogućnosti njegovog preciznijeg i vizualno jasnijeg prikaza u 3D okruženju. Posebna pažnja bit će posvećena povezivanju interaktivnih prikaza s matematičkim zadacima za vježbu u sklopu web stranice.

# 2. Tehničke značajke

## 2.1. Elastična tijela

U sklopu projekta izgrađena je aplikacija koja simulira elastično tijelo metodom dinamike temeljene na poziciji (eng. Position Based Dynamics – PBD). Ova metoda je prvi put razvijena u [7], a često se koristi za simulacije tkanine i mekih tijela u video igrama i drugim aplikacijama koje zahtijevaju izvođenje u stvarnom vremenu, kao i naprednija inačica XPBD (Extended Position Based Dynamics) razvijena u [8]. Aplikacija je izrađena u C++ i koristi sučelja OpenGL, GLFW i Imgui.

PBD je metoda simulacije fizike koja izračunava gibanje izravnim nametanjem ograničenja na položaje čestica, umjesto integriranjem sila. Ovaj pristup je bezuvjetno stabilan, jednostavan za upravljanje i dobro je prilagođen za primjene u stvarnom vremenu. Pseudokod algoritma PBD je prikazan na slici ispod.

**Algorithm 1** Position-Based Dynamics Algorithm

**for all** vertices $i$ **do**
  initialize $\mathbf{x}_i = \mathbf{x}_i^0$, $\mathbf{v}_i = \mathbf{v}_i^0$, $w_i = 1/m_i$
**end for**
generateConstraints()
**loop**
  **for all** vertices $i$ **do**
    $\mathbf{v}_i \leftarrow \mathbf{v}_i + \Delta t\, w_i \mathbf{f}_{\text{ext}}(\mathbf{x}_i)$
    $\mathbf{p}_i \leftarrow \mathbf{x}_i + \Delta t\, \mathbf{v}_i$
  **end for**
  **for** $k = 1$ to solverIterations **do**
    projectConstraints($C_1, \ldots, C_{M+M_{\text{coll}}}, \mathbf{p}_1, \ldots, \mathbf{p}_N$)
  **end for**
  **for all** vertices $i$ **do**
    $\mathbf{v}_i \leftarrow (\mathbf{p}_i - \mathbf{x}_i)/\Delta t$
    $\mathbf{x}_i \leftarrow \mathbf{p}_i$
  **end for**
**end loop**

Slika 2.1.1.: Algoritam dinamike temeljene na poziciji

Aplikacija sadrži klasu PBD, koja pri inicijalizaciji prima vrhove i strane tijela, te inicijalizira elastični objekt kao vektor čestica (linije 1-3). Svaki vrh tijela je predstavljen kao čestica strukturom Particle koja sadrži poziciju, brzinu, masu i vanjsku silu. Također se postavljaju ograničenja koja su ostvarena u klasi Constraint (linija 4). Ograničenja kontroliraju elastično tijelo tako da se može slobodno micati i sudarati s drugim objektima, pritom pazeći da ne prođu kroz drugo tijelo i da se vrate u svoj originalni oblik. U ovoj je aplikaciji implementirano nekoliko unutarnjih ograničenja i ograničenje sudara. Ograničenja udaljenosti, podudaranja oblika i

očuvanja volumena su unutarnja ograničenja i određuju kako se čestice unutar objekta mogu kretati jedna u odnosu na drugu. Oni određuju materijalno ponašanje i oblik objekta. Ograničenja sudara određuju interakcije između objekta i njegove okoline te sprječavaju prodiranje čestica kroz prepreke poput tla. Karakteristike elastičnog tijela (čestice i ograničenja) su implementirani u klasi SimulatedObject radi preglednosti, a klasa PBD sadrži instancu SimulatedObject.

Glavna petlja se izvodi u funkciji step() klase PBD. Za svaki vremenski korak izračunaju se nove pozicije i brzine primjenom vanjskih sila, poput gravitacije (linije 6-9), te se zatim pozicije ispravljaju iterativnim projiciranjem ograničenja (linije 10-12). Nakon što se pronađu ispravljeni položaji, brzine se ažuriraju na temelju promjene položaja (linije 13-16). Klasa main programa inicijalizira klasu PBD i poziva funkciju step(), te ažurirane pozicije iscrtava u prozoru zajedno s menijem.

Meni s postavkama je implementiran s ImGui. U sučelju se mogu podesiti parametri PBD algoritma: veličina vremenskog koraka i broj koraka za solver. Također se mogu podesiti parametri elastičnog objekta, tj. krutost unutarnjih ograničenja. Jedina vanjska sila koja se primjenjuje je gravitacija, koja se isto može podesiti. Simulacija se pokreće i zaustavlja pritiskom na gumb Run Simulation/Stop Simulation, a može se ponovo pokrenuti iz početnog položaja pritiskom na gumb Reset Simulation.

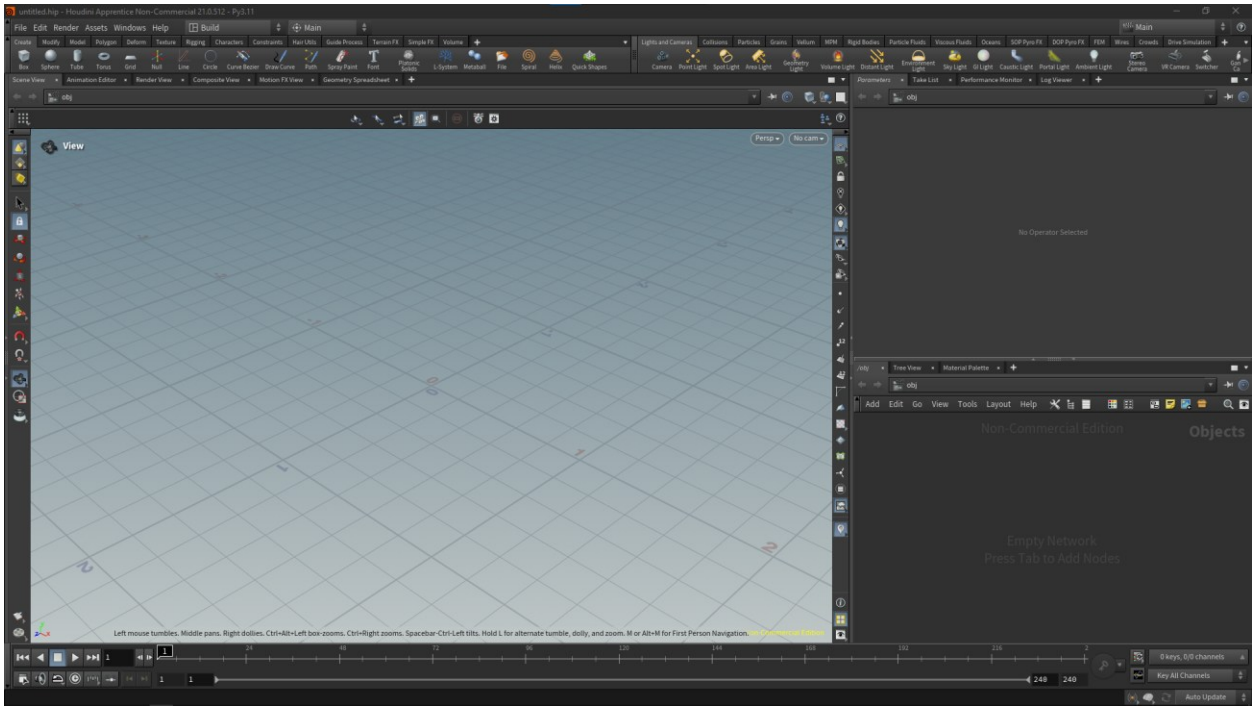
Slika 2.1.2.: Prikaz aplikacije

## 2.2. Proceduralna generacija modela koristeći alat Houdini

Houdini [11] (Slika 2.2.1) je napredni proceduralni alat za 3D modeliranje, animaciju i

vizualne efekte koji se koristi za izradu, obradu i automatizaciju složenih geometrijskih i simulacijskih procesa. Za razliku od tradicionalnih 3D alata, Houdini se temelji na nodalnom (node-based) pristupu, što omogućuje visoku razinu kontrole, ponovljivosti i fleksibilnosti u razvoju digitalnih sadržaja.

U kontekstu ovog projekta, Houdini služi kao ključni alat za generiranje proceduralnih elemenata. Takav pristup omogućuje brže iteracije, jednostavne izmjene dizajna i efikasno prilagođavanje projektnih zahtjeva bez potrebe za ručnom rekonstrukcijom sadržaja. Korištenje Houdinija doprinosi većoj tehničkoj preciznosti, skalabilnosti rješenja i boljoj integraciji s ostalim alatima i tehnologijama uključenima u projektni proces.



Slika 2.2.1 Korisničko sučelje programa Houdini

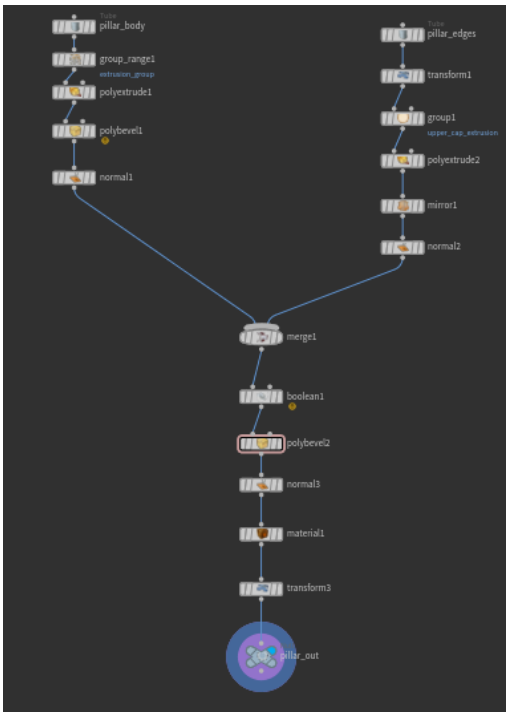
U svrhe demonstracije realizirana su 3 podprojekta proceduralnog modeliranja:

1. Starogrčki stupovi
2. Planinski snježni tereni
3. Kaktusi porodice “Angel Wings”

Kako bi bolje objasnili način rada u Houdiniju fokusirati ćemo se samo na jedan od ova tri podprojekta (starogrčke stupove) jer je primarno shvatiti način rada samog programa i osnove

tipove čvorova (nodes) [12]. Nakon savladavanja osnovnih pojmova ostatak proučavanja Houdinija se svodi na razmotavanje široke lepeze čvorova.

Kao što je moguće i vidjeti na slici 2.2.2, gotovo svaka operacija u Houdiniju se svodi na postavljanje čvorova i njihovog međusobnog povezivanja u radnom okviru. Pored samih čvorova potrebno je odabrati i njihove parametre kako bi se dobio željen rezultat (npr. postavljanje željenog broja stupaca u čvoru "pillar_body").

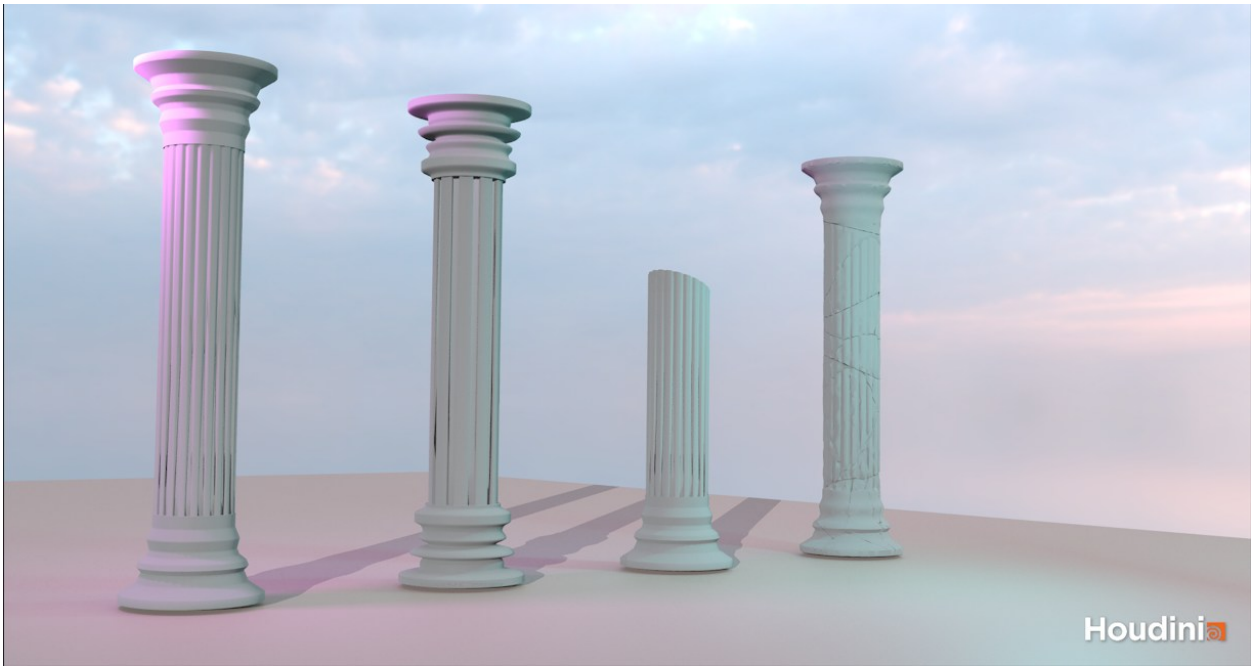



Slika 2.2.2 "Node-based" pristup modeliranja

Samo modeliranje stupova se sastoji od dvije komponente: trupa stupa i baze stupa. Modeliranje ove dvije komponente moguće je vidjeti na gornjoj polovici slike, gdje je na lijevoj strani prikazano modeliranje trupa, a na desnoj baze.
Kako bi se izmodelirao finalni izgled trupa korišteni su sljedeći čvorovi kronološki: Tube (za dobivanje cilindričnog oblika stupa) , Group Range (za selektiranje svake druge strane cilindra), Poly Extrude i Poly Bevel (ekstruzija i zakošenje svake od tih strana), te Normal (kako bi se ispravili artefakti pri sjenčanju).

Za modeliranje baza stupa korišten je sličan pristup s dodatkom: Transform čvora (za translaciju s obzirom na trup stupa) i Mirror čvora (za zrcaljenje baze po ordinati). Također, za svaki od ovih čvorova korišteni su proizvoljni parametri kako bi se postigao željeni oblik.

Nakon modeliranja pojedinih dijelova, cjelokupni model dobivamo korištenjem Merge čvora kojim ih spajamo. U ovome dijelu modeliranja moguće je dodavati razne čvorove kako bi se postigle finalne transformacije na cijeli stup (kao što su boolean operacije, materijali, frakture, dodatna zakošenja itd.) kao što je i prikazano finalnim rezultatom na slici 2.2.3.




Slika 2.2.3. Proceduralno generirani starogrčki stupovi

## 2.3. Uklanjanje šuma slikama renderiranim path tracingom

Implementacija uklanjača šuma je napravljena uz pomoć **dubokih neuronskih mreža**. Duboko učenje je jedan od podskupova strojnog učenja. Ideja pristupa problemu je koristiti neuronske mreže kao univerzalni funkcijski aproksimator s puno parametara koji se uče iz podataka. Konkretno u kontekstu uklanjača šuma, duboka neuronska mreža bi na ulazu primala zašumljenu sliku, a na izlazu bi izbacila aditivni šum kojeg moramo odbiti od slike kako bi dobili odšumljenu sliku.

Kao i kod svakog algoritma strojnog učenja, moramo definirati sljedeće

1) Podatke za učenje
2) Model (u ovom slučaju konkretnu arhitekturu neuronske mreže koje ćemo koristiti)
3) Način treniranja

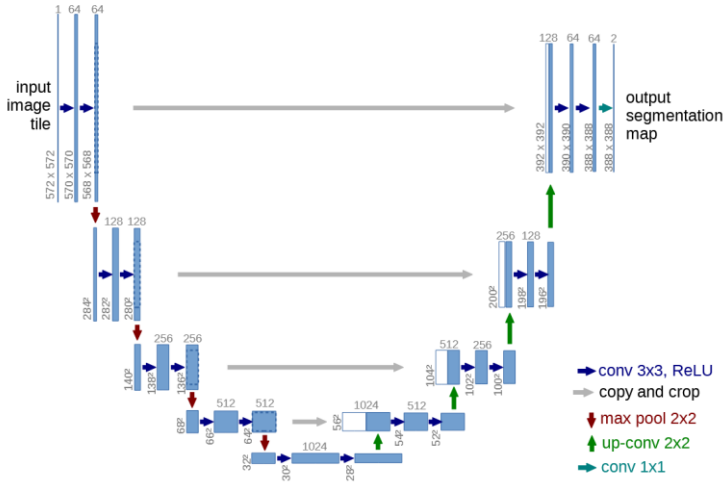
Kako bi dobili podatke za učenje, korišten je program Mitsuba 3 koji nudi mogućnosti skriptiranog pathtracing renderiranja i višekanalne izlaze. Mitsuba renderer je postavljen da

renderira Cornell box scenu i da izbaci sljedeće izlaze:

- Renderirana slika kvalitete 1 SPP (Sample Per Pixel) - zašumljena slika
- Renderirana slika kvalitete 1024 SPP (Sample Per Pixel) - visokokvalitetna slika bez primjetnog šuma
- Albedo - prikaz renderiran deterministički bez sjena
- Dubina - dubinski spremnik koji sprema dubinu od kamere do predmeta
- Normalna mapa u svakoj točki

Izlazi su svi veličine 1024x1024.

Kao model, korištena je neuronska mreža **U-Net**. Njena arhitektura je prikazana na slici 2.3.1.




Slika 2.3.1: U-Net arhitektura

Ona prima trodimenzionalni ulaz (visina, širina, broj kanala), transformira ga tako da mu progresivno smanjuje prostornu dimenzionalnost (visina, širina) i povećava dimenziju značajki (kanali). U sredini mreže je ulaz pretvoren u reprezentaciju veličine (1, 1, 1024), odnosno visokodimenzionalni vektor značajki. Taj vektor zovemo **latentni vektor** i on ustvari enkodira naš ulaz u međureprezentaciju koju dekodiramo desnim dijelom mreže natrag u prostorne dimenzije ulaza. Ovakvu arhitekturu prema tome nazivamo **autoenkoderskom** zbog kombinacije enkodera-dekora te **konvolucijskom** jer koristi konvolucijske slojeve koji su logičan izbor za obradu slike. Kod implementacije uklanjača šuma, arhitektura će izgledati na sljedeći način:

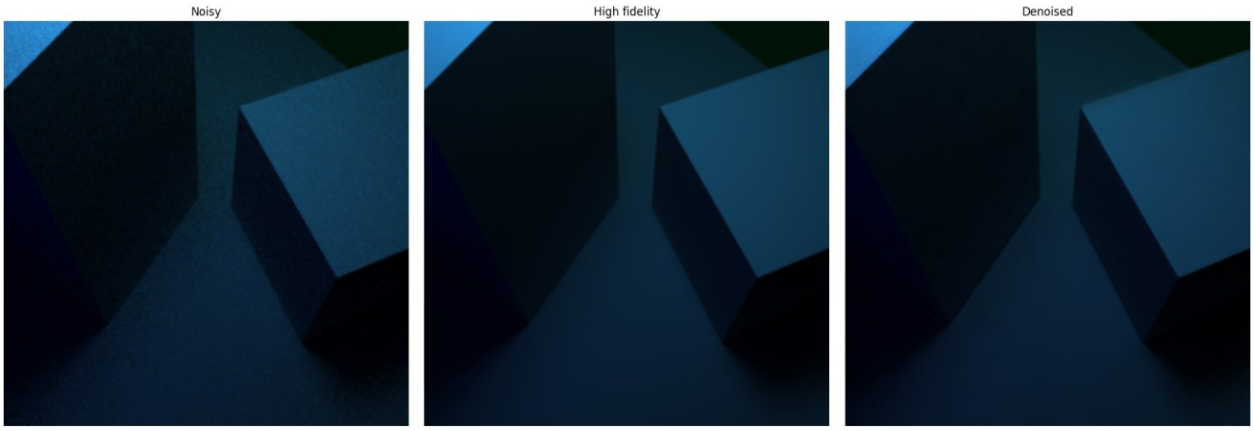
- ulaz: tenzor veličine (256, 256, 10) koji se sastoji od slika renderiranih pomoću Mitsuba 3 smanjenih na veličinu (256, 256) zbog bržeg učenja, a kanali su sljedeći: r, g, b kanal slike sa šumom, r, g, b albedo kanali, kanal dubinskog spremnika te x, y, z kanali normala što sve skupa čini 10 kanala

- izlaz: tenzor veličine (256, 256, 3) koji se sastoji od r, g, b kanala odšumljene slike.

Način treniranja je sljedeći: gradijentni spust sa stopom učenja 1e-4, regularizacijom 1e-5 na 1000 slika u 100 epoha. Kao funkciju gubitka korišten je L1 gubitak (apsolutna udaljenost). Moguće je još koristiti L2 gubitak (euklidska udaljenost), ali dalje zamućene slike što nipošto ne želimo.

Petlja za treniranje je napisana u programskom jeziku Python u razvojnom okviru PyTorch.

Primjer rezultata je prikazan na slici 2.3.2.

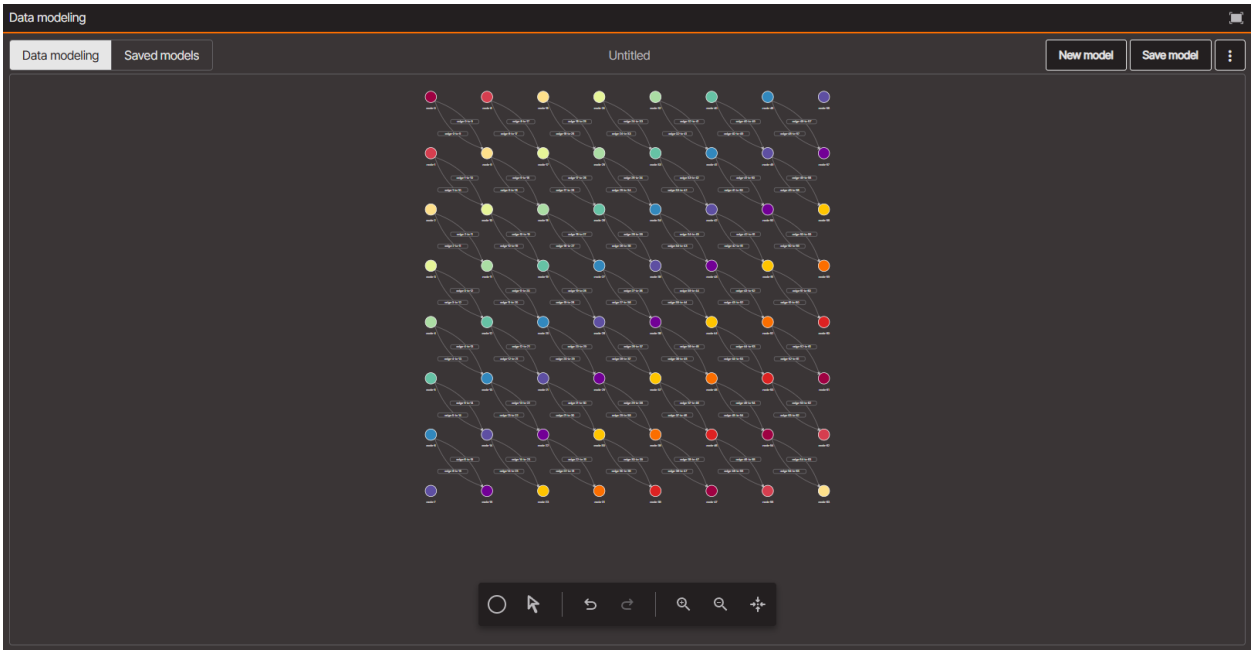



Slika 2.3.2: Primjer šumovite slike, slike bez šuma te odšumljene slike

## 2.4. Vizualizacija velike količine podataka

Cilj ovog istraživanja bio je evaluirati optimalne metode vizualizacije velikih grafova u web preglednicima kao preduvjet za potencijalnu implementaciju u WebGL-u. U tu svrhu provedena je usporedba dviju osnovnih tehnologija vizualizacije: SVG-a (Scalable Vector Graphics) i HTML5 Canvasa.

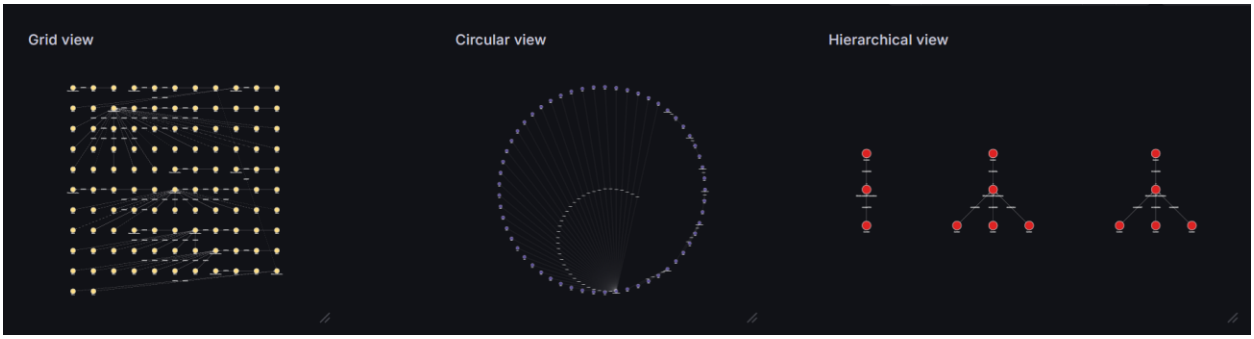
Za potrebe mjerenja performansi SVG tehnologije, razvijena je interaktivna web aplikacija (modul za Memgraph Lab) koja omogućuje CRUD operacije (kreiranje, uređivanje, ažuriranje, brisanje) nad čvorovima i vezama. Kako bi se omogućilo testiranje na velikim skupovima podataka, implementirana je funkcionalnost uvoza i izvoza u više JSON formata. To je omogućilo integraciju s Python skriptom, također razvijenom tijekom istraživanja, koja služi za automatizirano uvoženje i vizualizaciju grafova. Performanse su mjerene korištenjem Node.js modula performance. Aplikacija podržava i osnovnu stilizaciju (promjena boje i veličine čvorova) kako bi se simulirala kompleksnost stvarnih podataka.




Slika 2.4.1. - Prikaz manjeg skupa podataka sa definiranim stilovima u SVG okruženju

Za evaluaciju Canvas tehnologije korištena je open-source biblioteka Orb, koja je u sklopu ovog projekta značajno proširena. Kako bi se omogućila dinamička promjena svojstava čvorova i dizajna tijekom mjerenja performansi, implementiran je oblikovni obrazac Promatrač (Listener). Dodatno, postojeći sustav za vizualizaciju i pripadajući GSS (Graph Style Script) sustav za stiliziranje prošireni su primjenom oblikovnog obrasca Strategija. Time je, uz postojeći force-directed raspored (koji koristi D3 fizikalni pogon), omogućena podrška za hijerarhijski, cirkularni i mrežni (grid) raspored čvorova, što je olakšalo testiranje različitih vizualizacijskih scenarija.

Inicijalna verifikacija provedena je unutar aplikacije Memgraph Lab (izgrađene na Angularu). Mjerenja su pokazala neočekivane rezultate: Canvas vizualizacija bila je sporija od SVG-a, što je kontradiktorno tehničkim specifikacijama Canvasa koji nema dodatno opterećenje DOM stabla (DOM overhead) karakteristično za SVG. Analizom je utvrđeno da je uzrok loših performansi dodatno računalno opterećenje same Angular aplikacije i njezinih komponenti. Zbog toga su, u svrhu objektivnijeg mjerenja, kreirane dvije izolirane testne okoline. Prva okolina realizirana je kao jednostavna, sintetička HTML stranica s minimalnim dodatnim opterećenjem koja se izravno spaja na graf bazu i vizualizira podatke, te je upravo na njoj provedena većina testova. Drugo okruženje temelji se na integraciji s alatom Grafana, gdje se podaci dohvaćaju korištenjem Neo4j datasource dodatka iz bilo koje graf baze, dok se sama vizualizacija odvija putem prilagođenog panela Graforb, razvijenog specifično u sklopu ovog projekta i koji omogućava promjenu većina parametara i stilova vizualizacija pomoću pripadajućeg korisničkog sučelja.




Slika 2.4.2. - Prikaz više načina rasporeda čvorova pomoću Graforb plugina

Za testiranje skalabilnosti korišteni su sintetički podaci generirani posebnim Cypher upitima, te Hetionet skup podataka (deseci tisuća čvorova i milijuni veza) za testiranje rubnih slučajeva.

**Time to paint measures in milliseconds (Canvas)**

| # of Edges / # of Nodes | 10 | 100 | 1000 | 10000 |
|---|---|---|---|---|
| 0 | 3 | 9 | 82 | ~800 |
| 5 | 3 | 10 | ~100 | ~1000 |
| 10 | 3 | ~50 | ~300 | ~17000 |

**Time to paint measures in milliseconds (SVG)**

| # of Edges / # of Nodes | 10 | 100 | 1000 | 10000 |
|---|---|---|---|---|
| 0 | 7 | 30 | 650 | ~28000 |
| 5 | 10 | ~230 | 7000 | N/A |
| 10 | ~20 | ~500 | 27000 | N/A |

Slika 2.4.3. - Prikaz rezultata mjerenja performansi za SVG i Canvas okruženja u milisekundama

Rezultati su potvrdili očekivanja: iako je Canvas znatno brži i skalabilniji od SVG-a, prikazi s više od nekoliko tisuća čvorova rezultiraju drastičnim padom performansi i lošim korisničkim iskustvom. Kvaliteta interakcije dodatno opada uvođenjem velikog broja veza, što je ključno za analizu grafova. Zaključno, za skalabilnu vizualizaciju masivnih grafova nužan je prelazak na WebGL ili WebGPU tehnologije koje mogu iskoristiti hardversku akceleraciju grafičke kartice.

## 2.5. NPR

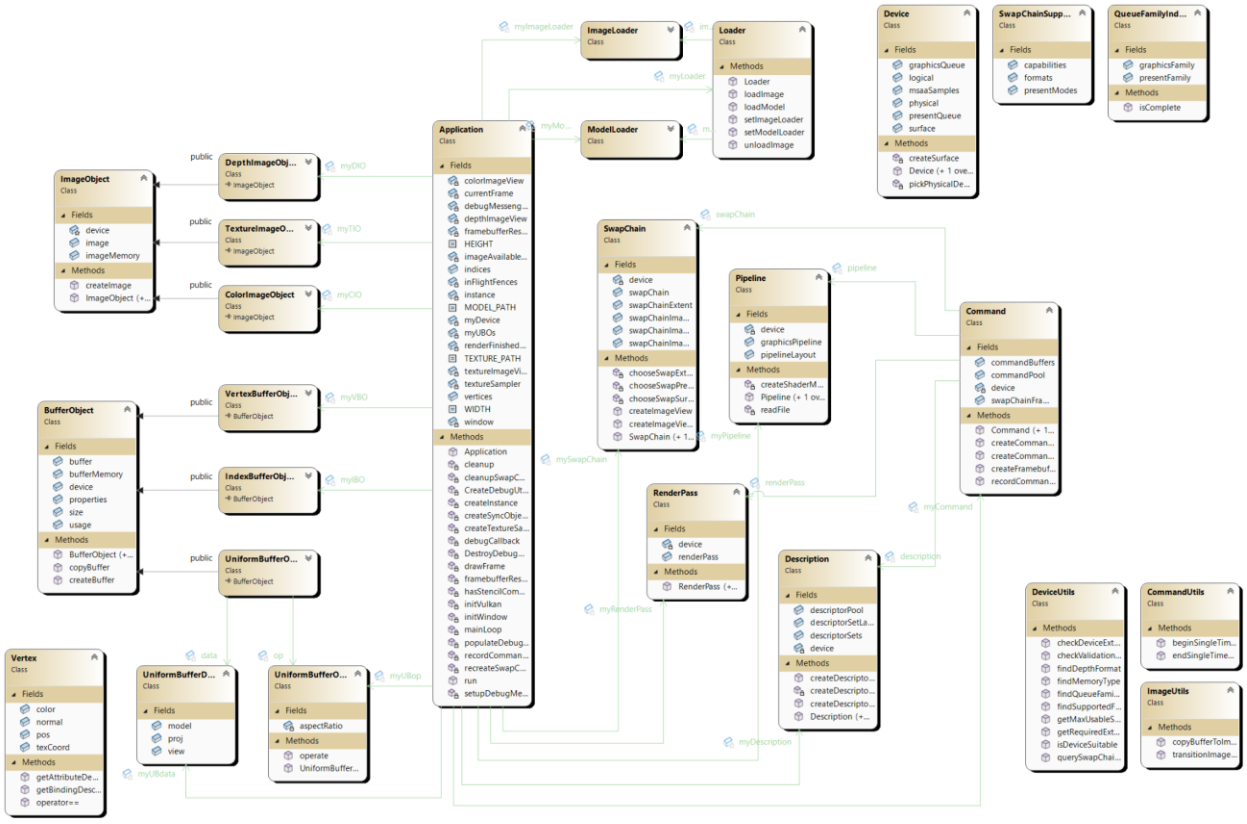
Aplikacija izrađena za projekt zamišljena je kao sloj apstrakcije iznad funkcionalnosti koje

pruža Vulkanovo sučelje. Baza aplikacije je implementacija Khronosove Vulkan specifikacije koju je razvio LunarG [1], a izrađena je po uzoru na popularni online tutorijal [2]. Specifikacija pruža mnoge funkcije i razrede kojima je moguće pristupiti GPU memoriji i definirati kako ju iskoristiti za iscrtavanje, koje su prepoznatljive po prefiksu "vk". Krajnji cilj aplikacije je omogućiti intuitivnu izradu i vizualnu provjeru sjenčara (engl. shader), s naglaskom na podršku za nefotorealistične (NPR) sjenčare. Rješenje je i dalje u izradi, no ideja je apstrahirati sve pozive "vk" funkcija i pristupe "vk" objektima kako korisnik ne bi morao razmišljati o stvarima "niske razine" kao što je pristup memoriji, a zadržati veliku razinu fleksibilnosti i mogućnosti podešavanja.

Središnji razred aplikacije je Application koji trenutno služi kao kontrolni razred koji se brine za stvaranje i uništavanje svih ostalih objekata, a zadužen je i za "vk" elemente poput validacijskog sloja i sinkronizacijskih objekata koji nisu apstrahirani. U njemu se također nalazi i funkcija zadužena za iscrtavanje na ekran. Iscrtavanje se u osnovi provodi definiranjem i zapisivanjem naredbe, te njezinim predavanjem u red za iscrtavanje. Sva funkcionalnost vezana za zapis naredbe enkapsulirana je unutar razreda Command. Njegova funkcija za iscrtavanje poziva "vk" naredbu za iscrtavanje, a za to koristi informaciju iz razreda SwapChain, RenderPass, Pipeline i Description. SwapChain sadrži informacije o iscrtanim slikama kao što su format boje i veličina (rezolucija). Description je zadužen za definiranje podataka koji će se proslijediti sjenčarima u jedinstvenom (engl. uniform) obliku, dakle jednaki za svaki istovremeno pokrenuti sjenčar te ga uz Command koristi i Pipeline. RenderPass opisuje kakve sve slike će biti krajnji proizvod jednog prolaska kroz grafički cjevovod. Trenutno je postavljen tako da se pri jednom prolazu iscrta jedna naduzorkovana slika u boji (zbog "anti-aliasinga"), jedan spremnik dubine (za prikaz 3D objekata) i jedna konačna slika u boji koja se prikazuje na ekranu. Pipeline opisuje sami prolaz kroz grafički cjevovod, sa svim njegovim fazama. Također je zadužen za učitavanje sjenčarskih programa i njihovo postavljanje za izvođenje u pravoj fazi, kao i pridodavanje pravih opisnika koje sjenčari koriste prilikom izvođenja te definiranje pravih slika iz RenderPassa u koje treba upisati određene podatke. Razred BufferObject napravljen je kako bi enkapsulirao strukture i metode potrebne za rad sa spremnicima, koji se u ovom kontekstu koriste kako bi se u njih spremali podaci o 3D modelima i teksturama koje se prikazuju. Osmišljen je kao apstraktni razred kojeg nasljeđuju VertexBufferObject, IndexBufferObject i UniformBufferObject (Koristi UniformBufferData i UniformBufferOperator za konkretne operacije nad konkretnim podacima. Trenutno su implementirane transformacijske matrice i operacija rotacije oko y-osi, no ovi razredi su osmišljeni s oblikovnim obrascem "strategija" na umu.) s konkretnim implementacijama za željeni tip podataka. Na sličan način ostvaren je i apstraktni razred ImageObject i njegove implementacije DepthImageObject, TextureImageObject i ColorImageObject.

Postoje još neke potporne strukture, od kojih valja spomenuti DeviceUtils, CommandUtils te ImageUtils koji su zamišljeni kao skupovi alata ostvarenih statičkim funkcijama koje druge klase mogu iskoristiti za ponovljeno ostvarenje često potrebne funkcionalnosti ili dobivanje kakve informacije od sučelja. Naposljetku, postoji razred Loader koji drži objekte tipa ImageLoader i ModelLoader i koristi se za svo potrebno učitavanje podataka. Ostvarena je jedna implementacija ImageLoader razreda koja koristi vanjsku knjižnicu stb [3], te jedna implementacija ModelLoader

razreda koja koristi vanjsku knjižnicu tinyobjloader [4]. Razredni UML dijagram cijele aplikacije prikazan je na slici 2.5.1.




Slika 2.5.1. - Razredni UML dijagram aplikacije za NPR sjenčanje

Uz samu aplikaciju, napisan je i par sjenčara po uzoru na 2.5.1: sjenčar vrhova (engl. vertex shader) i sjenčar fragmenata (engl. fragment shader). Oni služe kao primjer mogućnosti prikaza NPR sjenčara koristeći trenutnu inačicu aplikacije. Njihov GLSL kod prikazan je na slici 2.5.2.

```
#version 450

layout(binding = 0) uniform UniformBufferObject {
    mat4 model;
    mat4 view;
    mat4 proj;
} ubo;

layout(location = 0) out vec3 fragColor;
layout(location = 1) out vec2 fragTexCoord;
layout(location = 2) out vec3 fragNormal;

layout(location = 0) in vec3 inPosition;
layout(location = 1) in vec3 inColor;
layout(location = 2) in vec2 inTexCoord;
layout(location = 3) in vec3 inNormal;

vec3 lightPos = vec3(2.0, -2.0, 2.0);
vec3 cameraPos = vec3(0.0, -2.0, 0.0);

void main() {
    vec4 vPos = ubo.model * vec4(inPosition, 1.0);
    gl_Position = ubo.proj * ubo.view * vPos;
    fragColor = inColor;
    fragNormal = inNormal;

    float normalLightAngle = max(0.0, dot(normalize(lightPos - vec3(vPos)), inNormal));
    float normalCameraAngle = max(0.0, dot(normalize(cameraPos - vec3(vPos)), inNormal));
    fragTexCoord = vec2(normalLightAngle, normalCameraAngle);
}
```

```
#version 450

layout(binding = 1) uniform sampler2D texSampler;

layout(location = 0) in vec3 fragColor;
layout(location = 1) in vec2 fragTexCoord;
layout(location = 2) in vec3 fragNormal;

layout(location = 0) out vec4 outColor;

void main() {

    outColor = vec4(fragColor * texture(texSampler, fragTexCoord).rgb, 1.0);
}
```

Slika 2.5.2: GLSL kod primjera NPR sjenčara

## 2.6. Stožac i ravnina – interaktivni prikaz i matematički zadatci

Aplikacija izrađena za projekt zamišljena je kao interaktivni 3D prikaz presjeka stošca i ravnine unutar web preglednika. Glavna svrha aplikacije je omogućiti vizualno jasno razumijevanje geometrije presjeka te korisniku pružiti mogućnost jednostavnog upravljanja ravninom i promatranja rezultata u realnom vremenu. Aplikacija koristi Three.js library za 3D renderiranje i temelji se na modelu dvostrukog stošca (gornji i donji dio) kako bi se omogućio prikaz svih vrsta presjeka: kružnice, elipse, parabole i hiperbole. Ravnina je definirana kao geometrijski objekt (PlaneGeometry) koji se može pomaknuti i rotirati, a njezin položaj i orijentacija određuju tip presjeka.

Aplikacija se sastoji od HTML sučelja koje sadrži tri slidera za upravljanje ravninom te canvas element za prikaz 3D scene, kao i JavaScript modula `mainStozac.js` koji implementira logiku inicijalizacije scene, izračuna presjeka i prikaza krivulje. U HTML dokumentu nalaze se tri kontrolna elementa koji omogućuju pomak ravnine gore–dolje (Y os), pomak ravnine lijevo–desno (X os) te nagib ravnine (rotacija oko Y osi). Svaka promjena vrijednosti slidera poziva funkciju `updatePlane()`, koja ponovno izračunava jednadžbu ravnine i presjek te crta novu krivulju.

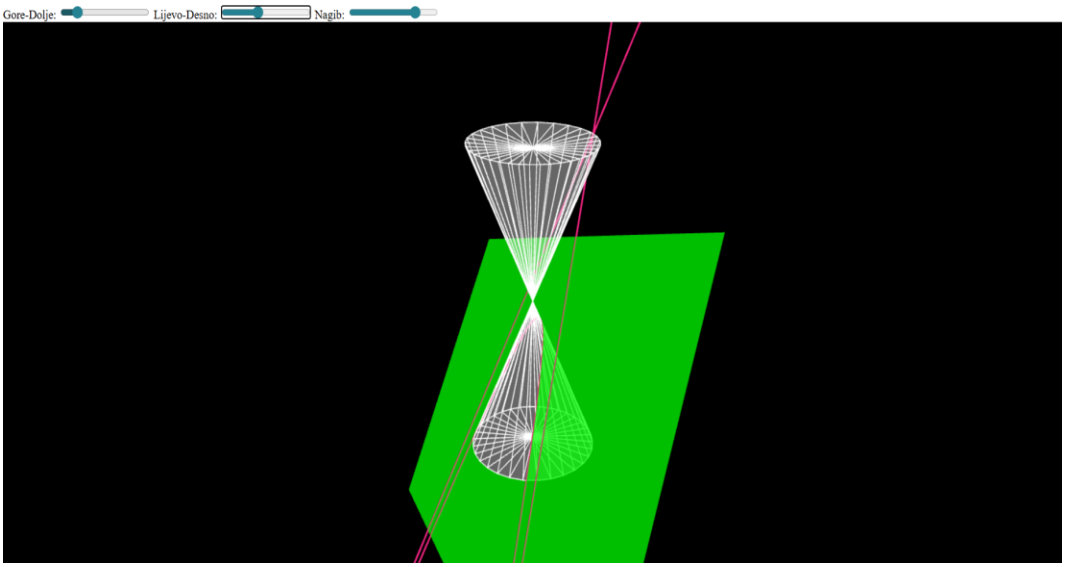
Stožac se u sceni prikazuje kao spoj dva stošca okrenuta vrhovima jedno prema drugome, čime se omogućuje prikaz hiperboličnog presjeka. Takva konstrukcija omogućuje prikaz svih vrsta presjeka stošca i ravnine. Stožac je prikazan kombinacijom prozirnog materijala za tijelo i wireframe materijala za rubove, čime se postiže bolja vizualna preglednost unutarnje strukture.

Ravnina je postavljena ispod stošca i prikazana prozirnim materijalom kako bi presjek bio jasno vidljiv. Kamera je perspektivna i postavljena je tako da promatra scenu iz udaljenosti, a korisnik može rotirati kameru pomoću miša pritiskom tipke „r“. Rotacija se može zaustaviti pritiskom tipke „s“, što omogućuje pregled presjeka iz različitih kutova.

Matematički model presjeka temelji se na jednadžbi stošca i jednadžbi ravnine. Stožac je definiran jednadžbom $x^2 + z^2 = k^2y^2$ , gdje je *k* nagib stošca, odnosno omjer radijusa i visine. Ravnina je definirana jednadžbom $Ax + By + Cz + D = 0$, pri čemu se koeficijenti dobivaju iz normalnog vektora ravnine i njezine pozicije u svjetskim koordinatama. Presjek se izračunava iterativno za kut $\Phi$ u rasponu od 0 do $2\pi$, pri čemu se za svaki $\Phi$ iz jednadžbe ravnine izračunava vrijednost *y*, a zatim se određuju koordinate *x* i *z*. Dobivene točke se potom transformiraju u lokalni koordinatni sustav ravnine i spajaju u liniju koja predstavlja presjek.

Implementacija aplikacije temelji se na nekoliko ključnih funkcija. Funkcija `getPlaneEquation()` izračunava koeficijente ravnine (A, B, C, D) te normalni vektor ravnine. Funkcija `getDoubleConeEquation()` računa nagib stošca *k* na temelju njegovog radijusa i visine. Funkcija `createPoints(k)` generira niz točaka presjeka koje se prosljeđuju funkciji `createCurveFromPoints()`, zaduženoj za crtanje presjeka korištenjem `Line2`, `LineGeometry` i `LineMaterial` objekata. Prije crtanja nove krivulje, prethodna se uklanja kako bi se izbjeglo preklapanje. Funkcija `calcAngle(t)` pretvara vrijednost slidera u kut rotacije ravnine te, ovisno o tom kutu, određuje tip presjeka i boju krivulje. Na taj način moguće je vizualno razlikovati kružnicu, elipsu, parabolu i hiperbolu.

Korisnička interakcija temelji se na korištenju slidera i kontrole kamere. Svaka promjena položaja ili nagiba ravnine automatski se reflektira na presjek, što korisniku omogućuje promatranje promjena u realnom vremenu. Aplikacija je zamišljena kao temelj za integraciju u web stranicu s matematičkim zadacima, gdje se interaktivni prikaz može koristiti kao vizualna podrška pri učenju i rješavanju zadataka. U budućnosti je moguće proširiti aplikaciju dodavanjem više ravnina i presjeka, automatskog prikaza fokusnih točaka i osi presjeka, integracije zadataka s automatskom provjerom rješenja te animacije pomicanja ravnine i dinamičke promjene presjeka.




Slika 2.6.1: Prikaz aplikacije s vidljivim stošcem, ravninom i presjekom.

U sklopu ove cjeline osmišljen je i skup matematičkih zadataka vezanih uz analitičku geometriju u trodimenzionalnom prostoru, s naglaskom na ravnine i odnose između točaka i ravnine. Zadatci su zamišljeni kao dio edukativne cjeline koja se sastoji od teorijske podloge, vizualnog prikaza u 3D okruženju te provjere znanja kroz samostalno rješavanje zadataka.

Cilj zadataka je omogućiti korisnicima utvrđivanje teorijskih pojmova vezanih uz jednadžbu ravnine i pripadnost točaka ravnini. Zadatci su oblikovani tako da nadopunjuju vizualni dio aplikacije, ali su konceptualno neovisni o samom načinu interakcije s prikazom.

Prvi tip zadatka odnosi se na određivanje jednadžbe ravnine na temelju triju zadanih točaka u prostoru. Korisnik treba iz koordinata zadanih točaka izvesti jednadžbu ravnine u općem obliku.

Drugi tip zadatka bavi se provjerom pripadnosti više točaka istoj ravnini. U ovom slučaju zadane su četiri točke u prostoru, a zadatak je ispitati pripadaju li sve točke istoj ravnini.

Treći tip zadatka odnosi se na provjeru pripadnosti zadanih točaka ravnini čija je jednadžba poznata. Korisniku su zadane koordinate točaka i jednadžba ravnine, a zadatak je odrediti pripadaju li zadane točke toj ravnini.

Osmišljeni zadatci služe kao potpora razumijevanju gradiva iz analitičke geometrije te omogućuju dodatnu provjeru znanja nakon upoznavanja s teorijskim pojmovima i njihovom vizualnom interpretacijom u trodimenzionalnom prostoru.

## 2.7.Simulacija fluida metodom SPH

U sklopu projekta razvija se aplikacija koja simulira fluide koristeći metodu SPH (Smoothed Particle Hydrodynamics). SPH je Lagrangeova metoda simulacije tekućina temeljena na česticama, koja se često koristi u računalnoj grafici za simulaciju vode, krvi, eksplozija i drugih fluidnih efekta. Metoda je prvi put predstavljena u radovima Gingold i Monaghan (1977.) [13] te Lucy (1977.) [14] za astrofizičke simulacije, a kasnije je prilagođena za računalnu grafiku. Aplikacija je izrađena u C++-u i OpenGL-u.

SPH metoda predstavlja fluid kao skup čestica koje međusobno utječu jedna na drugu. Za razliku od Eulerovih metoda koje koriste fiksnu mrežu, SPH je Lagrangeova metoda u kojoj se čestice slobodno kreću kroz prostor. Svaka čestica nosi fizikalna svojstva poput mase, gustoće, brzine i tlaka. Ključna ideja SPH metode je aproksimacija kontinuiranih polja pomoću funkcije jezgre (kernel function) koja omogućava izračun svojstava u bilo kojoj točki prostora kao ponderirani prosjek susjednih čestica.

Funkcija jezgre W(r, h) je tipično radijalno simetrična funkcija koja opada s udaljenošću, gdje je r udaljenost između dviju čestica, a h radijus utjecaja (smoothing length). Najpopularnije jezgre uključuju polinomijalnu jezgru, kubičnu spline jezgru i Gaussovu jezgru. Kubična spline jezgra najčešće se koristi zbog dobre ravnoteže između preciznosti i računalne učinkovitosti.

Bilo koje fizikalno svojstvo A u poziciji r može se aproksimirati kao:

$$A(r) \approx \Sigma_j \, m_j \, (A_j/\rho_j) \, W(r - r_j, h)$$

gdje je $m_j$ masa čestice j, $\rho_j$ gustoća čestice j, $A_j$ vrijednost svojstva A u čestici j, a suma se provodi po svim česticama unutar radijusa utjecaja h.

Navier-Stokesove jednadžbe opisuju gibanje viskoznih fluida i sadrže jednadžbu kontinuiteta i jednadžbu gibanja. U SPH formulaciji, ove jednadžbe se diskretiziraju pomoću čestica.

**Gustoća** svake čestice računa se iz masa i pozicija susjednih čestica:

$$\rho_i = \Sigma_j \, m_j \, W(r_i - r_j, h)$$

**Tlak** se najčešće računa pomoću jednadžbe stanja fluida. Za simulacije vode, koristi se Taitova jednadžba stanja koja povezuje tlak i gustoću:

$$p = B((\rho/\rho_0)^\gamma - 1)$$

gdje je $\rho_0$ gustoća u mirovanju, B je konstanta koja određuje kompresibilnost, a $\gamma$ je adiabatski

eksponent (tipično 7 za vodu).

**Sila tlaka** koja djeluje na česticu računa se kao gradijent polja tlaka:

$$F_i^{pressure} = -\Sigma_j m_j (p_i + p_j)/(2\rho_j) \nabla W(r_i - r_j, h)$$

**Viskoznost** uvodi disipaciju energije i prigušenje koje stabilizira simulaciju. Sila viskoznosti računa se kao:

$$F_i^{viscosity} = \mu \Sigma_j m_j (v_j - v_i)/\rho_j \nabla^2 W(r_i - r_j, h)$$

gdje je $\mu$ koeficijent viskoznosti, a $v_i$ i $v_j$ su brzine čestica.

Ukupna sila na česticu je suma sile tlaka, viskoznosti i vanjskih sila poput gravitacije. Ubrzanje čestice se računa prema Newtonovom zakonu, a pozicija i brzina se integriraju kroz vrijeme koristeći numeričku metodu poput Euler metode ili Leap-frog integracije.

Standardna SPH metoda ima nekoliko poznatih problema koji se mogu riješiti raznim tehnikama. **Površinska napetost** je važna za realističan izgled fluida, posebno pri simulaciji kapljica ili interakcija fluid-zrak. Implementira se dodatnom silom koja djeluje na površinske čestice i teži minimizaciji površine fluida.

**Detekcija sudara** s okruženjem je ključna za interakciju fluida s objektima u sceni. Sudari se tipično rješavaju metodom kazni (penalty method) gdje se čestice koje penetriraju u objekt guraju natrag van, ili metodom ograničenja gdje se položaj čestice direktno korigira.

**Prostorna particija** je nužna za performanse jer naivni pristup pronalaženja susjednih čestica ima složenost $O(n^2)$. Korištenje prostornih struktura poput uniformne mreže ili hash tablice smanjuje složenost na $O(n)$, što je kritično za simulacije s većim brojem čestica. U uniformnoj mreži prostor se dijeli na kubične ćelije veličine h, a svaka čestica se dodjeljuje svojoj ćeliji. Pri traženju susjeda potrebno je pregledati samo 27 susjednih ćelija.

Fluid je vizualiziran kao skup sfera. Realističniji prikaz može se postići ekstrakcijom površine fluida metodama poput Marching Cubes algoritma ili screen-space rendering tehnikama koje generiraju glatku površinu iz pozicija čestica.

# 3. Upute za korištenje

## 3.1. Elastična tijela

Instalacija i pokretanje: Preuzeti izvorni kod s https://github.com/UjcicHana/projekt_soft_bodies zatim izgraditi pomoću `cmake` i pokrenuti izvršnu datoteku.

Za minimalnu konfiguraciju treba imati instalirano C++, CMake, OpenGL, GLFW, Eigen i ImGui.

## 3.2. Proceduralna generacija modela koristeći alat Houdini

Preuzeti projektne datoteke s https://github.com/drito256/houdini_procedural_modelling, te ih otvoriti u alatu Houdini.

## 3.3. Uklanjanje šuma slikama renderiranim path tracingom

Repozitorij s projektom se nalazi na web-lokaciji https://github.com/spinzed/montecarlo-denoiser Instalacija, pokretanje i konfiguriranje se nalaze u datoteci "README.md" u korijenu projekta.

## 3.4. Vizualizacija velike količine podataka

Izvorni kod dodatka Graforb, zajedno s detaljnim uputama za instalaciju zavisnosti i pokretanje razvojne okoline, dostupan je unutar README datoteke na GitHub repozitoriju: https://github.com/AlexIchenskiy/graforb.

Aplikacija Memgraph Lab, koja sadrži implementiranu SVG vizualizaciju te inicijalnu Canvas inačicu modula, može se preuzeti sa službene stranice: https://memgraph.com/docs/memgraph-lab.

## 3.5. NPR

Instalacija i pokretanje: preuzeti posljednje izdanje (engl. release) s repozitorija https://github.com/franandroic/dipl, raspakirati arhivu i pokrenuti izvršnu datoteku.

## 3.6. Stožac i ravnina – interaktivni prikaz i matematički zadatci

Instalacija i pokretanje: Nakon preuzimanja izvornog koda s https://github.com/zoah4/diplomski, potrebno je instalirati zavisnosti pomoću `npm install`, zatim pokrenuti aplikaciju komandom `npm run dev:stozac`. Aplikacija će biti dostupna na lokalnoj adresi prikazanoj u terminalu.

## 3.7.Simulacija fluida metodom SPH

Instalacija i pokretanje: Preuzeti izvorni kod s https://github.com/MarinLovrinovic/FluidSim,

zatim izgraditi pomoću `cmake` i pokrenuti izvršnu datoteku.

## 4. Literatura


[1] LunarG VulkanSDK: https://www.lunarg.com/vulkan-sdk-and-ecosystem-tools-siggraph-2023/ , pristupljeno 26.01.2026.

[2] Vulkan Tutorial: https://vulkan-tutorial.com/Introduction, pristupljeno 26.01.2026.

[3] stb: https://github.com/nothings/stb, pristupljeno 26.01.2026.

[4] tinyobjloader: https://github.com/tinyobjloader/tinyobjloader, pristupljeno 26.01.2026.

[5] Barla, P., Thollot, J., & Markosian, L. (2006, June). X-toon: An extended toon shader. In *Proceedings of the 4th international symposium on Non-photorealistic animation and rendering* (pp. 127-132).

[6] Olaf Ronneberger, Philipp Fischer, Thomas Brox. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation: https://arxiv.org/abs/1505.04597

[7] M. Müller, B. Heidelberger, M. Hennix, J. Ratcliff (2006) Position Based Dynamics: https://matthias-research.github.io/pages/publications/posBasedDyn.pdf

[8] M. Macklin, M. Müller, N. Chentanez (2016) XPBD: Position-Based Simulation of Compliant Constrained Dynamics: https://matthias-research.github.io/pages/publications/XPBD.pdf

[9] Grafana docs: https://grafana.com/docs

[10] Memgraph docs: https://memgraph.com/docs

[11] SideFX Houdini: https://www.sidefx.com/, pristupljeno: 28.01.2026

[12] Houdini Tutorial: https://www.youtube.com/watch?v=JbxNElzALrM&list=PLOcFlKaT7TO8dmMhkDERt_JC_kdGSv3Vn, pristupljeno: 28.01.2026

[13] R. A. Gingold, J. J. Monaghan (1977) Smoothed particle hydrodynamics: theory and application to non-spherical stars: https://academic.oup.com/mnras/article/181/3/375/988212

[14] Lucy, L. B. (1977) A numerical approach to the testing of the fission hypothesis: https://ui.adsabs.harvard.edu/abs/1977AJ.....82.1013L/abstract