Simulacija tkanine

Perušić, Ani

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Preddiplomski sveučilišni studij računarstva

Završni rad

SIMULACIJA TKANINE / CLOTH SIMULATION

Rijeka, rujan 2022.

Ani Perušić
0069085291

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Preddiplomski sveučilišni studij računarstva

Završni rad

SIMULACIJA TKANINE / CLOTH SIMULATION

Mentor: izv. prof. dr. sc. Jerko Škifić

Rijeka, rujan 2022.

Ani Perušić
0069085291

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA ZAVRŠNE ISPITE

Rijeka, 12. ožujka 2021.

Zavod: **Zavod za mehaniku fluida i računarsko inženjerstvo**
Predmet: **Računalna grafika**
Grana: **2.09.06 programsko inženjerstvo**

ZADATAK ZA ZAVRŠNI RAD

Pristupnik: **Ani Perušić (0069085291)**
Studij: Preddiplomski sveučilišni studij računarstva

Zadatak: **Simulacija tkanine / Cloth simulation**

Opis zadatka:

Izraditi računalni program dinamičkog procesa savijanja tkanine pod utjecajem vanjskih sila modelom opruga i čestica. Voditi računa o geometrijskim i fizikalnim veličinama koje definiraju osobine simuliranog fleksibilnog objekta. Analizirati utjecaj diskretizacije i fizikalnih osobina objekta na konačni rezultat simulacije.

Rad mora biti napisan prema Uputama za pisanje diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Ani Perušić

Zadatak uručen pristupniku: 15. ožujka 2021.

Mentor:

Izv. prof. dr. sc. Jerko Škifić

Predsjednik povjerenstva za završni ispit:

Izv. prof. dr. sc. Kristijan Lenac

Izjava o samostalnoj izradi rada

Izjavljujem da sam samostalno izradila ovaj rad.

Rijeka, rujan 2022.

_____

Ani Perušić

Zahvala

Zahvaljujem mentoru izv. prof. dr. sc. Jerku Škifiću na podršci tijekom pisanja ovoga rada i korisnim raspravama i savjetima.

Sadržaj

Popis slika

Poglavlje 1

Uvod

Cilj ovog rada istražiti je i implementirati fizički ispravan model tkanine u programskoj podršci Blender (dalje u tekstu Blender), te njegovom aplikacijskom programskom sučelju u jeziku Python (dalje u tekstu Python API, odnosno API).

U uvodnom dijelu upoznaje se sa Blenderom i mogućnostima koje nudi korisniku u ručnom određivanju trodimenzionalnih objekata, fizike svijeta u kojima se objekti nalaze te fizike međudijelovanja tih objekata sa svijetom i ostalim objektima. Također, dotiče se Blenderovog korisničkog sučelja i pojedinih specifičnosti koje su zapažene tijekom rada u spomenutom sučelju. Na kraju poglavlja prikazuje se i odabrani način kreiranja i prikazivanja animacije koja je nastala kao produkt simulacije.

Glavni dio rada podijeljen je na dva veća poglavlja:

1. Poglavlje *Mass-spring* model (4) koje prolazi kroz odabran model za prikaz tkanine i prikazuje matematičke, analitičke i logičke korake koji dovode do određivanja ponašanja modela.
2. Poglavlje Implementacija (5) koje prikazuje implementaciju prethodno obrađenih koncepata, ideja i algoritama.

Pri odabiru modela za prikaz tkanine bilo je nužno imati dovoljno dokumentiranih ishoda korištenja takvog modela, te se pokazalo kako je odabran jedan od najkorištenijih modela zbog svojih jednostavnih elemenata i mogućnosti korištenja

algoritama poput Eulerove integracije i Verlet integracije. Ti matematički izrazi također se predstavljaju u kontekstu potreba ovog rada, te se ne ide u više detalja i izračuna nego što je nužno za razumijevanje ovog rada.

Najopširniji dio sadržan je u poglavlju Implementacija 5 kako bi bile što jasnije odluke tokom izrade rada i pisanja popratnog programskog koda. Izlažu se svi potrebni dijelovi koda, sa primjerima i zapažanjima, jedan po jedan, kako bi se cijela programska podrška mogla smisleno spojiti u cjelinu te povezati sa prethodnim poglavljem. Željeni ishod poglavlja o implementaciji jest čitatelju dati moguće početnu implementaciju u istoj ili nekoj drugoj tehnologiji, te moguća zapažanja gdje bi se sustav mogao poboljšati sa drugačijim idejama i proširenjima. Prikazani su i slikovni rezultati implementacije modela.

Dodatno, opisan je najveći problem s kojim se susrelo tokom rada, a to je problem super-elastičnosti koji se istraživanjem pokazao kao vrlo čestim u definiranju stvarne fizike za mrežaste objekte. Nažalost, nekoliko pokušaja rješavanja problema nije bilo uspješno te je jedan ostavljen u radu kao primjer mogućeg rješenja pri drugačijoj implementaciji modela.

Za kraj, izražen je zaključak na temelju svega istraženog, isprobanog i naučenog, te neke smjernice koje bi ubuduće mogle olakšati ulaz u svijet računalne grafike i početak "igranja" sa implementacijom simulacija i modela.

Poglavlje 2

Programska podrška Blender

2.1 Općenito o Blenderu

Programska podrška Blender je besplatni program za stvaranje otvorenog koda 3D prikaza. Podržava cjelokupni 3D cjevovod (eng. *pipeline*, u računarstvu definiran kao kontinuirano kretanje instrukcija prema procesoru ili u aritmetičkim koracima koje procesor poduzima da izvrši instrukciju). Pod time se podrazumijeva: modeliranje, animacija, simulacija, prikazivanje (renderiranje), sastavljanje i praćenje pokreta. Također, pruža podršku za uređivanje video uradaka i stvaranje računalnih igrica uz male napore. [4]

Blender je višeplatformska programska podrška, te podjednako radi na operacijskim sustavima Microsoft Windows, Linux i Macintosh računalima. Za grafičko sučelje koristi OpenGL kako bi iskustvo na svakoj od platformi bilo konzistentno, te se preko samoga sučelja i najčešće koristi. [4]

Kako grafička sučelja često radi jednostavnosti korištenja i intuitivnosti "sakriju" pojedine funkcionalnosti, Blender omogućava upravljanje većinom svojih mogućnosti preko aplikacijskog programskog sučelja koje koristi Python programski jezik. [4]

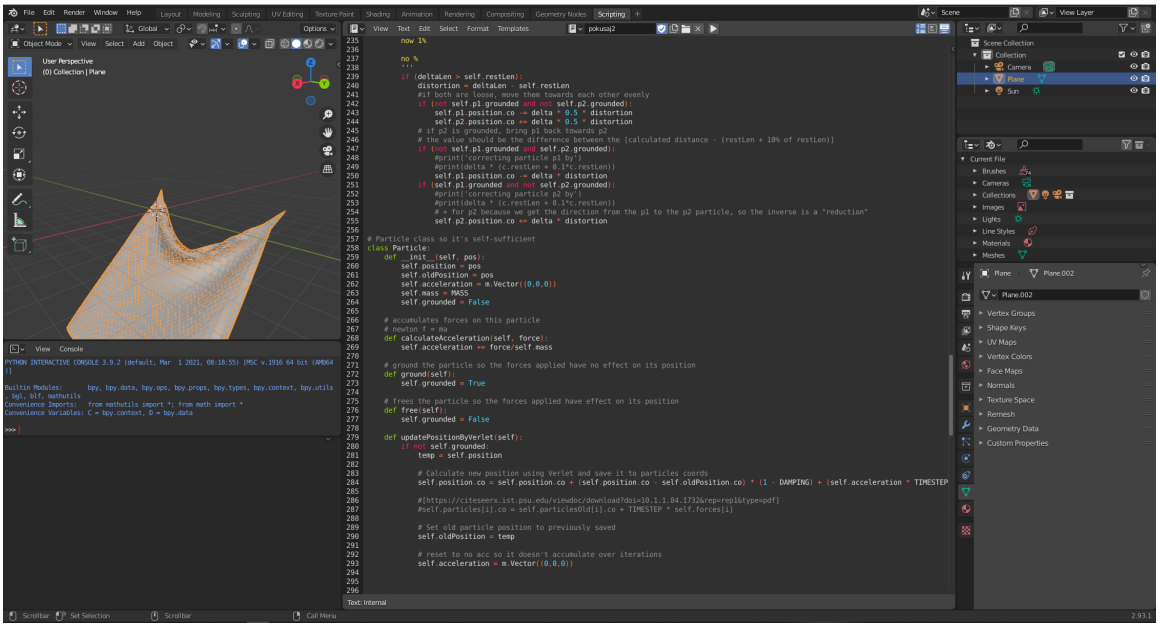
2.2 Python API

Pri izradi programskog koda ovog rada, u potpunosti je korišten Python API za postavljanje, kreiranje, simuliranje fizike i prikazivanje okvira (eng. *frame*) animacije. To je omogućeno pokrivenosti Blenderovih funkcionalnosti sa spomenutim API-jem, iako je i dalje definiran kao nedovršen od strane programera Blendera. [5]

API pruža dovoljne mogućnosti za skriptiranje simulacije u ovome radu, ali i mnoštvo drugih. Primjerice: u radu je korišteno uređivanje scene, mreže koja predstavlja objekt tkanine, manipuliranje položaja čestica te prikazivanje okvir-po-okvir (eng. *frame-by-frame*) kako bi se dobila finalna animacija. Uz to, nekorištene opcije obuhvaćaju kreiranje vlastitih alata, korištenje alata implementiranih od strane drugih korisnika, stvaranje elemenata korisničkog sučelja, mapiranje postojećih elemenata i ostale. [5]

2.3 Korisničko sučelje Blendera

Potrebno je napomenuti i specifičnosti programa, ponajviše specifičnosti korisničkog sučelja koje obuhvaća uređivač teksta (eng. *text editor*) odnosno Python skripte (koji se prikaže odabirom kartice *Scripting* na vrhu prozora), prikaz scene (eng. *Viewport*), Python konzolu za brzo testiranje linija koda, te desnu traku koja se sastoji od više izbornika koji prikazuju sve objekte na sceni te njihove parametre i postavke, vidljive na slici 2.1. Naravno, cijeli raspored elemenata može se prilagođavati prema korisnikovim željama i potrebama. Jedina nepogodnost, prema osobnom iskustvu, na koju je moguće pri korištenju korisničkog sučelja je važnost položaja pokazivača u odnosu na sve elemente sučelja. Primjerice, ako se pokazivač nalazi iznad prikaza scene, uređivač teksta neće dopustiti upisivanje iako je pokazivač u tekstu i Blender prozor je označen. To je često ometalo rad i prouzročilo "lomljenja" toka misli.



Slika 2.1 Pregled korisničkog sučelja programske podrške Blender

2.4 Prikaz animacije u Blenderu

Prikazivanje scene (renderiranje) je proces generiranja slikovnih prikaza dvodimenzionalnih ili trodimenzionalnih objekata. Blender nudi tri različita načina za renderiranje, od kojih svaki ima svoje prednosti:

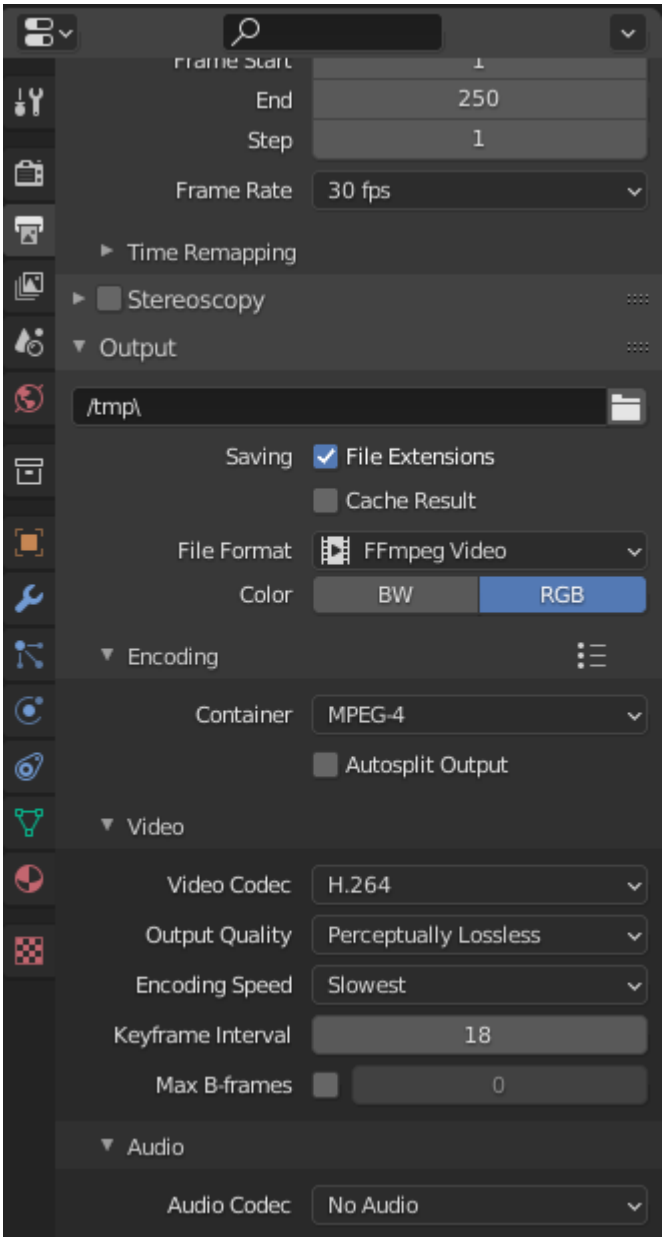
- **Eevee** je fiziči baziran renderer u stvarnome vremenu.
- **Cycles** je fizički baziran tragač puta (eng. *path tracer*).
- **Workbench** je dizajniran za raspored, modeliranje i pregled.

Kako slika izgleda naposljetku definiraju kamere, svijetla i materijali na objektima koji se prikazuju. Njih dijele *Eevee* i *Cycles* rendereri, no neke značajke su podržane samo u jednom ili drugom.

U ovome radu korišten je renderer *Eevee* [6], ali ne interaktivno u stvarnome vremenu (zbog ograničenja računalne opreme), već za izradu finalnih slika. Budući da *Eevee* ne koristi *raytrace* način rada poput *Cycles* načina, nikada neće biti dostići fizičku ispravnost koju *Cycles* pruža, ali je zato manje zahtjevan. Koristi se proces

rasterizacije koja procjenjuje način na koji svjetlost stupa u interakciju s objektima i materijalima pomoću raznih algoritama kojima Blender sam rukuje.

Kako je već spomenuto, prikazivanje animacije nije interaktivno u stvarnome vremenu, već se svaki okvir (slika koja prikazuje jedno finalno stanje iteracije simulacije) po završetku izračuna i ažuriranja objekata na sceni sprema kao slika u PNG formatu. Na kraju simulacije, od niza slika slaže se animacija pomoću Blenderovog *Video sequencer* (hrv. nizatelj, alat za nizanje slika u video uradak), određuju se *frame rate* (hrv. okvirna stopa, stopa koja određuje koliko slika, odnosno okvira, čini jednu sekundu videa), format izlazne datoteke (odabran FFMPEG), način enkodiranja i slično [6]. Izbornik za određivanje svih navedenih postavka izlazne datoteke, zajedno sa postavkama korištenim u ovome radu, vidljiv je na slici 2.2.



Slika 2.2 Izbornik postavka za izlaznu datoteku u Blenderovom alatu *Video sequencer*

Poglavlje 3

Izračuni, algoritmi i modeli simulacije

Pri izradi rada testirano je i korišteno više algoritama, verzija izračuna i konstanti prema potrebi situacije simulacije i očekivanog ishoda.

3.1 Vektori i normale

Za shvaćanje i korištenje algoritama korištenih u modeliranju tkanine i simuliranju fizike u trodimenzionalnom svijetu potrebno je shvaćanje vektora, vektorskih veličina i njihovih normala.

3.1.1 Vektori

U elementarnoj matematici i fizici, vektor se označava kao veličina koja ima iznos, smjer i orijentaciju. To se donosi samo na trodimenzionalno okruženje s kojime smo najbliže i upoznati iskustvom. Vektori su uvedeni kao složenija veličina od skalara koji predstavljaju samo iznos koji je opisan realnim brojem. Za opis vektora potrebna su tri broja od kojih svaki predstavlja iznos toga vektora u jednoj od tri dimenzije koordinatnog sustava omeđenog sa tri osi - **x**, **y** i **z**. [7]

Vektori su posebni prema pravilima koja moraju biti zadovoljena za njihovo ispravno korištenje te je uvijek dobro znati rukovati osnovnim operacijama nad vektorima, kao što su (korišteni u ovome radu):

1. Zbrajanje i oduzimanje vektora
2. Vektorski produkt
3. Duljina vektora
4. Množenje vektora sa skalarom
5. Suprotni vektor.

Također, korištena je L^2 norma vektora, kojom se izračunava udaljenost vektorskih koordinata. Kao takva, poznata je i kao euklidska norma jer se izračunava kao euklidska udaljenost (udaljenost koju karakterizira najkraći razmak između dvije točke u jednom prostoru). Rezultat je pozitivna vrijednost udaljenosti.

$$\|\vec{\mathbf{x}}\| = \sqrt{x_x^2 + x_y^2 + x_z^2} \tag{3.1}$$

Pri tome je bitno razlikovati L^2 normu vektora i normaliziran vektor, koji se oboje koriste u radu. Dok norma rezultira udaljenosti između dvije točke, normalizirani vektor zadržava podatak o svome smjeru ali je jediničnog iznosa (jedinični vektor), koji nije nul vektor:

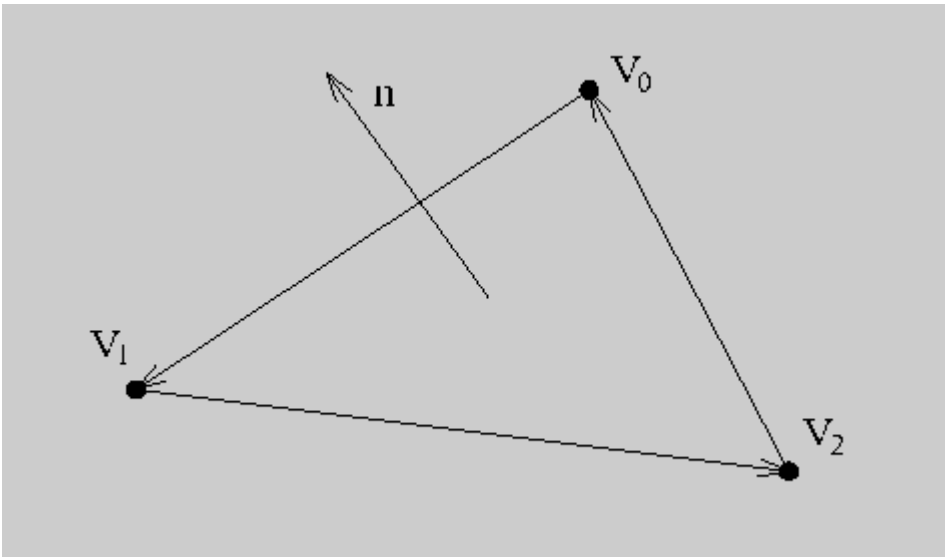
$$\begin{aligned} x_0 &= \frac{x}{|x|} \\ |x_0| &= \left|\frac{x}{|x|}\right| = \frac{1}{|x|}\,|x| = 1 \end{aligned} \tag{3.2}$$

3.1.2 Normale

Normale su određene kao vektori ili pravci okomiti na objekt na kojega se odnose. U ovome radu bitne su nam normale na površinu, odnosno poligone, tkanine jer određuju smjer utjecaja aerodinamičnih sila na tkaninu objašnjenih u kasnijem poglavlju (5.3.3). Kao primjer korištenja, za trokut omeđen točkama V_0, V_1 i V_2 komponente normale n u tri dimenzije bi se izračunale pomoću izraza:

$$\vec{n} = \frac{V_0\vec{V}_1 \times V_0\vec{V}_2}{\|V_0V_1 \times V_0V_2\|} \tag{3.3}$$

vidljivo na slici 3.1.



Slika 3.1 Orijentacija trokuta i njegova normala

3.2 Općeniti pregled modela u računalnoj grafici i utjecaji na njih

Pri implementaciji bilo kakvih simulacijskih izračuna na nekome objektu, potrebno je prvo odlučiti koji model za prikaz željenog objekta će biti korišten. Modeli su matematička reprezentacija bilo kojeg trodimenzionalnog objekta, a postaju grafika kada se prikažu na ekranu. [8]

3.2.1 Poligoni (građevni blokovi/elementi modela)

Svaki objekt se u matematičkom i grafičkom smislu može prikazati uz pomoć jednostavnih geometrijskih oblika, od kojih je najkorišteniji trokut. Iako samostalno matematički izuzetno dosadan oblik, spajanjem mnoštva trokuta u mrežu (eng. *mesh*) trokuta dobivamo mogućnost stvaranja zapanjujuće složenih geometrijskih oblika (primjeri vidljivi na slici 5.1) kojima se i dalje može manipulirati sa jednostavnim matematičkim operacijama koje bi vrijedile i za pojedinačne elemente. [2] Iz tog razloga je i u ovome radu korišten model koji se sastoji od mnoštva malenih trokuta.

3.2.2 Čestice

Svaki trokut mreže je opisan sa tri čestice koje ga omeđuju i predstavljaju njegove vrhove. Te čestice su u kodu definirane sa svojom trodimenzionalnom pozicijom i svojom brzinom, odnosno ubrzanjem.[2] Stanje svake čestice se opisuje sa kombinacijom tih dvaju parametara, te se često pohranjuju kao liste pozicija i brzina kojima se indeksi podudaraju, što je i bila prvotna implementacija u ovome radu. Kasnije, u sekciji 6.1 objašnjen je razlog *ne korištenja* pohrane u ovome obliku.

3.2.3 Sile koje utječu na model

Na tkaninu bi u stvarnome svijetu konstantno utjecale vanjske sile poput gravitacije, vjetra, sudara sa ostalim objektima i samom sobom. Dok one uzrokuju da se tkanina pomiče, a ne da stacionarno lebdi, pravoj i stvarnoj kretnji tkanine doprinose najviše unutarnje sile. To su sile istezanja (eng. *stretch*), smicanja (eng. *shear*) i savijanja (eng. *bend*) koje djeluju na segmente tkanine (čestice i opruge, objašnjene u sekciji 4) kako bi se ponašala poput tekstila. [2]

Važno je napomenti da se ovaj rad odvija na računalu, što znači da nemamo beskonačno dostupne resurse, pa čak se ne bi reklo niti *puno* resursa. Kroz ovaj rad će se zato jednostavnim izračunima formulirati te unutarnje sile kao reakcija na vanjske, te će se koristiti numerička integracija za unaprijeđivanje simulacije tijekom vremena.

Poglavlje 4

Mass-Spring model

U ovome poglavlju biti će pobliže objašnjen odabran i korišten matematički model za implementaciju simulacije tkanine.

4.1 Uvod

Kako je već odavno poznato da se cijeli svijet sastoji od mnoštva atoma, koristeći tu ideju se u računalnoj grafici stvarni svijet može imitirati, te se za simuliranje tkanine najčešće koristi sličan oblik modeliranja. Tako ideja modela postaje predstaviti tkaninu kao neprekinuti diskretni skup točaka koje međusobnim poveznicama sprječavaju pretjerano istezanje ili kompresiju. [2]

4.2 Čestice i njihove mase

Iz imena samog modela već je vidljivo da se radi o točkama mase, odnosno točkama koje imaju svoju određenu masu, koje su povezane oprugama. Kako se tim točkama ne može dodati stvarna težina kakvu možemo osjetiti, svakoj točki se pridodaje masa kao parametar koji ju opisuje i koji će utjecati na izračun njezine pozicije u vremenu, te se time može dobro aproksimirati ponašanje različitih vrsta tkanine. Primjerice, svilena tkanina će u svome modelu pridodati relativno malenu masu svim točkama

dok će se vuni pridodati veća masa da bi se prikazala težina tkanine kakvu se očekuje iz stvarnog iskustva.

To rezultira prvim od dva građevna elementa koji određuju ovaj model, a to su **čestice** (eng. *particles*). Svaka čestica se karakterizira svojom pozicijom $\mathbf{x}$, brzinom ili ubrzanjem (akumulacijom sila, parametar koji se odabire prema algoritmu korištenom u implementaciji) $\mathbf{v}/\mathbf{a}$ i svojom masom $\mathbf{m}$.

U ovom poglavlju, čestice će biti određene brzinom radi jednostavnosti matematičkih izraza.

U mnogim izvorima i matematičkim reprezentacijama modela mase-opruge, čestice se prikazuju kao matrice, dok će u poglavlju Implementacija (5), biti prikazano kako je jednostavnije pohraniti ih kao listu elemenata tipa čestice (kako često ne postoji takav predodređen tip, najbolje je vlastoručno implementirati klasu ili strukturu prema potrebama simulacije).

4.3 Opruge

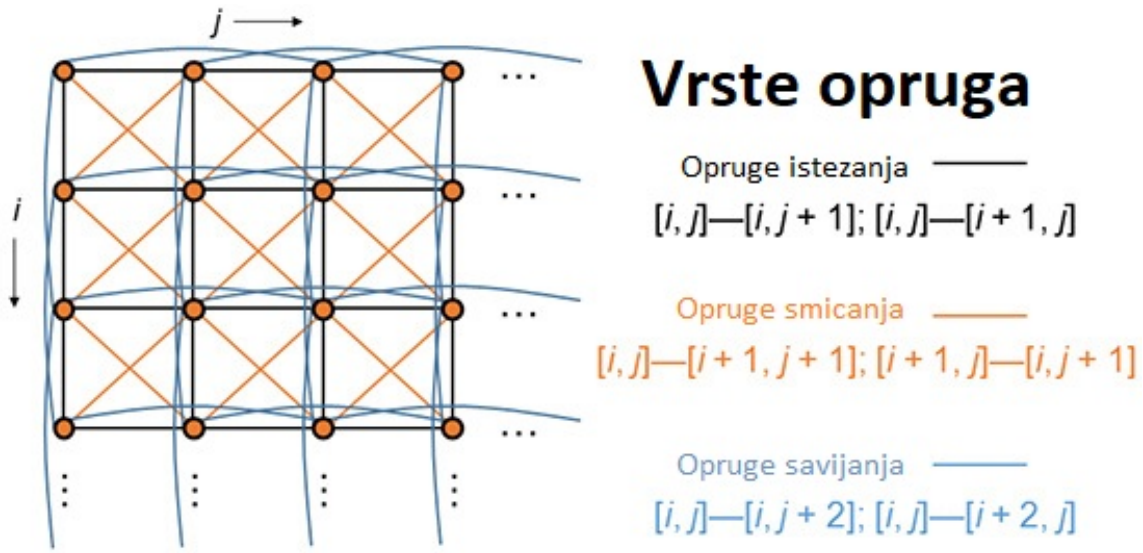
Nošenjem odjeće i međudjelovanjem sa tkaninom u svakodnevnom životu, znamo očekivati kako se tkanina ponaša, odnosno koje rezultate simulacije *ne želimo*. To mogu biti pretjerano rastezanje i smicanje, dok s druge strane, tkanina u stvarnome svijetu ima sklonost savijanju i stvaranju nabora ponovno ovisno o materijalu od kojeg je predmet napravljen, te je cilj svake računalne simulacije pronalazak stabilnog ponašanja koje najprirodnije imitira kako kretnja tkanine reagira na sile koje utječu na nju.

Kako bi se modelirala otpornost na deformacije, mogu se konstruirati jednostavni spojevi za svaki par susjednih čestica, još nazivani **oprugama** (eng. *springs*). To je drugi građevni element modela **masa-opruga** (eng. *mass-spring*), uz prethodno spomenute čestice. Opruge tako omeđuju svaki poligon (trokut) modela kao njihove stranice dok čestice predstavljaju njihove vrhove.

Postoji tri vrste opruga koje svaka spajaju druge susjedne čestice:

1. Opruge istezanja

Slika 4.1 Prikaz opruga istezanja, smicanja i savijanja te kombinacija čestica koje ih definiraju [1]



2. Opruge smicanja
3. Opruge savijanja

Opruge istezanja (eng. *stretch springs*) - također zvane i **Strukturalne opruge** (eng. *structural springs*), na slici 4.1 prikazane crnom bojom. Spajaju susjedne čestice prema kombinacijama: $[i, j+1], [i, j-1], [i+1, j], [i-1, j]$. [1] Može ih se zamisliti kao katete svakog poligona.

Opruge smicanja (eng. *shear springs*) - na slici 4.1 prikazane narančastom bojom. Povezuju dijagonalno susjedne čestice prema kombinacijama: $[i + 1, j + 1], [i + 1, j - 1], [i - 1, j - 1], [i - 1, j + 1]$. [1] Može ih se zamisliti kao hipotenuzu svakog poligona.

Opruge savijanja (eng. *bend springs*) - na slici 4.1 prikazane plavom bojom. Jedine spajaju ne-susjedne čestice prema kombinacijama: $[i, j + 2], [i, j - 2], [i + 2, j], [i - 2, j]$. [1] Može ih se zamisliti kao katete duplo većih poligona od odabranih.

Svaka čestica, prema gore navedenim kombinacijama, može imati čak i do 12 opruga koje su spojene na njih.

Prema karakteristikama i ponašanju, najviše se izdvajaju opruge savijanja. Dok su opruge istezanja i smicanja zaslužne za većinu stabilizacije sustava čestica, i njihovo zadržavanje na korektnoj međusobnoj udaljenosti, najviše zanimljivih vizualnih efekata kao što su zahtjevniji i kompleksniji nabori proizlaze iz opruga savijanja. Zato i najčešće najveću varijaciju u konstanti opruge (konstanta koja reprezentira krutost opruge, kolika je mogućnost opruge da se istegne, predstavlja elastičnost) imaju opruge savijanja, te ih se često odvojeno pohranjuje u modelima od opruga smicanja i istezanja. [2]

Preporučuje se, kao smjernica, oprugama istezanja zadati veću vrijednost (krutu konstantu), dok bi opruge smicanja i savijanja imale manje vrijednosti (elastičnije konstante), ali svaki model je u potpunosti prilagodljiv potrebama te se često implementiraju samo poneke opruge i zadaju im se proizvoljne vrijednosti konstanta opruge [2].

4.3.1 Potencijalna energija i sile na opruzi

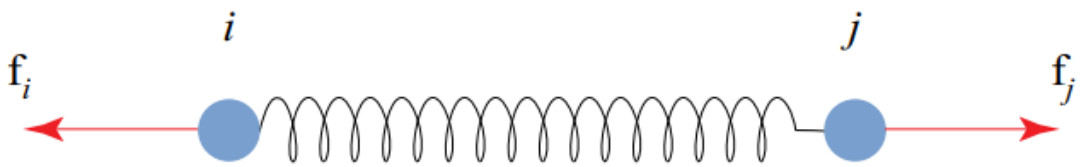
Energija se pohranjuje i odražava na ponašanje opruge jedino kada se njena duljina ne podudara sa duljinom kada su čestice na svojoj udaljenosti u mirovanju (eng. rest length), odnosno duljinom opruge kada je tkanina u potpunosti ravna i u stanju mirovanja. Ukratko rečeno, kada je opruga u stanju kompresije ili istezanja. Hooke-ov zakon (eng. *Hooke's law*) pruža nam jednostavan način određivanja kolika je potencijalna energija sadržana u opruzi ako imamo podatak duljine opruge u mirovanju i trenutne duljine u određenom vremenskom koraku. [2] Primjerice za čestice i i j, ta ekspresija bi glasila:

$$E_{ij}(x) = \frac{1}{2}k(\|x_i - x_j\| - L)^2 \tag{4.1}$$

pri čemu je **k** konstanta krutosti opruge (eng. *spring stiffness constant*) te **L** duljina u mirovanju opruge.

Ta energija utječe na obje čestice podjednako, iako na svaku u suprotnome smjeru ($f_i = -f_j$) vidljivo na slici 4.2, te se iznos sile koji se pridodaje akumuliranom iznosu sila na svaku česticu može odrediti pomoću prve derivacije energije:

Slika 4.2 Prikaz utjecaja unutarnjih sila na oprugu [2]



$$f_i(x) = -\frac{\delta E_{ij}(x)}{\delta x_i} = -k(\|x_i - x_j\| - L)\frac{(x_i + x_j)}{\|x_i - x_j\|} \tag{4.2}$$

$$f_j(x) = -\frac{\delta E_{ij}(x)}{\delta x_i} = k(\|x_i - x_j\| - L)\frac{(x_i + x_j)}{\|x_i - x_j\|} \tag{4.3}$$

4.3.2 Sila prigušenja opruge

Najjednostavniji način modeliranja sile prigušenja u sustav je dodavanje sile koja se protivi trenutnoj kretnji čestice. Primjerice, za čestice i i j, sila prigušenja iznosila bi:

$$d_i(x) = -k_d(v_i - v_j) = -d_j(x) \tag{4.4}$$

Naravno, kada bi se ta sila pridodala, poput svih drugih u sustavu, imala bi suprotne predznake za svaku česticu spojenu na istu oprugu. Varijabla k_d iz izraza predstavlja konstantu prigušenja (eng. *damping coefficient*). [2] Dodavanje sile prigušenja imitira efekt u stvarnome svijetu stalnog gubitka energije prouzročenog time da se tkanina ne nalazi u zatvorenim sustavima.

4.4 Verlet integracija i Euler-ova integracija

Sada, znajući kako reprezentirati tkaninu uz pomoć čestica i opruga, potrebno je sagledati kako izračunati promjenu njihovih pozicija u nekome vremenu.

Krećući od jednostavnijih algoritama prema kompleksnijima, prva tehnika s kojom se treba upoznati je sama vremenska integracija, odnosno matematičko određivanje kako se dinamički sustavi unaprijeđuju kroz vrijeme. Ponovimo da je svaka čestica određena sa svojom pozicijom $x_i = [x_{ix}, x_{iy}, x_{iz}]$ te brzinom $v_i = [v_{ix}, v_{iy}, v_{iz}]$. Naravno, te dvije vrijednosti se moraju mijenjati tijekom vremena, inače bi simulacija bila jako dosadna i statična. [2]

4.4.1 Eksplicitna integracija

Znamo iz osnovno-školske fizike i zakona gibanja da je brzina v_i prva derivacija pozicije x_i kroz vrijeme, te da je akceleracija $a_i = [a_{ix}, a_{iy}, a_{iz}]$ jednaka drugoj derivaciji pozicije, odnosno prvoj derivaciji brzine u vremenu. [2] Matematički, te izraze možemo zapisati kao:

$$\begin{aligned} \frac{d\mathbf{x}(t)}{dt} &= \mathbf{v}(t) \\ \frac{d\mathbf{v}(t)}{dt} &= \mathbf{a}(t) \end{aligned} \tag{4.5}$$

te, diskretizacijom korištenjem metode aproksimacije konačne razlike [9]:

$$\begin{aligned} \frac{\mathbf{x}(t+h) - \mathbf{x}(t)}{h} &\approx \mathbf{v}(t) \\ \frac{\mathbf{v}(t+h) - \mathbf{v}(t)}{h} &\approx \mathbf{a}(t) \end{aligned} \tag{4.6}$$

možemo zaključiti da se svaka iduća pozicija čestice može odrediti kao stara pozicija vektorski zbrojena sa vremenskim korakom h koji se odvio između trenutaka stare i nove pozicije, te pomnoženo sa tim vremenskim korakom i trenutnom brzinom čestice (isto i za akceleraciju) [2]:

$$\begin{aligned} \mathbf{x}(t+h) &\approx \mathbf{x}(t) + h\mathbf{v}(t) \\ \mathbf{v}(t+h) &\approx \mathbf{v}(t) + h\mathbf{a}(t) \end{aligned} \tag{4.7}$$

Kako se na čestice odražavaju direktno sile $\mathbf{f}$, a znamo masu $\mathbf{m}$ svake čestice, možemo zaključiti da se sile koje utječu na sustav ponašaju kao akceleracije $\mathbf{a}$ prema Newtonovom zakonu o gibanju [2].

$$\mathbf{f}(\mathbf{x}, \mathbf{v}, t) = \mathbf{ma}(t) \tag{4.8}$$

Iz iste formule vidimo da trebamo dobiti jednu silu, odnosno akumulaciju svih sila koje djeluju na sustav u trenutnom vremenskom koraku, te to obuhvaća sve *unutarnje* i *vanjske* sile koje utječu na svaku čestice zasebno (zašto zasebno, a ne na sve čestice jednako, primjerom je objašnjeno u sekcijama implementacije (5.3.2) i (5.3.3).

Unutarnje sile rezultiraju iz samoga modela tkanine. Označavaju reakciju na unutarnje deformacije tkanine kao što su rastezanje i savijanje. **Vanjske sile** su bilo koje sile koje iz okruženja utječu na tkaninu poput gravitacijske sile, sudara sa drugim objektima ili tkanine samom sobom te ostali aerodinamički utjecaji poput sile fluida (primjerice, sile vjetra).

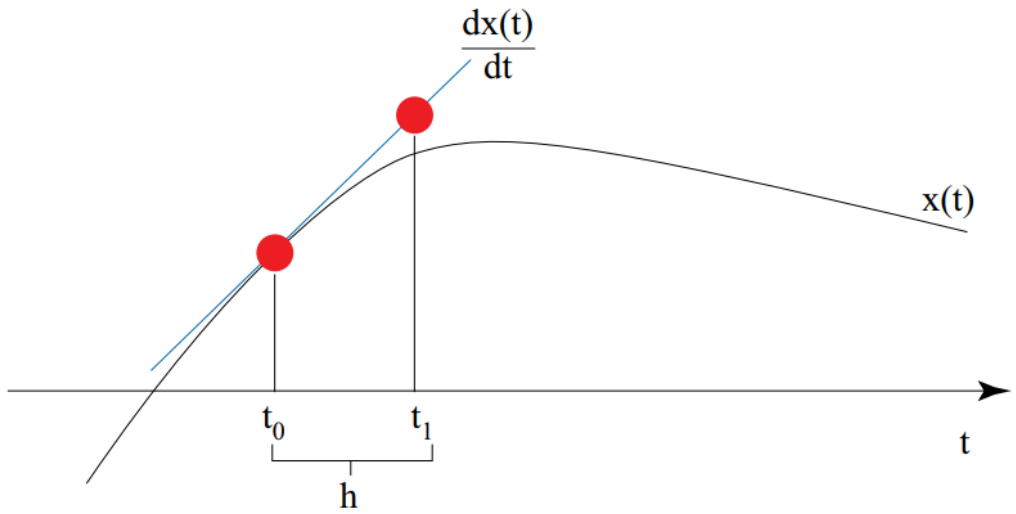
Koristeći prethodne izraze 4.7 i 4.8, možemo vidjeti da se pozicija $\mathbf{x}$ i brzina $\mathbf{v}$ u svakom vremenskom koraku h mogu definirati kao:

$$\begin{aligned} \mathbf{x}_{n+1} &= \mathbf{x}_n + h\mathbf{v}_n \\ \mathbf{v}_{n+1} &= \mathbf{v}_n + h\mathbf{mf}_n \end{aligned} \tag{4.9}$$

Ukratko, kada se otkriju nove sile $\mathbf{f}$, mogu se uvrstiti kao akceleracije $\mathbf{a}$, čime se otkrivaju nove brzine $\mathbf{v}$ kojima se unaprijeđuju pozicije $\mathbf{x}$ čestica i tako unaprijeđuje čitava simulacija.

Eksplicitna integracija pretpostavlja da su sile i brzine koje utječu na sustav konstantne od početka do kraja vremenskog koraka. Naravno, u stvarnome životu to skoro pa nikada neće biti slučaj, ali gledajući na jednostavnost sustava, ovo rezultira dovoljno stabilnom aproksimacijom kada je u pitanju dovoljno mali vremenski korak [2]. Do sada objašnjena diskretizacija se naziva **Euler-ova integracija unaprijed** (eng. *Forward Euler Integration*), odnosno **Eksplicitna integracija** (eng. *Explicit Integration*). Dobra usporedba ispravnosti ove funkcije može se vidjeti na slici 4.3, gdje je prikazana kao tangenta na funkciju u trenutnom vremenskom koraku kada se dogodi mali vremenski pomak [2].

Slika 4.3 Aproksimacija konačne razlike brzine za jedan vremenski korak koji napreduje od $t0$ do $t1$ [2]



4.4.2 Implicitna integracija

Iako je eksplicitan Euler vrlo jednostavan izračun, stabilnost simulacije previše ovisi o malom vremenskom koraku koji često "traje predugo", odnosno samoj simulaciji treba predugo da se unaprijedi. Iz tog razloga se u izračune uključuje shema implicitne integracije. Ideja iza toga jest da će to stabilizirati simulaciju čak i kada se uzme veći vremenski korak, iako će se izračuni "zakomplicirati" na drugu derivaciju.

Kod prethodnog eksplicitnog Euler-a, izračuni su se događali u trenutku dodavanja sile, što bi dovodilo do problema kod stabilnosti sustava zato što se nije moglo predvidjeti da će buduće stanje oscilirati u odnosu na prethodno ili trenutno. Kod implicitnog izračuna, još zvanog **Euler-ova integracija unazad** (eng. *Backwards Euler Integration*), vidimo da se svaki izračun događa na *kraju* vremenskog koraka, gdje možemo biti sigurni da su se sve moguće oscilacije u tome periodu već uzele u obzir, te tako pruža buduće stanje koristeći prethodno i završno kao smjernice (određujući time gradijent od trenutnog na iduće stanje što vizualno rezultira puno stabilnijom simulacijom). Time se prestaje gledati na svaki izračun izolirano, već se povezuje stanja u gladak prijelaz [2].

$$\begin{aligned} \mathbf{x}_{n+1} &= \mathbf{x}_n + h\mathbf{v}_{n+1} \\ \mathbf{v}_{n+1} &= \mathbf{v}_n + h\mathbf{m}\mathbf{f}_{n+1} \end{aligned} \tag{4.10}$$

pri čemu je $\mathbf{f}_{n+1} = \mathbf{f}(\mathbf{x}_{n+1}, \mathbf{v}_{n+1}) = \mathbf{f}(\mathbf{x}_n + \Delta\mathbf{x}, \mathbf{v}_n + \Delta\mathbf{v})$.

Primjetno je da ovaj način sigurno koristi više resursa kod izračuna u svakom vremenskom koraku, ali uzimajući u obzir golem iznos izračuna potreban za stabilnu eksplicitnu integraciju, implicitan način postaje puno održiviji, pogotovo za kruće opruge [10].

4.4.3 Verlet integracija

Do sada je prikazano kako se promjene pozicija čestica mogu dobiti služeći se pozicijom $\mathbf{x}$ i brzinom $\mathbf{v}$ čestice, no u ovome radu je odabran algoritam bez brzine, koji služi jako sličnu shemu integracije Euler-ovoj implicitnoj integraciji. Umjesto da pohranjujemo poziciju i brzinu svake čestice, pohranjujemo samo trenutnu poziciju $x_{(t)}$ i *prethodnu* poziciju $x_{(t-\Delta t)}$, vidljivo iz izraza 4.11 [11].

$$\mathbf{x}_{(t+\Delta t)} = 2\mathbf{x}_{(t)} - \mathbf{x}_{(t-\Delta t)} + \mathbf{a}\Delta t^2 \tag{4.11}$$

Ovaj pristup naziva se Verlet integracijom i koristi se pri simuliranju molekularne dinamike. Prilično je stabilan budući da je brzina implicitno dana i onemogućava desinkroniziraciju pozicije i brzine.

Funkcionira zato što trenutnu brzinu možemo aproksimirati preko udaljenosti koju je čestica prešla u prethodnom vremenskom koraku, 4.12 i 4.13 [11]:

$$2\mathbf{x}_{(t)} - \mathbf{x}_{(t-\Delta t)} = \mathbf{x}_{(t)} + (\mathbf{x}_{(t)} - \mathbf{x}_{(t-\Delta t)}) \tag{4.12}$$

$$\mathbf{x}_{(t)} - \mathbf{x}_{(t-\Delta t)} \tag{4.13}$$

Ovaj način nije uvijek vrlo precizan (energija može napustiti sustav, tj. rasipati se), ali je brz i stabilan što je u računalnoj grafici često potreba.

Poglavlje 5

Implementacija

Ovo poglavlje prolazi kroz samu implementaciju modela i izraza iz Poglavlja 4 *Mass-spring* model.

5.1 Uvod

Pri implementaciji u Blenderu i Pythonu API-u, prvotno je bilo potrebno dobiti sve potrebne objekte na sceni na koje bi se primijenili modeli i koji bi izvodili i prikazali međuodnos u simulacijama. Dalje, bilo je potrebno implementirati svu "matematiku" u zasebne klase te odrediti kojim načinom bi se izvodila konačna simulacija.

5.2 Pregled dizajna

5.2.1 Podjela programskog koda

Kod je podijeljen u tri sekcije:

1. Definiranje globalnih konstanti
2. Definiranje klasa i pomoćnih funkcija

3. Postavljanje scene i objekata, pokretanje simulacije i prikazivanje i pohrana okvira animacije.

5.2.2 Definiranje globalnih konstanti

Prilikom implementacije, pokazalo se kao najjednostavnije "podignuti" konstante na globalnu razinu, da bi se izbjeglo prolaženje statičnih podataka kroz više poziva metoda u nizu izračuna. Tako su određene vrijednosti *MASS* (hrv. masa) koja iznosi jedan (`MASS = 1`) i predstavlja masu jedne čestice. Masa čestice u kompleksnijim modelima može biti definirana drugačije, posebice za čestice na rubovima tkanine kako bi se imitirali mogući šavovi ili presavijeni sloj tkanine. Određena je još jedna početna vrijednost koja se odnosi na inicijalizaciju čestica, *INIT_PARTICLE_ACC* (hrv. početna akceleracija čestice) koja iznosi vektorsku veličinu jednaku nula (0,0,0) i predstavlja početnu akceleraciju čestice pri postavljanju modela tkanine.

Konstanta *SATISFY_CONSTRAINT_ITERATOR* određuje koliko puta će se pri svakoj iteraciji simulacije pokrenuti metoda *satisfyConstraints()* objašnjena u sekciji 5.3.6 te tako definira elastičnost, odnosno krutost, tkanine. Što je ova konstanta većeg iznosa, to se tkanina ponaša čvršće te se ograničava mogućnost rastezanja opruga. Iako pomoću ove konstante možemo dodatno stabilizirati sustav, njeno povećanje izravno utječe na vrijeme koje je potrebno za izvođenje jedne iteracije simulacije.

BALL_RADIUS (hrv. radijus lopte), kako i samo ime kaže, definira radijus sfere koja se koristi pri simulaciji sudara sa krutim objektom.

Za definiranje vanjskih sila koje utječu na tkaninu koriste se *GRAVITY* i *WIND* vektori, koji iznose (0,0,-9,81) i (5,0,-3) respektivno. Ove sile određene su vektorima jer je simulaciji potreban podatak smjera i snage sile, što je i prikazano na primjeru u sekcijama 5.3.2 i 5.3.3.

Konstante *TIMESTEP* (hrv. vremenski korak) koja iznosi 0.1 i *DAMPING* (hrv. prigušenje) koja iznosi 0.1 (proizvoljno određene u okviru eksperimentalno isprobanih vrijednosti) konačne su globalne varijable potrebne za izvođenje simulacije. Vremenski korak određen je radi diskretizacije, dok je prigušenje, često nazivano i *drag* (hrv.

vuča, povlačenje), sila koja imitira rasipavanje energije koje je potrebno ručno pridodati u zatvorenom sustavu (dok je prisutno u otvorenom sustavu stvarnog svijeta).

5.2.3 Definiranje klasa i pomoćnih funkcija

Većina programskog koda nalazi se u ovoj sekciji gdje su definirani model tkanine objašnjen u sekciji 5.3, njegove popratne metode, pomoćne klase *Constraint* 5.4 i *Particle* 5.5 sa svojim metodama te sve podfunkcije koje su olakšavale implementaciju metoda modela.

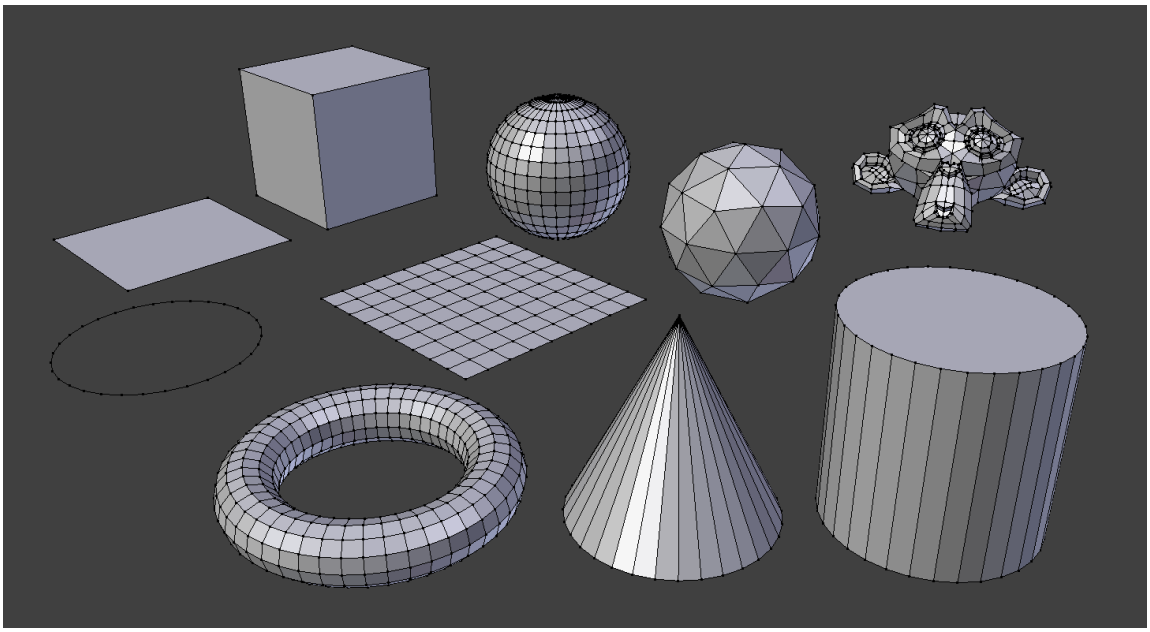
5.2.4 Postavljanje scene i objekata, pokretanje simulacije i prikazivanje i pohrana animacije

Glavna, *main()* funkcija programskog koda započinje sa Blenderovim Python API pozivom za dodavanje objekta tipa *primitive plane* `bpy.ops.mesh.primitive_plane_add(location=(0, 0, 0), size=10)` (hrv. primitivna, obična, jednostavna ravnina), pri čemu joj se zadaju parametri lokacije na (0,0,0) u trodimenzionalnoj sceni i veličine deset [3]. Prema Blenderovoj dokumentaciji, objekt ravnine je opisan kao ploha koja se sastoji od četiri vrha, četiri ruba i jedne plohe, vidljivo na slici 5.1 , te sama za sebe nije trodimenzionalna jer nema debljinu. Pomoću API poziva za dohvaćanje podataka o trenutno odabranom objektu, referenca za lakši pristup ravnini pohranjena je u varijablu *plane*.

Kako želimo da nam tkanina izgleda stvarno i dinamično, ostaviti ravninu u "jednome komadu", odnosno ostaviti ju definiranu samo jednim četverokutom, nije optimalno za simulaciju. Tako pozivom `bpy.ops.mesh.subdivide(number_cuts = 30)` u *Edit* (hrv. uređivačkom) načinu rada dijelimo ravninu na 30 manjih četverokutnih poddijelova. Ovime su se odredile opruge istezanja, no za realističniju simulaciju potrebne su nam još barem i opruge smicanja.

Uvođenjem pomoćnog Blender modula, *BMesh Module (bmesh)*, koji daje pristup Blenderovom unutarnjem API-ju za uređivanje mreže, te sadrži podatke o povezivanju geometrije i pristup operacijama uređivanja objekata koji se ponašaju kao mreže, API pozivom `bm = bmesh.from_edit_mesh(plane_data)` pridajemo referencu na

Slika 5.1 Primitivni i kompleksni objekti dostupni preko Blender Python API-ja [3]



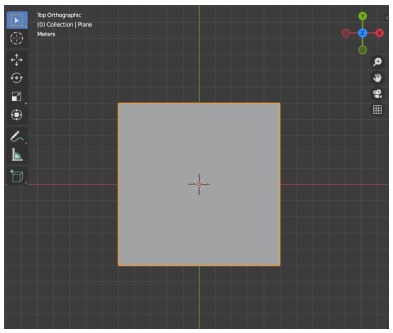
mrežne odredbe ravnine varijabli *bm* (kratica za blender mesh), te prolaskom kroz sve poligone ravnine (prethodno definirane podijelom ravnine na 30 "rezova") u mrežu za svaki poligon dodajemo dijagonalne stranice koje simulaciji predstavljaju opruge smicanja. Kreiranje i pridruživanje novih stranica vidljivo je iz isječka koda (5.1):

Listing 5.1 Dodavanje stranica koje predstavljaju opruge smicanja ravnini

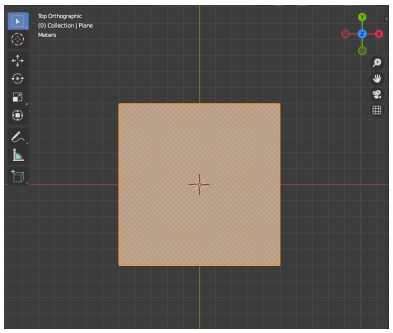
```
bpy.ops.object.editmode_toggle()
bm = bmesh.from_edit_mesh(plane.data)

bm.verts.ensure_lookup_table()
for i in range(len(plane.data.polygons)):
    v1 = bm.verts[plane.data.polygons[i].vertices[0]]
    v2 = bm.verts[plane.data.polygons[i].vertices[1]]
    v3 = bm.verts[plane.data.polygons[i].vertices[2]]
    v4 = bm.verts[plane.data.polygons[i].vertices[3]]
    bm.edges.new((v1, v3))
    bm.edges.new((v2, v4))
```

```
bmesh.update_edit_mesh(plane.data)
bpy.ops.object.editmode_toggle()
```

Sve novo određene podatke je naposljetku potrebno pridružiti objektu *plane* pozivom `bmesh.update_edit_mesh(plane_data)` zato što se do toga trenutka drže u privremenoj memoriji. Slike 5.2 i 5.3 prikazuju ravninu prije i nakon podijele i dodavanja novih opruga.



Slika 5.2 Ravnina prije primjene isječka koda za podijelu i dodavanje dijagonalnih spojnica između čestica

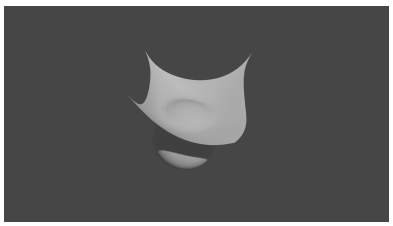


Slika 5.3 Ravnina nakon primjene isječka koda za podijelu i dodavanje dijagonalnih spojnica između čestica

Na pripremljenu ravninu se primjenjuje klasa *Cloth* (`cloth = Cloth(plane)`), gdje se inicijaliziraju svi potrebni parametri i metode za unaprijeđivanje simulacije, objašnjeno u sekciji 5.3.

Utjecaj koji dodavanje opruga smicanja ima na stabilnost sustava jednostavno je prikazati pokretanjem iste simulacije sa i bez njih. Na slikama 5.4 i 5.5 očito je koliko opruge smicanja tkanini dodaju na stvarnosti i odupiru se previsokim elastičnim svojstvima tkanine (obje slike su prikaz simulacije sudara sa sferom pri čemu su fiksirane tri rubne točke tkanine, te je uzet okvir 50. iteracije).

Za prikazivanje simulacije, odnosno prikaz (*render*) animacije, potrebni su još podaci o objektima kamere i svjetlosti na sceni. Pozivima `bpy.data.objects.new()` definiraju se objekti tipova *light* i *camera* kojima se proizvoljno postavljaju parametri lokacije, intenziteta, usjmerenja i slično. Lokaciju i usmjerenje bilo je najjednostavnije odrediti pomicanjem objekata u trodimenzionalnoj sceni *Viewport* Blendera te

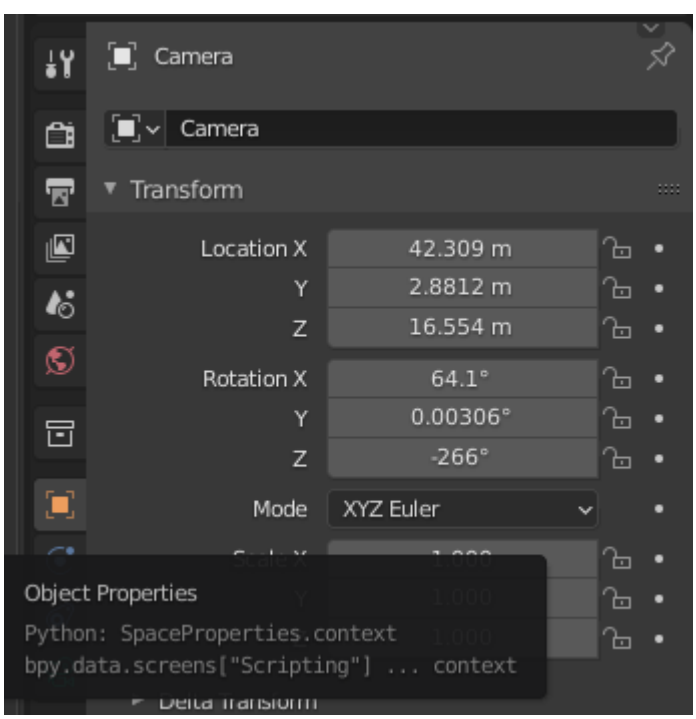


Slika 5.4 Prikaz simulacije sudara sa sferom bez pridodavanja opruga smicanja u model

Slika 5.5 Prikaz simulacije sudara sa sferom sa pridodanim oprugama smicanja u model

pregledom njihovih parametara u *Object properties* izborniku (vidljiv sa slike 5.6) i unosom istih u programski kod.

Neobavezan dio je postavljanje podataka o sferi ako se radi o simulaciji sudara sa krutim objektom, te su korišteni slični pozivi kao za dodavanje objekta ravnine `bpy.ops.mesh.primitive_uv_sphere_add(radius=BALL_RADIUS, location=(-1,`



Slika 5.6 Prikaz desnog izbornika gdje Blender omogućuje manulanu manipulaciju parametara svih objekata scene

`-1,-7))` i referenca na objekt je dodijeljena varijabli *ball*.

Završni dio programskog koda definira okvir-po-okvir način unaprijeđenja i pohrane slika scene nakon odrađenih svih izračuna u metodama klase *Cloth*. Definiranjem prvog i zadnjeg okvira pozivima: `bpy.context.scene.frame_set(0)` i `bpy.context.scene.frame_end = 100` određeno je koliko slika po završetku izvođenja imamo, a unaprijeđenje simulacije izaziva se pozivom funkcije *updateScene()*, koja sadrži jednu metodu klase *Cloth simulate()*, te se poziva na svaki render pozivom `bpy.app.handlers.frame_change_pre.append(updateScene)`.

5.3 Klasa *Cloth*

Klasa *Cloth* (hrv. tkanina) je jezgra cijelog koda jer sadrži pozive za rukovanje međuodnosom objekta na koji je primijenjena sa okolinom, u ovoj implementaciji to je Blenderov objekt *plane* (hrv. ravnina) iz modula *Data Access (bpy.data)*: `bpy.data.objects["Plane"]`. To je jedini unos koji klasa *Cloth* očekuje, te pri inicijaliziranju pokreće metodu *__init__()* koja je zaslužna za sljedeće akcije:

1. Definiranje inicijalnih parametara klase:
   (a) *object* - Parametar koji sadrži cijeli objekt koji je proslijeđen klasi (sama ravnina).
   (b) *particles* - Lista namijenjena pohranjivanju čestica sustava. Pohranjuju se elementi tipa `Particle` (prilagođena klasa objašnjena kasnije u sekciji 5.5) koja predstavlja česticu sustava.
   (c) *constraints* - Lista namijenjena pohranjivanju podataka tipa `Constraint` (prilagođena klasa objašnjena kasnije u sekciji 5.4) koja predstavlja oprugu sustava.
2. Poziv metode *setup()* koja služi za popunjavanje prethodno definiranih lista sa podacima iz objekta ravnine.

Metoda *setup()* koristi Blenderove Python API pozive: `self.object.data.vertices` i `self.object.data.edges` za pristup podacima o svim vrhovima (eng. *ver-*

tices) i rubovima (eng. *edges*) poligona ravnine. Time u modelu definiramo koliko čestica tkanina sadrži, koje pozicije one zauzimaju te koje relacije formiraju opruge povezane na njih.

5.3.1 Metoda *simulate()*

Glavna metoda klase *Cloth* zove se *simulate()* te se poziva iz *main()* funkcije pri svakoj iteraciji, odnosno prikazu simulacije. Kao parametre prima neobavezan podatak položaja centra sfere ako se radi o simulaciji kolizije tkanine sa krutim objektom, a pokreće ostale metode koje zapravo unaprijeđuju simulaciju i rješavaju čitavu matematiku "u pozadini". Metoda bi također mogla prihvaćati ostale utjecaje iz okruženja kao što su utjecaj sile gravitacije, te utjecaj sile vjetra, ali radi jednostavnosti programa, ti podaci se pridružuju iz globalnih konstanti definiranih u sekciji 5.2.2.

Ukratko, metoda *simulate()* prema potrebi poziva metode: *addSimpleForce()* (5.3.2), *addWindForce()* (5.3.3), *handleBallCollision()* (5.3.4), *updateParticles()* (5.3.5), *satisfyConstraints()* (5.3.6) te *handleDeformationRate()* (5.4.2).

5.3.2 Metoda *addSimpleForce()*

Početna metoda svake iteracije simulacije je *addSimpleForce()* (hrv. dodavanje "jednostavnih" sila) te je namijenjena nadodavanju svih jednostavnih sila koje u nekome trenutku utječu na čestice sustava. U slučaju simulacije ovog rada to je samo gravitacijska sila. Kako većina sila na sve čestice *ne utječe* istom jačinom niti u istome smjeru, ova metoda služi samo za pridodavnje sila koje ne padaju pod taj uvjet (jednostavne su po pitanju smjera i utječu na sve čestice uvijek istom jačinom i smjerom).

U ovoj metodi, dodaje se vrijednost sile gravitacije (u metodu ulazi pod nazivom *force*) tako što se poziva metoda *calculateAcceleration()* 5.5.2 na svakoj čestici.

Listing 5.2 Petlja dodavanja gravitacijske sile na čestice

```
for particle in self.particles:
    particle.calculateAcceleration(force)
```

5.3.3 Metoda *addWindForce()*

Sila vjetra je malo kompliciranija u izračunu jer vjetar djeluje na razini poligona tkanine, ne može se aproksimirati kao djelovanje sile direktno na čestice poput gravitacije. Iako se na prvu čini zahtjevna matematički za razumjeti, u implementaciji se vidi koliko je zapravo jednostavna logika iza djelovanja aerodinamičkih sila na objekte koji se ponašaju kao mreža poput tkanine.

Vidljivo iz isječka koda 5.3, vjetar dodaje silu čestici u smjeru njezine normale s veličinom jednakom vektorskom produktu između normale poligona i smjera vjetra. Na taj način vjetar ima puni učinak na okomitu tkaninu, a gotovo nikakav na paralelnu tkaninu (paralelnu sa smjerom vjetra) [12]. Taj postupak ponavlja se za svaki poligon sustava.

Listing 5.3 Dodavanje sile vjetra na jedan poligon (čestice koje ga omeđuju)

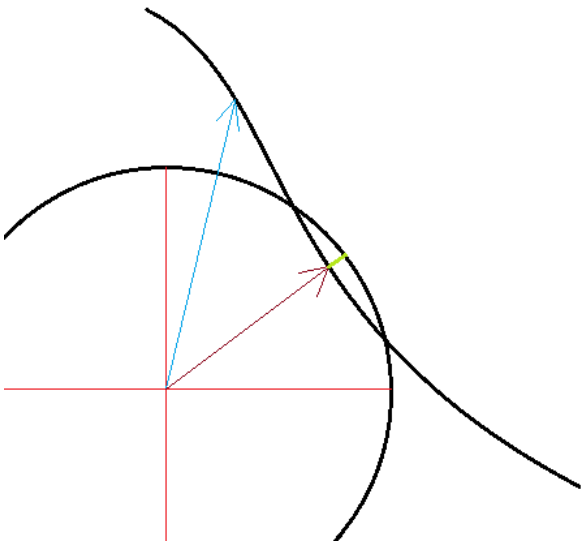
```
normal1 = _computeNormal(
    p1.position.co, p2.position.co, p3.position.co)
nVect1 = m.Vector(normal1)
nVect1.normalize()
force1 = normal1 * (nVect1.dot(WIND))
if (not p1.grounded):
    p1.calculateAcceleration(force1)
if (not  p2.grounded):
    p2.calculateAcceleration(force1)
if (not  p3.grounded):
    p3.calculateAcceleration(force1)
```

5.3.4 Metoda *handleBallCollision()*

U prijevodu, "rješavanje" sudara sa loptom, kao što i sam naziv nalaže rješava problem sudara tkanine sa krutim objektom poput sfere. Sama metoda sastoji se od prolaska kroz sve čestice sustava i provjere ako je ta čestica uopće u sudaru. U računalnoj grafici, ta provjera se naziva *collision detection* (hrv. otkrivanje sudara) te služi za otkrivanje čestica koje zahtjevaju ažuriranje pozicije da bi se našle na točnom

dijelu površine krutog objekta. Otkrivanje sudara tkanine i sfere je vrlo jednostavno i zahtjeva samo znanje o trenutnoj poziciji čestice i radijusu sfere koja sudjeluje u simulaciji.

Otkrivanje sudara se izvrši tako da se prati pozicija središta sfere te, ako se udaljenost između čestice i središta pokaže manja od radijusa sfere, tada je uspješno otkriven sudar. Pozicija se korigira tako da se vektor između čestice i središta normalizira te se poveća za iznos koji je čestica "prekoračila" u sferu (razliku između radijusa/površine sfere i trenutne pozicije čestice), vidljivo iz isječka 5.4 te slike 5.7.



Slika 5.7 Otkrivanje sudara sa sferom i ispravak položaja čestice

Slika 5.7 je dvodimenzionalni prikaz simulacije sudara radi jednostavnosti, te su sfera i tkanina označene crnom bojom. Čestica na koju pokazuje plava strelica, koja ide od središta sfere, nije u sudaru, dok je čestica na koju pokazuje crvena strelica od središta otkrivena kao da je u sudaru te se u istome smjeru vrši ispravak pozicije za iznos za koji je čestica prešla površinu sfere (prikazano zelenom bojom).

Ovaj način rješavanja problema otkrivanja i korigiranja sudara sa krutim objektima dobro predstavlja potrebu u računalnim animacijama za dinamičkim inverznim postupcima. Dinamički inverzni postupci označavaju postupke koji se izvršavaju *nakon* samog događaja koji bi se trebao korigirati. Tako i ovdje, prvo se propusti

pomak čestice kroz površinu sfere te se naknadno otkrije ako je čestica u prostoru u kojem se smije nalaziti i izvede korekcija ako taj uvjet nije zadovoljen [13].

Listing 5.4 Otkrivanje sudara i ažuriranje pozicija čestica

```
def handleBallCollision(self, ballLoc):
    for particle in self.particles:
        if(not particle.grounded):
            vect = m.Vector(particle.position.co - ballLoc)
            dist = vect.magnitude
            if(dist < BALL_RADIUS + 0.59):
                vect.normalize()
                vect = vect * (BALL_RADIUS + 0.59 - dist)
                particle.position.co =
                    particle.position.co + vect
```

5.3.5 Metoda *updateParticles()*

Ažuriranje pozicije čestica poziva se tek nakon što sve sile i objekti stignu stupiti u interakciju sa tkaninom i dati podatke potrebne za smještanje čestice u točan dio trodimenzionalne scene. Kako je sam algoritam izračuna nove pozicije čestica implementiran u pomoćnoj klasi *Particle* (5.5, 5.5.1), ova metoda samo poziva tu metodu iterirajući kroz sve čestice.

5.3.6 Metoda *satisfyConstraints()*

Kako se česticama u prethodnim metodama pridodaju sile, što im direktno daje neku akceleraciju u smjeru utjecaja te sile (točnije agregaciji smjera poligona i smjera sile), to rezultira promjenom u duljini svake opruge, u ovome radu *Constraint* (5.4) objekta. Kao i u prethodnoj, ova metoda služi za iteriranje kroz sve opruge i poziv istoimene metode (5.4.1) nad samim *Constraint* objektima.

Ova metoda još ispunjuje zadaću određivanja krutosti, odnosno elastičnosti, opruga iteracijom kroz prethodno definiranu konstantu *SATISFY_CONSTRAINTS_ITERATOR*.

5.4 Klasa *Constraint*

Pomoćna klasa za definiranje opruga koja sadrži parametre:

1. *p1* i *p2* - Čestice koje opruga povezuje (tipa `Particle`)
2. *restLen* - Podatak o duljini opruge u mirovanju

koji su sve definirani pri inicijalizaciji objekta i više se naknadno ne izmjenjuju.

Također sadrži metode *satisfyConstraint()* i *handleDeformationRate()* koje se pozivaju u glavnoj *Cloth* klasi.

5.4.1 Metoda *satisfyConstraint()*

Hooke-ov zakon iz sekcije 4.3.1 tvrdi da je istezanje opruge proporcionalno primijenjenoj sili, vidljivo iz izraza 5.1 gdje je k_s konstanta opruge i $\mathbf{x}$ duljina opruge. Primjenom Hooke-ovog zakona, uz dodatak prigušenja (5.2, gdje je k_s konstanta prigušenja i $\mathbf{v}$ brzina čestice) za smanjenje oscilacija, nova unutarnja sila prouzročena protudjelovanjem opruga na vanjske sile može se primijeniti na simulaciju [14].

$$\mathbf{f}_{spring} = -k_s\mathbf{x} \tag{5.1}$$

$$\mathbf{f}_{damping} = -k_d\mathbf{v} \tag{5.2}$$

5.4.2 Metoda *handleDeformationRate()*

Ova metoda pokušava riješiti problem implementacije efekta super-elastičnosti na koji simulacija nailazi 6.2. Naime, simulacija pokazuje veliku razinu deformacije kod duljine opruga povezanih na čestice koje su fiksirane u simulaciji. Ta deformacija se lokalno proširuje, te je implementirana ova metoda kao pokušaj saniranja pogreške umanjenjem duljine deformirane opruge ne bi li se vratila na svoju duljinu mirovanja [13]. Metoda ima vidljivog utjecaja na model, ali nedovoljnog intenziteta, te pogreška i dalje ostaje prevelika za normalno funkcioniranje modela pri tim uvjetima.

5.5 Klasa *Particle*

Pomoćna klasa za definiranje čestica koja sadrži parametre:

1. *position* - Vektor koji definira trenutnu poziciju čestice.
2. *oldPosition* - Vektor koji definira poziciju čestice iz prethodne iteracije, te se pri inicijalizaciji postavlja na istu vrijednost kao parametar *position*.
3. *acceleration* - Vektor koji definira trenutnu akceleraciju čestice, te se pri inicijalizaciji postavlja na vrijednost *INIT_PARTICLE_ACC*.
4. *mass* - Vrijednost koja definira masu čestice (sve čestice radi jednostavnosti u sustavu imaju masu jednaku konstanti *MASS*).
5. *grounded* - Boolean vrijednost koja definira da li je čestica fiksirana ili slobodna.

Ova klasa sadrži najvažniji algoritam u programskom kodu, samu Verlet integraciju koja mijenja položaje čestica u vremenu. Izračun se poziva pokretanjem metode *updatePositionByVerlet()*.

5.5.1 Metoda *updatePositionByVerlet()*

Pri implementaciji se zapravo i vidi snaga implicitne Euler-ove integracije, odnosno Verlet integracije, te se cijela metoda svodi na svega nekoliko linija koda.

Privremeno pohranjujemo trenutnu poziciju čestice u varijablu **temp** te prelazimo na sam algoritam. Iz isječka je vidljiv izraz Verlet integracije objašnjene u sekciji 4.4.3. Sama implementacija je izraz 4.11 prebačen u isječak koda 5.5 [15].

Listing 5.5 Ažuriranje pozicije čestice pomoću Verlet integracije

```
self.position.co = self.position.co
    + (self.position.co - self.oldPosition.co)
    * (1 - DAMPING)
    + (self.acceleration * TIMESTEP * TIMESTEP)
```

Nakon izračuna, parametar *oldPosition* ažuriramo sa varijablom **temp** te resetiramo parametar *acceleration* na vektor *INIT_PARTICLE_ACC*. Iz stvarnoga života

znamo da se sile koje utječu na tkaninu ne akumuliraju (nakupljaju) u tom smislu riječi, već konstantno utječu na nju istom (ili izmjenjivom, ali nikada akumuliranom) jačinom. Iz toga razloga se sve sile na kraju metode resetiraju.

5.5.2 Metoda *calculateAcceleration()*

Pomoćna metoda koja prima neku silu i izračunava akceleraciju koju ta sila predstavlja na česticu implementirajući Newtonov zakon o gibanju (4.8):

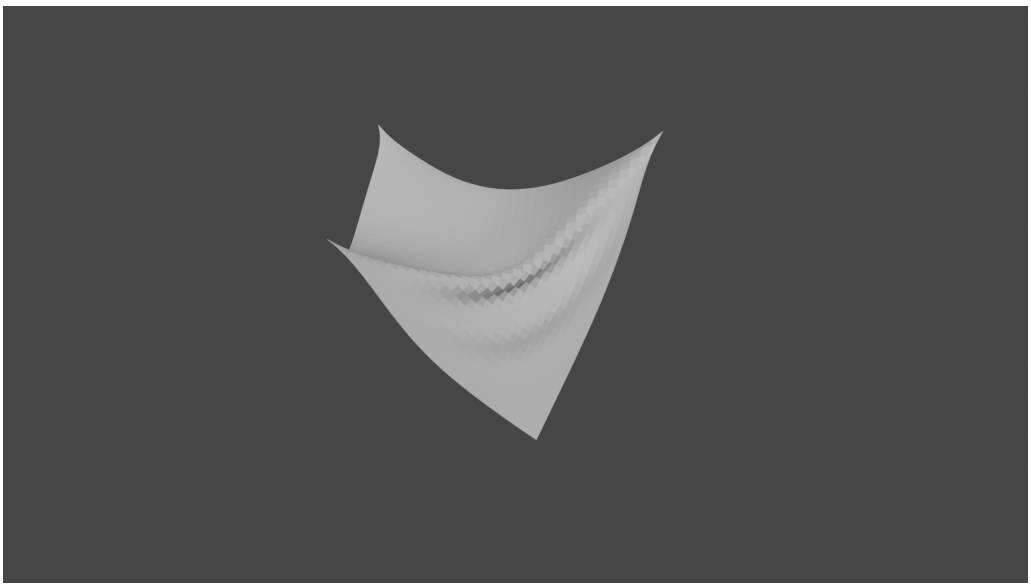
Listing 5.6 Metoda calculateAcceleration()

```
self.acceleration += force/self.mass
```

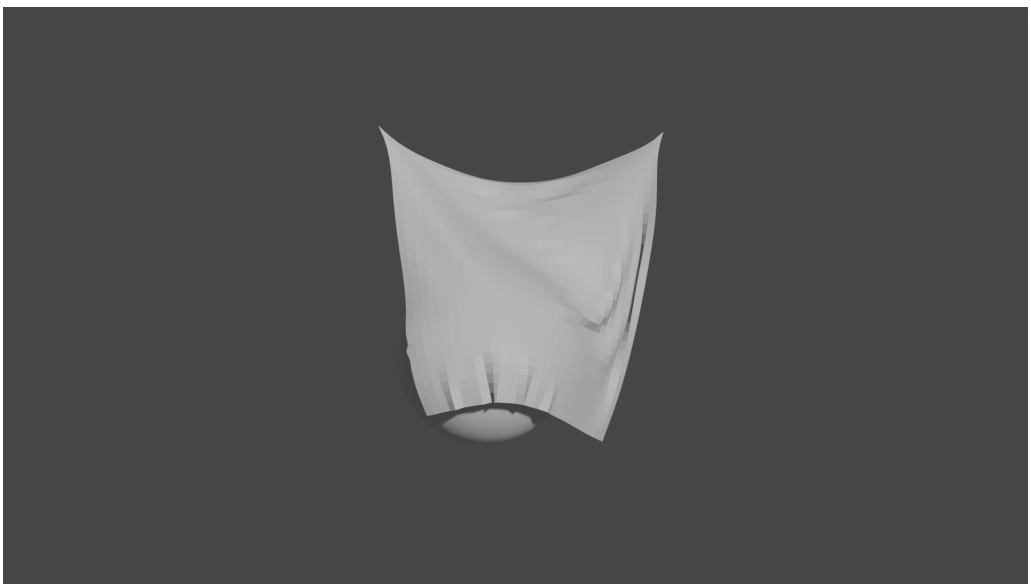
5.6 Rezultantne animacije

Ova sekcija prikazuje nekoliko okvira rezultantnih animacija dobivenih pokretanjem simulacija:

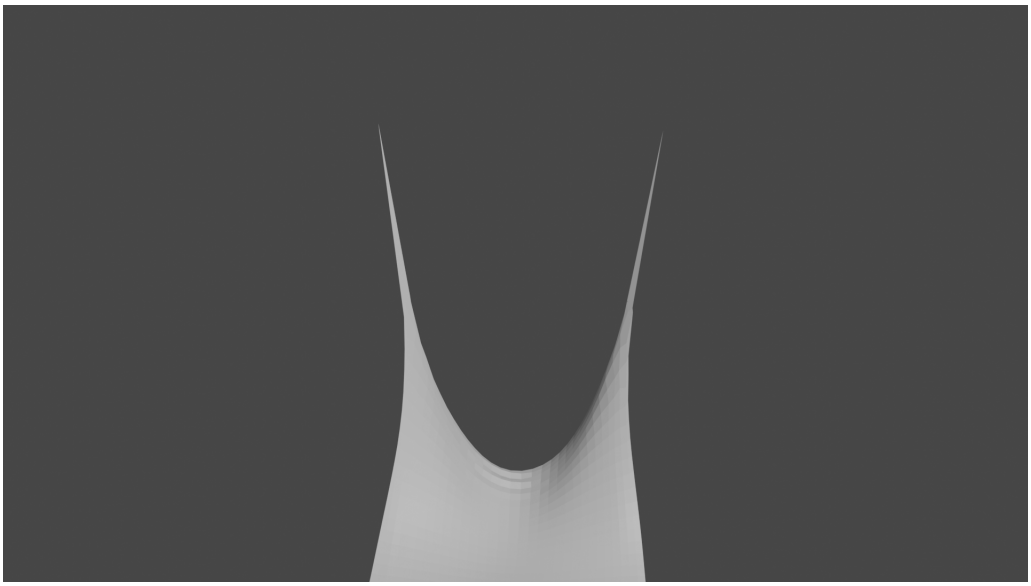
1. Gravitacijske sile sa fiksirane tri točke (5.8)
2. Gravitacijske sile i sudara sa sferom sa fiksirane dvije točke (5.9)
3. Gravitacijske sile i sile vjetra i problema super-elastičnosti (implementacija strukture sa listama parametara 6.1) (5.10)
4. Gravitacijske sile i sile vjetra i problema super-elastičnosti (implementacija strukture sa pomoćnim strukturama 5) (5.11)



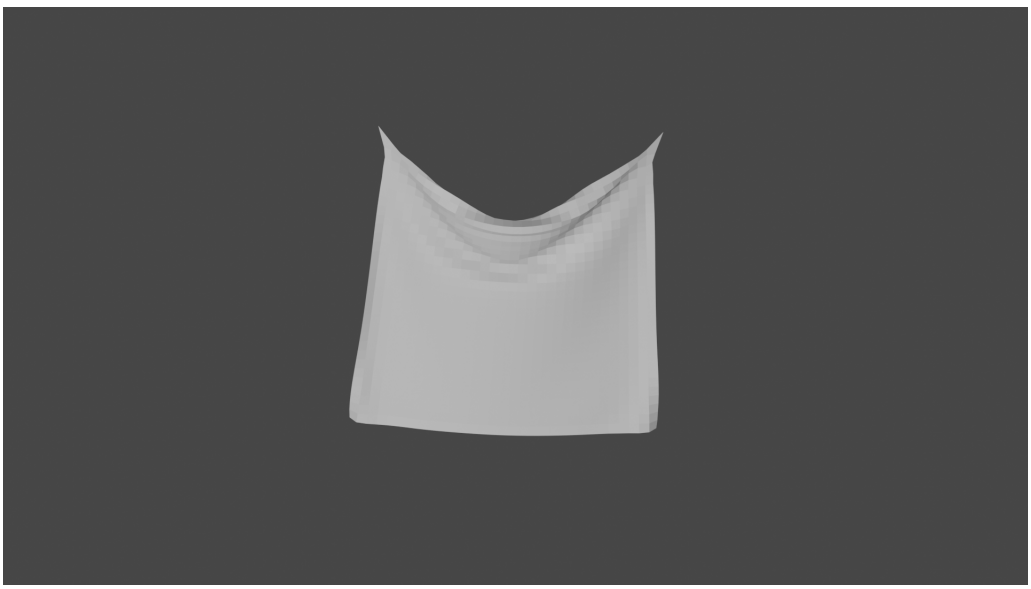
Slika 5.8 Okvir simulacije gdje na tkaninu utječe gravitacijska sila



Slika 5.9 Okvir simulacije gdje na tkaninu utječe gravitacijska sila i sudar sa sferom



Slika 5.10 Okvir simulacije gdje na tkaninu utječe gravitacijska sila i sila vjetra (prikaz problema super-elastičnosti (implementacija strukture sa listama parametara 6.1) 6.2)



Slika 5.11 Okvir simulacije gdje na tkaninu utječe gravitacijska sila i sila vjetra (prikaz problema super-elastičnosti (6.2) pri čemu je korištena implementacija strukture sa pomoćnim strukturama 6.1)

Poglavlje 6

Zapažanja i problemi pri implementaciji

Pri prvotno implementiranoj strukturi modela tkanine (prikazana u sekciji 6.1) došlo je do više problema pri izračunima koje je bilo gotovo nemoguće otkriti zbog ograničenja koje je Blender postavio pri izvršavanju programa. Iz toga razloga ukazala se potreba za restrukturiranjem jezgre klase *Cloth* te je implementirana prethodno viđena struktura (5). Ovo poglavlje približava razliku te dvije strukture i pokušava ukazati na moguće probleme kod prethodne.

6.1 Zapažene razlike pri implementaciji klase *Cloth* pomoću lista elemenata naspram implementacije pomoću lista pomoćnih klasa *Constraint* i *Particle*

Inicijalno, pri implementaciji klase *Cloth* ideja je bila da sama klasa sadrži sve potrebne metode i liste parametara koje su joj potrebne za izvođenje izračuna i unaprijeđivanje simulacije. Ta struktura u grubo izgleda ovako:

1. *object* - Parametar koji sadrži cijeli objekt koji je proslijeđen klasi (sama rav-

nina).

2. *particles* - Lista namijenjena pohranjivanju čestica sustava. Pohranjuju se koordinate njihovih pozicija (`<class 'bpy.types.MeshVertex'>`)
3. *particlesOld* - Lista namijenjena pohranjivanju čestica sustava iz prethodne iteracije
4. *forces* - Lista namijenjena pohranjivanju sila koje dijeluju na čestice sustava, te prema indeksima odgovaraju listama particles i particlesOld. Pohranjuju se vektori (`<class 'Vector'>`)
5. *grounded* - Lista namijenjena pohranjivanju podatka koja čestica je fiksirana, odnosno koja je slobodna. Sastoji se od jedinica i nula, pri čemu jedinica označava fiksiranu česticu a nula slobodnu česticu. Također odgovara listi *particles* prema indeksima
6. *constraints* - Lista namijenjena pohranjivanju podataka tipa *Constraint* (5.4) koji predstavljaju opruge sustav

te sadrži metodu *simulate()* koja prema potrebi poziva metode: *accumulateForces()* , *handleBallCollision()*, *update()*, *satisfyConstraints()* te *handleDeformationRate()*.

Sve metode su strukturno iste kao u trenutnoj implementaciji, jedino se razlikuju prema pristupu podacima na koje se odnose i koje izmjenjuju. Umjesto pristupa elementu tipa *Particle* (koji sadrži parametre prikazane u sekciji 5.5) i pozivu metode na razini toga elementa, podaci o trenutnoj poziciji čestice, njenoj prethodnoj poziciji, statusu slobode i sile koje djeluju na nju sadržavali su se u prigodno nazvanim listama te im se pristupalo preko istog indeksa. Primjerice, kod izračuna Verlet integracije za dobivanje nove pozicije čestice x_i, uzimali bi se podaci iz liste *particles*, *particlesOld*, *grounded* i *forces* pod indeksom **i** koji označava tu česticu.

Iz primjera (5.10 i 5.11) je vidljivo koliko stabilnost takve implementacije ovisi o *točnom* rukovanju i znanju koja čestica zauzima koji indeks te mogućnosti pogreške koju takva implementacija pruža ako slučajem dođe do "zabune" oko indeksa. Pretpostavlja se da je to mogao biti uzrok nestabilnosti sustava pri uvođenju sile vjetra u prethodnoj strukturi, prikazano na slici 5.10.

Kako je praćenje te količine indeksa iznimno zahtjevno za čovjeka, odlučilo se za

reimplementaciju sa puno jasnijim pristupom i pohranom. Nova implementacija je stabilizirala sustav, iako ne u potpunosti zbog i dalje vidljivih deformacija pri fiksiranim rubovima te se čini da je pretpostavka da je do problema došlo pri pristupanju podacima preko indeksa moguće točna.

Također, dobro je napomenuti da prilikom izvođenja simulacije, Blender prikazuje podatak o trajanju svake iteracije, te je nova struktura uspjela poboljšati vrijeme rendera iako nekonzistentno zbog računalne opreme.

6.2 Problem super-elastičnosti

Elongacija opruga koje su izravno vezane za fiksirane čestice vrlo je visoka u usporedbi sa svim ostalim oprugama. Deformacija je stoga vrlo lokalno koncentrirana i razina deformacije brzo opada s udaljenošću opruga i fiksiranih čestica. Vrijednost deformacije najizduženijih opruga prelazi čak 100%. Sličan fenomen učestalo se pojavljuje kod polimera budući da se radi o materijalima podložnim vrlo visokim razinama elastične deformacije. To ponašanje predstavlja problem u realističnosti modela zato što elastičnost tkanine često nije linearna, već se smanjuje proporcionalno povećanju razine deformacije (što može dovesti i to paranja tkanine u stvarnome svijetu) [13].

Do navedene deformacije u ovome radu dolazi zbog ne tretiranja rubnih točaka, odnosno fiksiranih čestica tkanine, u modelu. Sam rubni uvjet nije ispravno implementiran.

Maksimalna razina deformacije većine tkanina je oko 10% [13] te je takva vrijednost i uzeta u obzir pri implementaciji metode *handleDeformationRate()* (5.4.2).

Poglavlje 7

Zaključak

Razumijevanje fizičkih svojstava i ponašanja trodimenzionalnog svijeta koji nas okružuje nužno je i korisno za svih, makar se ne bavili simuliranjem istoga. Iskustvo nam navodi kakvo ponašanje očekivati i pretpostavljati od svakodnevnih predmeta koje percipiramo i s kojima smo u međuodnosu, ali zašto se ta ponašanja uopće događaju? Koja pravila diktiraju gibanje neživih objekata? Izvođenjem ovog rada dolazimo do potrebe razumijevanja matematičkog objašnjenja kretnje jednog od takvih predmeta, tekstila, koje inače uzimamo zdravo za gotovo. Tako pobliže sagledavamo trodimenzionalan sustav u kojem se nalazimo i kojeg percipiramo.

Tema ovog završnog rada je implementacija fizikalno temeljenog modela tkanine, uvođenje stvarnih utjecaja koji međudjeluju sa tkaninom te prikaz iste u obliku video uratka ili sekvence slika, što je i realizirano. Prikazalo se jednostavno snalaženje u trodimenzionalnom sustavu koje pruža Blender te manipulacija objekata u istom da poštuju stvarne zakone gibanja do granica koje računalna oprema podržava. Prikazani su matematički izrazi u kontekstu potreba simulacije tkanine, za koju je korištena numerička metoda Verlet integracije. Teorijski je objašnjena odluka korištenja Verlet integracije, te primjenom isplativost i jednostavnost koju pruža. Zaključno, prikazani su rezultati dobiveni pokretanjem implementirane simulacije, te razlika u korištenju različitih struktura modela.

Prikupljena su mnoga znanja, od korištenja moderne programske podrške za modeliranje i simuliranje Blender, korištenja programskog jezika Python u okuženju

Blendera, izvođenja vlastitih zaključaka na temelju prethodnih pretpostavka i zaključaka velikana računalne grafike i matematike poput Newtona i Verleta do vlastoručne implementacije istih i upoznavanja sa problematikama na koje je moguće pritom naići.

Prikazani su i dijelovi implementacije ne bi li bili pomoć i smjernica u narednim zahvatima. Glavni cilj rada bio je dobiti dovoljno stabilnu i vizualno zadovoljavajuću simulaciju tkanine u predodređenim stvarnim situacijama, te su proizvoljno odabrani utjecaj gravitacijske sile na tkaninu, utjecaj aerodinamičke sile (vjetar) te sudar tkanine sa krutim objektom. Povezivanjem svih utjecaja u jednu simulaciju gdje se tkanina ponaša slično očekivanom stvarnom ponašanju, možemo reći da je cilj rada ispunjen. Pritom je važno napomenuti više prepreka na koje se nailazilo prilikom implementacije, odnosno koliko prepisivanja (eng. *rewrite*-anja) je bilo potrebno za dobivanje zadovoljavajućeg rezultata. Naravno, i dalje postoje prepreke koje nisu savladane, poput problematike super-elastičnosti.

Ovaj rad rezultirao je uspješno implementiranim fizikalnim modelom tkanine, te može poslužiti kao osnova za daljnji rad na poboljšanju modela, posebice pri ispravljanju zapaženog problema super-elastičnosti vidljivog kod deformacije fiksiranih točaka tkanine.

Bibliografija

[1] Cs114 project 3: Cloth simulation using mass-spring system. , s Interneta, https://www.ics.uci.edu/~shz/courses/cs114/docs/proj3/index.html , rujan 2022.

[2] T. Stuyck, "Cloth simulation for computer graphics," *Synthesis Lectures on Visual Computing: Computer Graphics, Animation, Computational Photography, and Imaging*, vol. 10, no. 3, pp. 1–121, 2018.

[3] Službena blender dokumentacija - primitives. , s Interneta, https://docs.blender.org/manual/en/latest/modeling/meshes/primitives.html , rujan 2022.

[4] Službena blender stranica - about. , s Interneta, https://www.blender.org/about/ , rujan 2022.

[5] Službena blender dokumentacija - python api quickstart. , s Interneta, https://docs.blender.org/api/current/info_quickstart.html , rujan 2022.

[6] Eevee renderiranje. , s Interneta, https://docs.blender.org/manual/en/latest/render/eevee/index.html , rujan 2022.

[7] Vektor - wikipedija. , s Interneta, https://hr.wikipedia.org/wiki/Vektor , rujan 2022.

[8] 3d računalna grafika - wikipedija. , s Interneta, https://en.wikipedia.org/wiki/3D_computer_graphics , rujan 2022.

[9] Konačna razlike - upwikihr. , s Interneta, https://upwikihr.top/wiki/Finite_difference , rujan 2022.

[10] Rpi computer graphics research - lecture mass-spring systems - professor barbara cutler. , s Interneta, https://www.cs.rpi.edu/~cutler/classes/advancedgraphics/S12/lectures/06_mass_spring_systems.pdf , rujan 2022.

[11] T. Jakobsen, "Advanced character physics," in *Game developers conference*, vol. 3. IO Interactive, Copenhagen Denmark, 2001, pp. 383–401.

[12] A simple cloth simulator - yair yarom and ely levy (cs.huji.ac.il). , s Interneta, https://www.cs.huji.ac.il/w~irush/cloth/ , rujan 2022.

[13] X. Provot *et al.*, "Deformation constraints in a mass-spring model to describe rigid cloth behaviour," in *Graphics interface*. Canadian Information Processing Society, 1995, pp. 147–147.

[14] G. H. Elaine Kieran and L. Openshaw, "Cloth simulation." , s Interneta, https://nccastaff.bournemouth.ac.uk/jmacey/OldWeb/MastersProjects/Msc05/cloth_simulation.pdf

[15] Verlet integration. , s Interneta, https://www.algorithm-archive.org/contents/verlet_integration/verlet_integration.html , rujan 2022.

Sažetak

Ovaj rad prikazuje jednostavan način shvaćanja i implementacije fizičkog modela masa-opruga za simuliranje tkanine u programskoj podršci Blender, koristeći njegovo aplikacijsko programsko sučelje u jeziku Python. Također prikazuje problem super-elastičnosti i pritom prouzročene lokalizirane deformacije kod fiksiranih točaka tkanine, na koji simulacija nailazi pod utjecaj raznih sila. Pobliže objašnjava koncepte računalne grafike nužne za shvaćanje i snalaženje u radu u trodimenzionalnom okruženju, manipulaciju objektima i njihovim sastavnim elementima u fizički ispravnim okvirima te prikazu takve scene u obliku dvodimenzionalne animacije.

Ključne riječi **— simulacija tkanine, Eulerova integracija, Verlet integracija, model masa-opruga**

Abstract

This paper shows a simple way to understand and implement a physical mass-spring model for cloth simulation in the Blender software, using its Python application programming interface. It also touches on the problem of superelasticity and the resulting localized deformations at fixed points of the cloth, which the simulation encounters under the influence of various forces. It explains in more detail the concepts of computer graphics necessary to understand and navigate working in a three-dimensional environment, the manipulation of objects and their constituent elements in physically correct limits, and the presentation of such a scene in the form of two-dimensional animation.

Keywords **— cloth simulation, Euler integration, Verlet integration, mass-spring model**