SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 610

SIMULACIJSKI MODEL MEKIH TIJELA

David Čemeljić

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 610

SIMULACIJSKI MODEL MEKIH TIJELA

David Čemeljić

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 11. ožujka 2022.

ZAVRŠNI ZADATAK br. 610

Pristupnik:	**David Čemeljić (0036523170)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentorica:	prof. dr. sc. Željka Mihajlović

Zadatak: **Simulacijski model mekih tijela**

Opis zadatka:

Proučiti osnove u izradi modela mekih tijela. Proučiti postupke izrade fizikalno temeljenih simulacija. Razraditi simulacijski model za meka tijela. Ostvariti animaciju realiziranog simulacijskog modela za meka tijela. Načiniti testiranje na nizu primjera. Analizirati i ocijeniti ostvarene rezultate. Diskutirati upotrebljivost ostvarenih rezultata kao i moguća proširenja. Izraditi odgovarajući programski proizvod. Rezultate rada načiniti dostupne putem Interneta. Radu priložiti algoritme, izvorne kodove i rezultate uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 10. lipnja 2022.

Sadržaj

Uvod

Većina današnjih fizikalnih simulacija i pogona simulira fiziku krutih tijela (engl. *rigid body*). Kruta tijela podrazumijevaju da se lokacije vrhova modela ne mijenjaju odnosno kruta tijela se ne mogu deformirati.

Simulacije mekih tijela omogućavaju promjenu oblika tijela temeljenu na fizikalnim zakonima. Ovo područje je vrlo široko te obuhvaća simulacije tkanine, organskih tijela, kose/dlake itd. Većina razvijenih modela simulacije mekih tijela nisu velike preciznosti, njihova ciljana uporaba je u filmovima i videoigrama te u modnoj industriji za realističan prikaz odjeće prije nego što je proizvedena. Postoje i preciznije simulacije i simulacije u stvarnom vremenu.

Kroz ovaj rad prvo će biti objašnjeni različiti modeli simulacija a zatim će biti objašnjena implementacija simulacije korištenjem sustava masa i opruga. Na kraju će se analizirati ostvareni rezultati te diskutirati upotrebljivost i moguća proširenja.

1. Osnove dinamike mekih tijela

Dinamika mekih tijela (*engl. soft-body dynamics*) područje je računalne grafike koje se bavi tijelima koja se mogu deformirati u interakcijama s ostalim tijelima u sceni, samim sobom te pod utjecajem vanjskih sila poput vjetra u simulaciji tkanine. Potrebno je simulirati fiziku te kolizije tako da tijelo svejedno zadrži dio svojeg oblika. To se pokazalo znatno kompliciranijim od dinamike krutih tijela (*engl. rigid-body dynamics*). Postoji nekoliko ideja kako pristupiti problemu mekih tijela, ali još uvijek ne postoji idealno rješenje za simulacije u stvarnom vremenu. Glavni razlog tome je što su postojeći algoritmi računalno previše zahtjevni za simulacije u stvarnom vremenu ako želimo visoku kvalitetu tako da postojeće implementacije moraju koristiti nisku razinu detalja u simulaciji.

U ovom poglavlju bit će objašnjene postojeći modeli za ostvarivanje simulacija mekih tijela.

2. Postojeći algoritmi i principi simulacija mekih tijela

U ovom poglavlju bit će dan pregled postojećih algoritama i opisani generalni koncepti koji se koriste u njima.

2.1. Model masa - opruga

Model masa-opruga funkcionira tako da po modelu postavi čestice te ih poveže oprugama. Česticama se dodaje masa te se za svaku česticu simulira fizika.

Sile koje djeluju na svaku česticu su elastična sila od nekoliko opruga, ovisno o broju čestica, sila teža, sila prigušenja te unutarnja pritisna sila.

Elastične sile koje stvaraju opruge računaju se pomoću $F = kx$ između susjednih masa. Konstanta k je ručno odabrana.

Sila teža se računa za svaku česticu pomoću formule $F = mg$.

Sila prigušenja služi za postepeno stabiliziranje čestica tako da nakon nekog vremena dođu u pozicije određene oblikom tijela.

Unutarnja pritisna sila sprječava da model kolabira. Obično se računa pomoću jednadžbe idealnog plina [1]. Potrebno je izračunati površinu i volumen tijela te uvrstiti u jednadžbu pritisne sile.

Ovaj model bit će detaljno objašnjen u poglavlju s implementacijom.

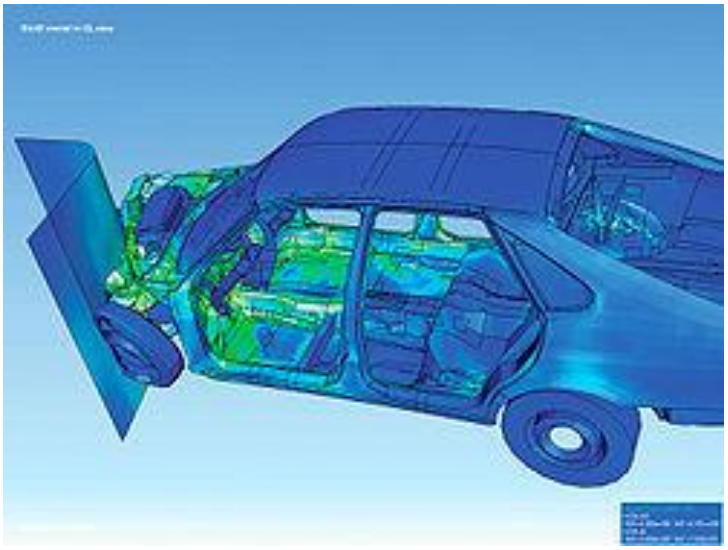
2.2. Model simulacije s konačnim brojem elemenata

Ovaj model (*engl. finite element method, FEM* [2]) baziran je na računanju parcijalnih diferencijalnih jednadžbi. Objekt se diskretizira u velik broj manjih elemenata – poddomene, svaka poddomena ima svoj skup jednadžbi. Na ovaj način velik problem se podijeli u mnogo manjih. Na kraju se skupovi jednadžbi kombiniraju u ukupni sustav jednadžbi te se pronađe rješenje.

FEM se koristi za razne tipove problema, poput simulacija deformacija, simulacija napetosti te aerodinamike, u medicini, autoindustriji.

FEM uzima u obzir velik broj parametara materijala koji simulira, npr. može simulirati deformaciju metala pod određenom temperaturom, utjecaj elektromagnetskih polja, fluide itd.

S obzirom na to da ovakva metoda detaljno simulira ponašanje uzimajući u obzir velik broj parametara, računanje sustava je vrlo računski zahtjevno te je daleko od stvarnog vremena, ali daje najpreciznije rezultate. Koristi se kod testiranja novih proizvoda prije nego što su proizvedeni prototipovi poput automobila na slici Slika 2.1.



Slika 2.1 FEM simulacija sudara automobila sa zidom [5]

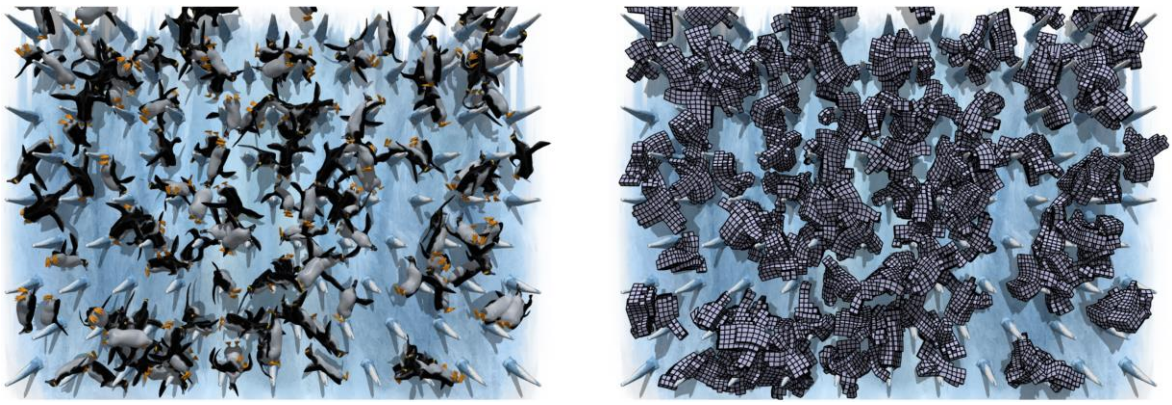
2.3. Deformacija temeljena na krutim tijelima

Deformacije tijela simuliraju se uz pomoć krutih tijela s ograničenjima. Oko tih krutih tijela generira se model koji se obavija oko njih.

2.4. Model podudaranja oblika

Model podudaranja oblika (*engl. shape matching*) je aproksimacija modela s konačnim brojem elemenata. Generalni koncept je pamćenje originalnog oblika tijela kojem tijelo u svakom trenutku teži, kao da je načinjeno od memorijske pjene.

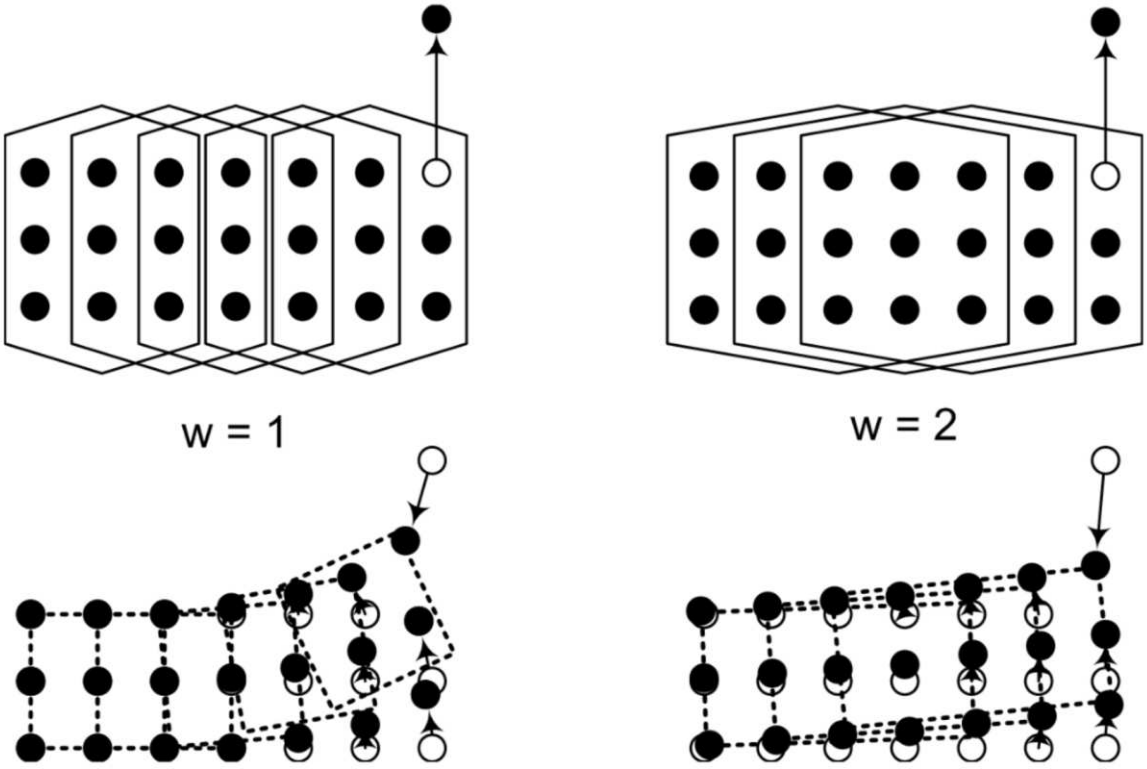
2.4.1. Metoda Fast Lattice Shape Matching (FastLSM)



Slika 2.2 Simulacija mekih tijela korištenjem FastLSM algoritma [3]

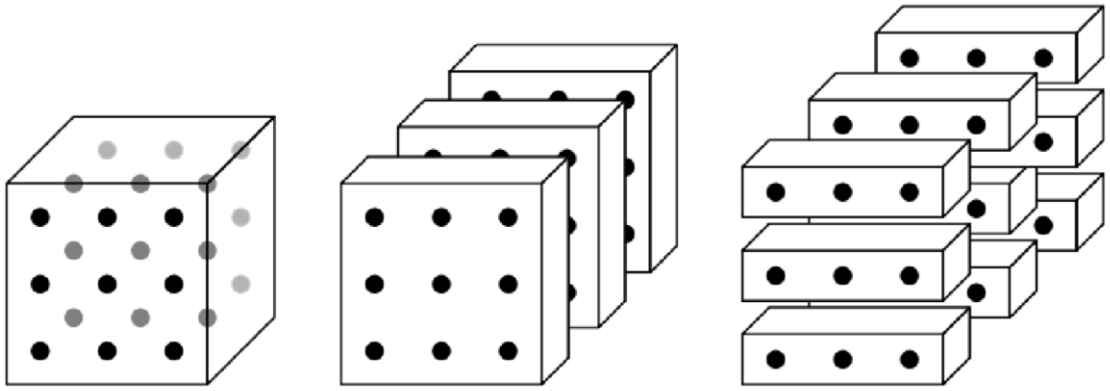
FastLSM [3] dijeli tijelo na određeni broj čestica tako što se tijelo vokselizira te se u centar svakog voksela postavi čestica (Slika 2.2). Oko čestica generiraju se regije širine određene parametrom *w* kao što je prikazano na slici Slika 2.3. Širina regije određuje koliko je tijelo savitljivo odnosno kruto.

U izračunu novih lokacija čestica metoda FastLSM uzima u obzir ciljnu poziciju i vanjske sile. Vanjske sile su npr. sila reakcije podloge kod sudara te sila teža.



Slika 2.3 Prikaz regija i ponašanja ovisno o parametru *w*

FastLSM definira nekoliko optimizacija kojima se dobije složenost O(1). Regije se dijele u supsumacije *bar-plate-cube* (Slika 2.4). Supsumacije je mali dio izračuna nekog atributa za sve čestice, npr. računanje brzine za neku regiju. S obzirom na to da se regije uvijek djelomično preklapaju, imaju neke zajedničke supsumacije. Stoga prilikom prvog računanja supsumacije spremamo i koristimo izračunatu sumu za druge regije koje sadržavaju tu supsumaciju.



Slika 2.4 Podjela regija u supsumacije *bar-plate-cube*

Algoritam također definira izračun rotacija uz pomoć polarne dekompozicije matrice A. Polarna dekompozicija računa se dijagonalizacijom umnoška transponirane matrice A i same matrice A uz pomoć Jacobijevih iteracija te spremanjem vlastitih vektora (*engl. eigenvector*) za svaku regiju za svaki okvir.

Kako bi se sustav stabilizirao, potrebno je uvesti usporavanje (*engl. damping*). Usporavanje se za svaku regiju uz korištenje principa supsumacija.

Metoda FastLSM također podržava frakturiranje. Frakturiranje se događa kad razmak između susjednih čestica postane prevelik.

3. Implementacija sustava masa i opruga

U ovom poglavlju bit će opisana implementacija jednostavnog sustava koji simulira deformacije pomoću masa povezanih oprugama. Sustav masa i opruga prema dizajnu ima nekoliko mana koje će također biti opisane u sljedećim poglavljima.

Cijeli kod ove aplikacije dostupan je na

https://gitlab.com/davidcemeljic/springmasssimulation

3.1. Koncept masa i opruga

Tijelo koje simuliramo sastoji se od masa povezanih oprugama. Mase su simulirane česticama. Broj čestica i njihove lokacije definirane su lokacijama vrhova modela koji simuliramo, dakle što je veći model, algoritam je skuplji.

Za svaku česticu simulira se fizikalno ponašanje te računa nova lokacija za svaki okvir. Opruge ne postoje fizički u simulatoru već čestice pamte s kojim ostalim česticama su povezane te originalne udaljenosti od svojih povezanih čestica. Polje susjednih čestica se popunjava prilikom konstruiranja mekog tijela kako bi se mogla izračunati razlika u udaljenosti za elastičnu silu.

Ovako definirano tijelo možemo usporediti s komadom tkanine. Površina je elastična te se uvijek vraća u početno stanje napetosti. No problem je što će se tijelo spljoštiti na tlo. Potrebna je neka sila koja će održavati tijelo da se ne spljošti.

Pokazalo se da dodavanje još jedne čestice u središte tijela neće riješiti problem sam po sebi, ali svejedno je korisna.

Pravo rješenje problema je unutarnja pritisna sila temeljena na idealnom plinu. U ovom radu prva ideja je bila sljedeća: unutarnja pritisna sila je aproksimirana silom koja djeluje na svaku česticu u smjeru njezine originale normale, tj. izračunate normale vrha prilikom kreiranja mekog tijela. Pokazalo se da ova sila odlično vraća tijelo u originalni oblik, no tijelo se uvijek vraća u uspravni položaj zato što se normale ne mijenjaju, odnosno tijelo se ne može kotrljati. Računanje novih normala za svaki okvir također ne pomaže, tijelo se djelomično uruši i ne vrati u svoj originalni oblik. Iz tog razloga implementirana je skuplja metoda pomoću pritisne sile temeljene na idealnom plinu.

Na sve ovo također treba dodati prigušivanje (*engl. damping*) kako bi se sustav stabilizirao u razumnom vremenu.

3.2. Razred *Particle*

Čestice su prikazane razredom `Particle`. Svaka čestica ima definiran niz parametara koji se koriste kako bi se ostvarila simulacija. Razred `Particle` prikazan je isječkom Programski kod 1.

```
class Particle {
std::vector<std::pair<Particle*, float>>* initialDistancesToConnected
= new std::vector<std::pair<
Particle*, float>>();
std::vector<std::pair<Particle*, float>>*
initialDistancesToInnerConnected = new std::vector<std::pair<
Particle*, float>>();
public:
Mesh* parentMesh;
float* mass;
float* k;
float* k_inner;
float* c;
float* internalPressureConstant;

glm::vec3 normal;
glm::vec3 color = {0.0f, 1.0f, 0.0f};

glm::vec3 position = {0, 0, 0};
glm::vec3 velocity = {0, 0, 0};
glm::vec3 force = {0, 0, 0};


Particle(Mesh* parentMesh, glm::vec3 position, glm::vec3
initialNormal, float* mass, float* outerSpringConstant,
float* innerSpringConstant, float* dampingConstant, float*
pressureConstant):
parentMesh(parentMesh), mass(mass), k(outerSpringConstant),
k_inner(innerSpringConstant),
c(dampingConstant), internalPressureConstant(pressureConstant),
normal(initialNormal),
```

```
        position(position) { }

    void addConnected(Particle* connected);
void addInnerConnected(Particle* innerConnected);

    void applyPhysics(float deltaTime);

    inline void calculateSpringForces();
inline void calculateInternalPressureForce(float meshSurfaceArea,
                    float meshVolume);
    inline void calculateDampingForces();

    void applyForce(float deltaTime);};
```

Programski kod 1 *Header file* razreda `Particle`

Najvažniji parametri čestice su njezina lokacija, vektor brzine i vektor sile. Parametri `k` i `k_inner` su konstante opruge za susjedne čestice i za središnju česticu. Parametar `c` je konstanta koja se koristi za usporavanje. Konstante za računanje sila su pokazivači kako bi se jednostavno mogli mijenjati kroz grafičko sučelje aplikacije.

Metode `void addConnected(Particle* connected)` i `void addInnerConnected(Particle* innerConnected)` su pomoćne metode koje se koriste prilikom konstruiranja mekog tijela. One popunjavaju polja `initialDistancesToConnected` te `initialDistancesToInnerConnected` računajući trenutne udaljenosti od predanih čestica.

Ostale metode služe za računanje sila i primjenjivanje fizike na česticu odnosno računanje nove lokacije.

3.3. Fizika

Fizika se računa za svaki okvir za svaku česticu. Nakon što se izračunaju rezultante sile za svaku česticu, računaju se nove lokacije svih čestica ovisno od duljini prikaza okvira (*engl. frame time, delta time*). Nove lokacije računaju se preko drugog Newtonovog zakona, odnosno izračuna se akceleracija preko ukupne sile koja djeluje na česticu i mase čestice, iz akceleracije i duljine prikaza okvira računa se brzina, a iz brzine i duljine prikaza okvira

nova lokacija čestice. Opisano ponašanje implementirano je metodom u isječku Programski kod 2.

```
void Particle::applyForce(float deltaTime) {
    // F = m * a
    // a = F / m
    const glm::vec3 acceleration = force / *mass;
    // v = v0 + a * t
    velocity += acceleration * deltaTime;

    // clamp velocity to 10
    if (length(velocity) > 10.0f)
        velocity = normalize(velocity) * 10.0f;

    // x = x0 + v * t
    position += velocity * deltaTime;

    // reset force for next iteration
    force = glm::vec3(0.0f);
}
```

Programski kod 2 Metoda koja primjenjuje silu na česticu, odnosno računa njenu novu lokaciju

3.3.1. Razred *Force*

Za potrebe definiranja i korištenja vanjskih sila, koristi se bazni razred `Force` prikazan isječkom Programski kod 3. Razred definira metodu `apply` koja računa i dodaje iznos sile svakoj čestici nekog mekog tijela.

```
class Force {
public:
    virtual void apply(std::vector<Particle*> particles) = 0;

    virtual ~Force() = default;
};
```

Programski kod 3 Bazni razred *Force* koji implementiraju vanjske sile

3.3.2. Sila teža

Sila teža je vanjska sila koja se računa za svaku česticu jednako preko formule (1)

$$\overrightarrow{F_g} = m\vec{g} \quad (1)$$

gdje je *m* masa čestice a $\vec{g}$ je vektor gravitacije definiran kao {0, -9.81, 0}.

Vrijednost sile se izravno dodaje na trenutnu vrijednost sile čestice bezuvjetno.

Kod metode `apply` prikazan je u isječku Programski kod 4.

```
void GravityForce::apply(std::vector<Particle*> particles) {
    for (auto p : particles)
        p->force += p->mass * gforce;
}
```

Programski kod 4 Metoda `apply` razreda koji računa silu težu

3.3.3. Sila reakcije podloge

Sila reakcije podloge djeluje na česticu kad se ona nalazi na podlozi. Ona je također vanjska sila, ali njen iznos nije jednak za svaku česticu.

Podloga je definirana kao ravnina zadana točkom i vektorom normale. Konstanta D se računa kao negativni skalarni produkt točke i normale.

U slučaju da je točka na podlozi ili vrlo blizu podloge, pod uvjetom da je još uvijek iznad podloge, sila reakcije podloge računa se izrazom (2).

$$\overrightarrow{F_r} = \vec{n} * \overrightarrow{F_R} \cdot (-\vec{n}) \quad (2)$$

$\vec{n}$ predstavlja normalu ravnine, a $\overrightarrow{F_R}$ predstavlja ukupnu silu koja djeluje na česticu za koju računamo silu reakcije podloge. Dakle, iznos sile reakcije podloge je projekcija vektora ukupne sile koja djeluje na česticu na suprotni vektor normale. Iznosu dajemo smjer vektorom normale ravnine. Na ovaj način rezultanta sila koja djeluje na česticu u smjeru kolinearnom vektoru normale postaje 0.

U slučaju da je čestica prošla kroz podlogu, treba je vratiti na podlogu. Trenutnoj lokaciji čestice dodaje se vektor $-distance * \vec{n}$ te se njezina brzina postavlja na 0.

Metoda kojom se računa sila reakcije podloge prikazana je isječkom Programski kod 5.

```
RigidBodyCollisionForce::RigidBodyCollisionForce(glm::vec3
position, glm::vec3 direction) {
    this->position = position;
    this->direction = normalize(direction);
    D = -dot(direction, position);
}

void RigidBodyCollisionForce::apply(std::vector<Particle*>
particles) {
    for (auto particle : particles) {
        float distance = dot(particle->position, direction) +
D;

        if (distance < 0.001f) {
            float forceProjectionMagnitude = dot(particle-
>force, -direction);
            particle->force += direction *
forceProjectionMagnitude;

            float velocityProjectionMagnitude = dot(particle-
>velocity, -direction);
            particle->velocity += direction *
velocityProjectionMagnitude;
        }
        if (distance < 0) {
            particle->position += direction * -distance;
            particle->velocity = glm::vec3(0);
        }
    }
}
```

Programski kod 5 Konstruktor i metoda apply razreda koji računa silu reakcije podloge

3.3.4. Međusobne elastične sile susjednih čestica

Međusobne elastične sile nisu vanjske sile te ne koriste razred `Force`. Svaka čestica ima nekoliko susjednih čestica i one međusobno djeluju elastičnom silom jedna na drugu.

Također postoji čestica u središtu koja koristi drugu konstantu opruge te se njezin utjecaj računa na isti način kao za susjedne čestice.

Elastična sila računa se prema izrazu (3).

$$\overrightarrow{F_{el}} = \frac{k * \Delta\vec{x}}{2} \quad (3)$$

k je konstanta opruge, $\Delta\vec{x}$ je razlika trenutne udaljenosti od susjedne čestice i originalne udaljenosti. Izraz je podijeljen s 2 zato što čestice jedna na drugu djeluju istom silom. U isječku Programski kod 6 prikazana je implementacija elastične sile u aplikaciji.

```
inline void Particle::calculateSpringForces() {
    for (auto initDistConnected :
*initialDistancesToConnected) {
        const float deltaX = initDistConnected.second -
distance(position, initDistConnected.first->position);
        // Fs = k * deltaX * 0.5 where k is spring constant
and deltaX is the difference between the initial distance and
the current distance as vector
        force += *k * deltaX * normalize(position -
initDistConnected.first->position) * 0.5f;
    }

    for (auto initDistInnerConnected :
*initialDistancesToInnerConnected) {
        const float deltaX = initDistInnerConnected.second -
distance(position, initDistInnerConnected.first->position);
        // Fs = k * deltaX * 0.5 where k is spring constant
and deltaX is the difference between the initial distance and
the current distance as vector
        force += *k_inner * deltaX * normalize(position -
initDistInnerConnected.first->position) * 0.5f;
    }
}
```

Programski kod 6 Metoda koja računa elastične sile između susjednih čestica

3.3.5. Unutarnja pritisna sila

Unutarnja pritisna sila je tu da bi tijelo zadržalo otprilike konstantni volumen. Pokazala se nužnom za sustav masa i opruga.

Unutarnja pritisna sila računa se izrazom (4) za svaku česticu.

$$\overrightarrow{F_p} = \frac{SnRT}{V} * \vec{d} \quad (4)$$

S predstavlja površinu tijela. Računa se kao suma površina svih trokuta od kojih je tijelo sastavljeno.

n je broj molova plina. Vrijednost 0.01 mol se pokazala idealnom.

R je univerzalna plinska konstanta, $8.3145\ J * mol^{-1}\ K^{-1}$

T je temperatura u Kelvinima, simulacija se pokreće na $25\ °C$ što je $298.15\ K$

$\vec{d}$ je normala čestice

V je volumen tijela. Volumen se računa na sljedeći način. Potrebno je pronaći središte tijela što je aritmetička sredina svih lokacija vrhova. Za svaki trokut uzima se središte tijela te iz tih informacija kreira tetraedar. Na ovaj način problem računanja kompleksnog tijela svodimo na računanje volumena tetraedara. Isječak Programski kod 7 prikazuje kako je izračunat volumen tetraedra u aplikaciji, a isječak Programski kod 8 kako je izračunata unutarnja pritisna sila za svaku česticu.

Volumen tetraedra računa se izrazom (5)

$$V_{tetraedar} = \frac{\left|(v0 - C) \cdot \left((v1 - C) \times (v2 - C)\right)\right|}{6} \quad (5)$$

$v0, v1, v2$ su vrhovi trokuta

C je središte tijela

```
static float tetrahedronVolume(const Triangle& triangle,
const glm::vec3& top) {
    return (1.0f / 6.0f) * glm::abs(dot(triangle.v0->position
- top, cross(triangle.v1->position - top, triangle.v2-
>position - top)));
}
```

Programski kod 7 Metoda za računanje volumena tetraedra unutar razreda `Util`

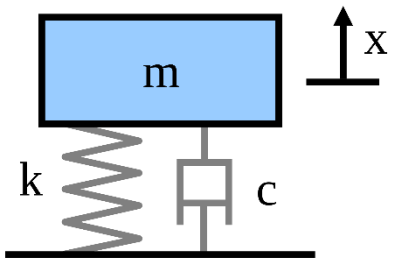
```
inline void Particle::calculateInternalPressureForce(float
        meshSurfaceArea, float meshVolume) {
            // Fp = normal * SnRT / V
force += normal * meshSurfaceArea * *internalPressureConstant * R * T
            / meshVolume;
                }
```

Programski kod 8 Metoda koja računa iznos unutarnje pritisne sile za česticu

Bitno je napomenuti da zbog ovakvog načina računanja volumena i unutarnje pritisne sile algoritam funkcionira samo na konveksnim modelima.

3.3.6. Prigušivanje (*engl. damping*)

Prigušivanje je nužno kako bi se sustav stabilizirao u prihvatljivom vremenu. Na slici Slika 3.1 prikazana je ideja prigušivanja opruge.



Slika 3.1 Prikaz amortizera [4]

Prigušenje se računa izrazom (6).

$$\overrightarrow{F_d} = -c\vec{v} \qquad (6)$$

c je konstanta prigušenja, a $\vec{v}$ je brzina čestice.

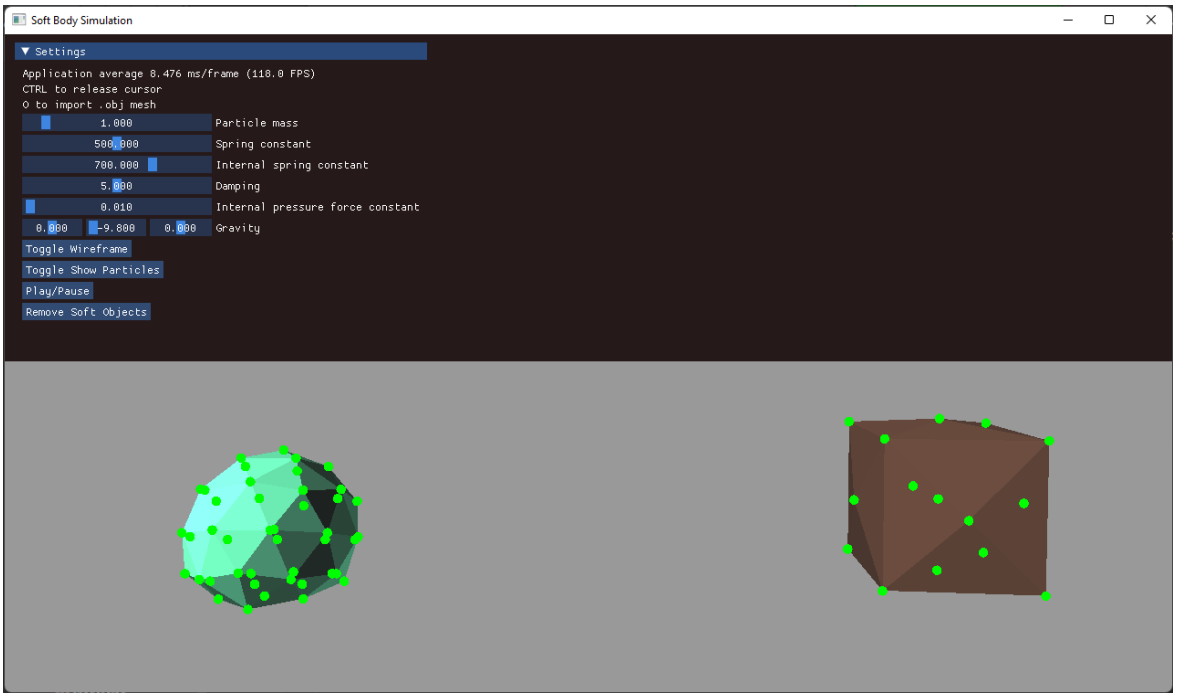
Isječak Programski kod 9 prikazuje metodu koja računa prigušenje za neku česticu.

```
inline void Particle::calculateDampingForces() {
      force -= *c * velocity;
}
```

Programski kod 9 Izračun prigušenja

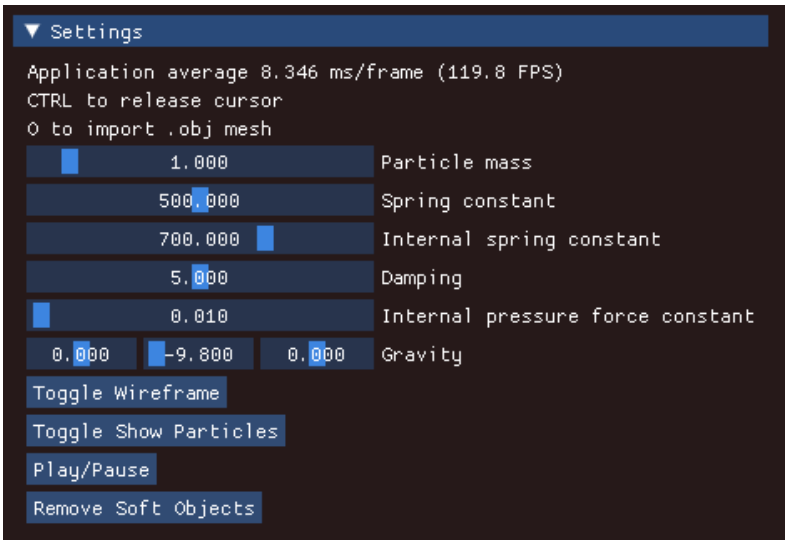
3.4. Implementacijski detalji

Simulator je izveden kao OpenGL 4.6.0 aplikacija pisana u programskom jeziku C++20. S obzirom na to da koristi Windows-ov *file explorer*, nije prenosiva i moguće ju je izvoditi samo na Windows OS. Slika 3.1 prikazuje izgled aplikacije. Koristi se konstantno sjenčanje uz jedno usmjereno svjetlo kako bi se bolje vidjeli trokuti prilikom simulacije. Zelenim točkama prikazane su čestice.



Slika 3.2 Izgled aplikacije

3.4.1. Grafičko sučelje



Slika 3.3 Grafičko sučelje aplikacije

Na slici Slika 3.3 prikazano je grafičko sučelje. Grafičko sučelje izvedeno je pomoću biblioteke Dear ImGui.

Sučelje se može micati te zatvoriti klikom na gumb pokraj naslova. Prikazuje vrijeme trajanja prikaza okvira (*engl. frame time*) i broj okvira u sekundi (*engl. framerate*).

Prilikom pokretanja aplikacije, kursor je nevidljiv i pomiče kameru. Klikom tipke *CTRL* kursor postaje vidljiv i moguća je interakcija s grafičkim sučeljem. Tipka *O* otvara *Windows File Explorer* kako bi se mogla odabrati *.obj* datoteka za uvoz. Uvoženjem datoteke stvara se novo meko tijelo u sceni ispred kamere.

`Particle mass` mijenja mase svih čestica. Preporučena vrijednost je 1, s obzirom na to da su ostale vrijednosti odabrane uzimajući u obzir vrijednost mase 1.

`Spring constant` je konstanta opruge koja se koristi za izračun elastičnih sila između susjeda.

`Internal spring constant` je konstanta opruge koja se koristi za izračun elastičnih sila između pojedine čestice i središnje čestice.

`Damping` je konstanta c za izračun prigušenja.

`Internal pressure force constant` je konstanta n kod izračuna unutarnje pritisne sile

Sučelje također omogućuje promjenu vektora sile teže.

Opcija `Toggle Wireframe` mijenja način iscrtavanja u žični.

`Toggle Show Particles` omogućuje prikaz s i bez čestica.

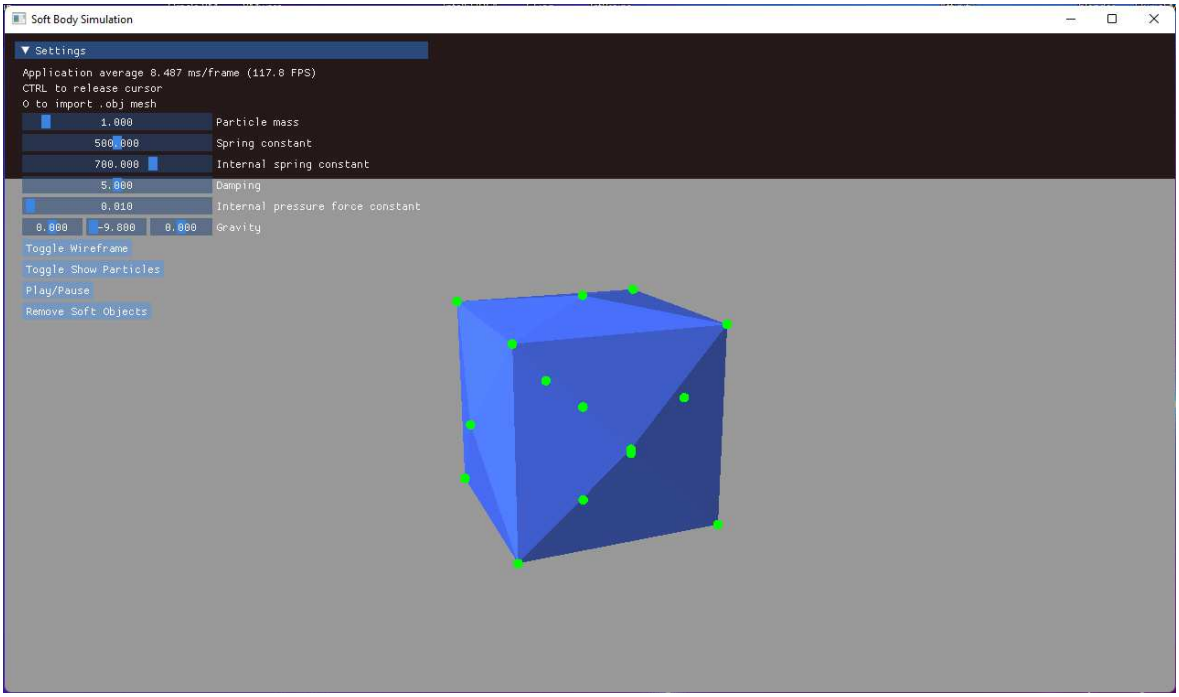
Tipka `P` pauzira simulaciju fizike u aplikaciji.

Tipka `Remove Soft Objects` miče sva meka tijela u simulaciji.

4. Rezultati

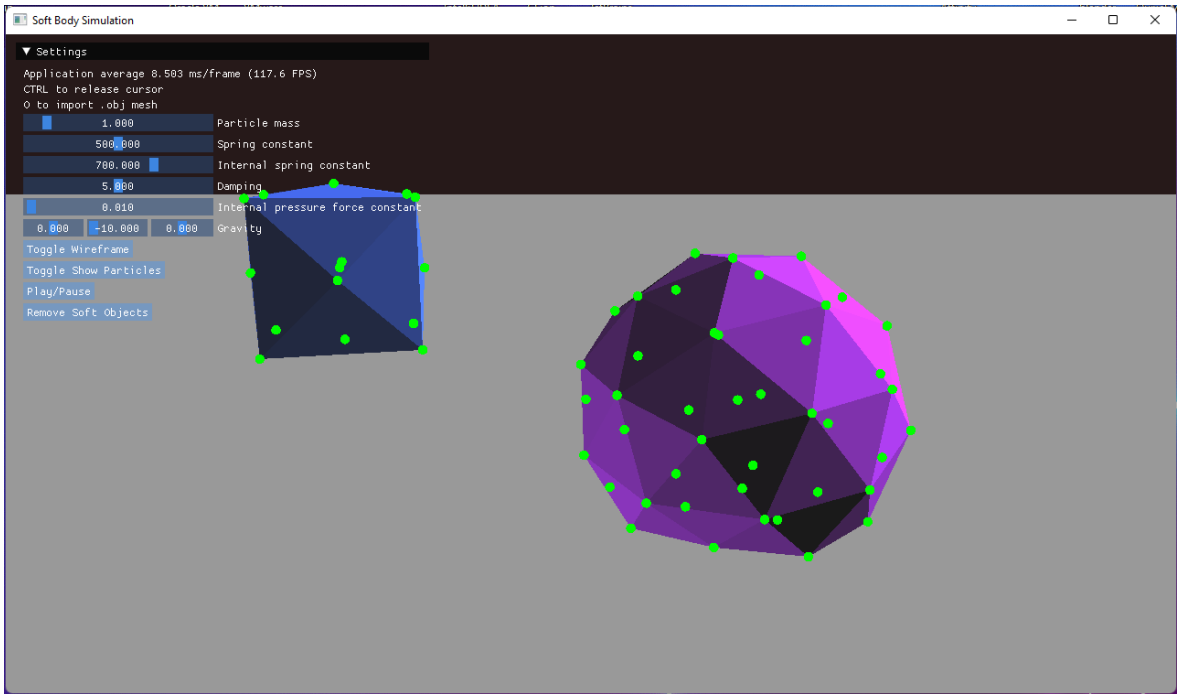
U ovom poglavlju bit će prikazani rezultati simulacije te problemi s algoritmom i ograničenja implementacije. Na kraju će biti opisane mogućnosti nadograđivanja simulatora. Također će biti prikazani rezultati za različite parametre simulacije.

4.0. Rezultati sa zadanim postavkama



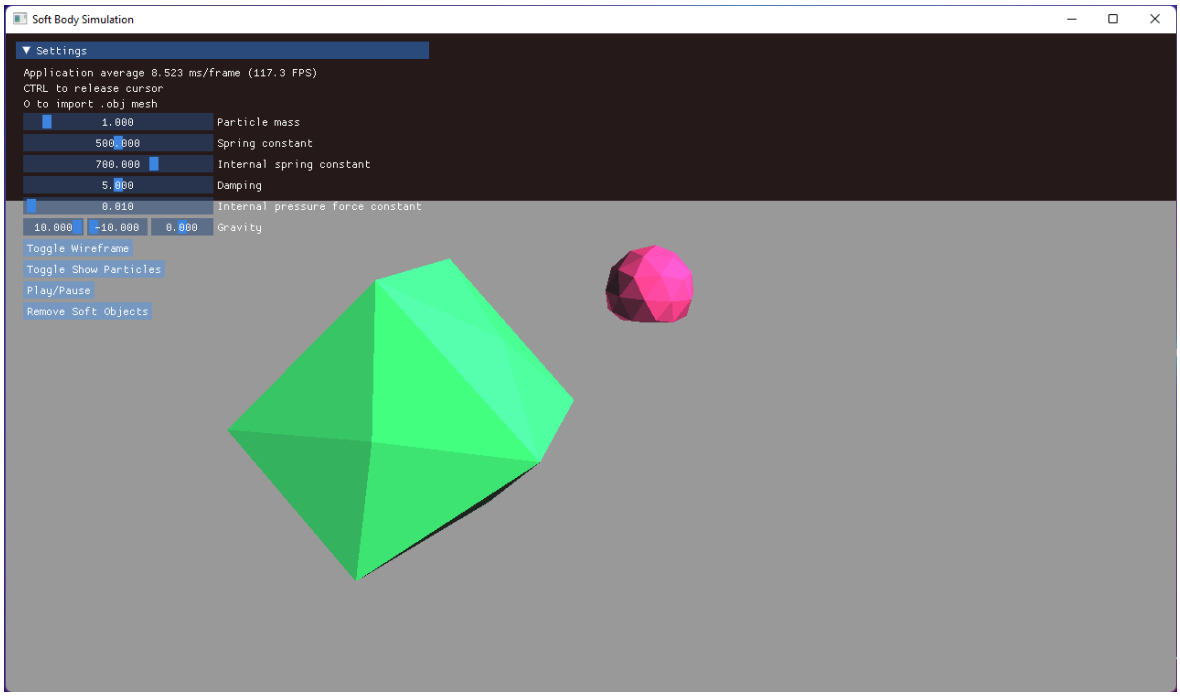
Slika 4.1 Kocka s dodatnim vrhovima u središtima svake stranice

Za zadane postavke, kocka sa slike Slika 4.1 brzo reagira na sudar s tlom odnosno napeta je.



Slika 4.2 Sfera uvezena u simulator

Sfera sa slike Slika 4.2 zapravo nije sfera već *Ico Sphere* iz *Blendera* što je zapravo ikosaedar s višom razinom detalja. Sfera se također odlično ponaša u simulatoru uz tvorničke postavke. Izgleda malo manje napeto, malo se više spljoštila od kocke.



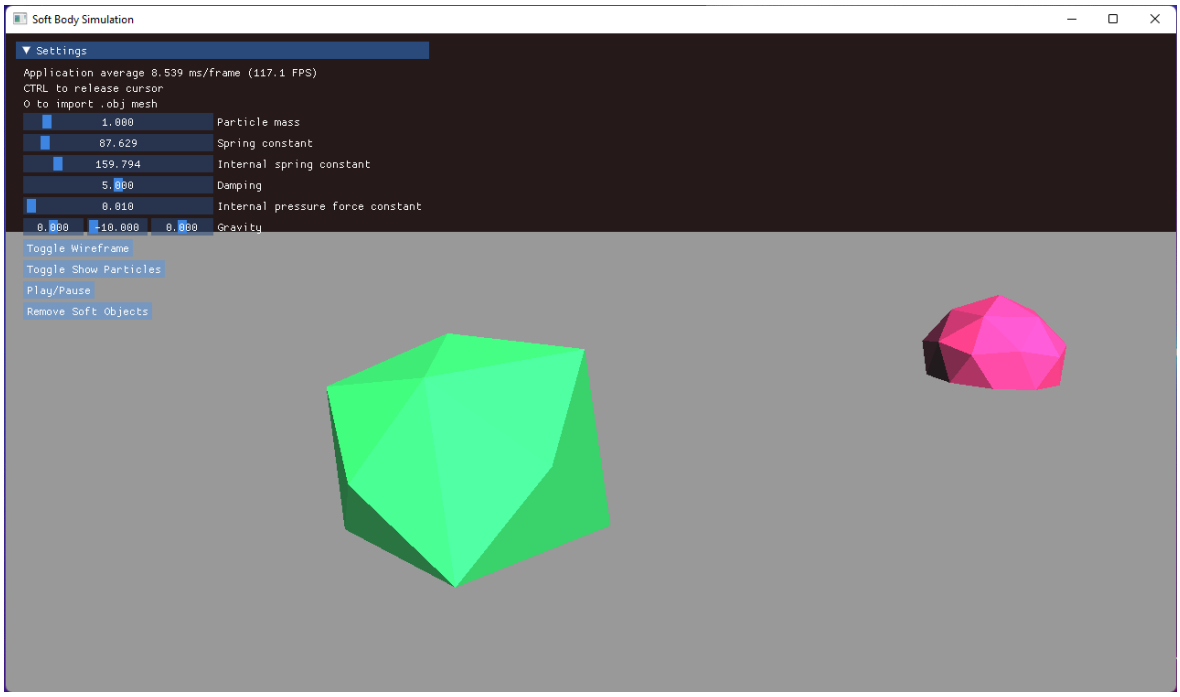
Slika 4.3 Kotrljanje kocke i sfere

Kotrljanje prikazano na slici Slika 4.3 dobiveno je promjenom vektora sile teže. Oba modela ostvaruju odlične rezultate.

Za kompleksnije modele, poput glave robota, algoritam je nažalost prespor.

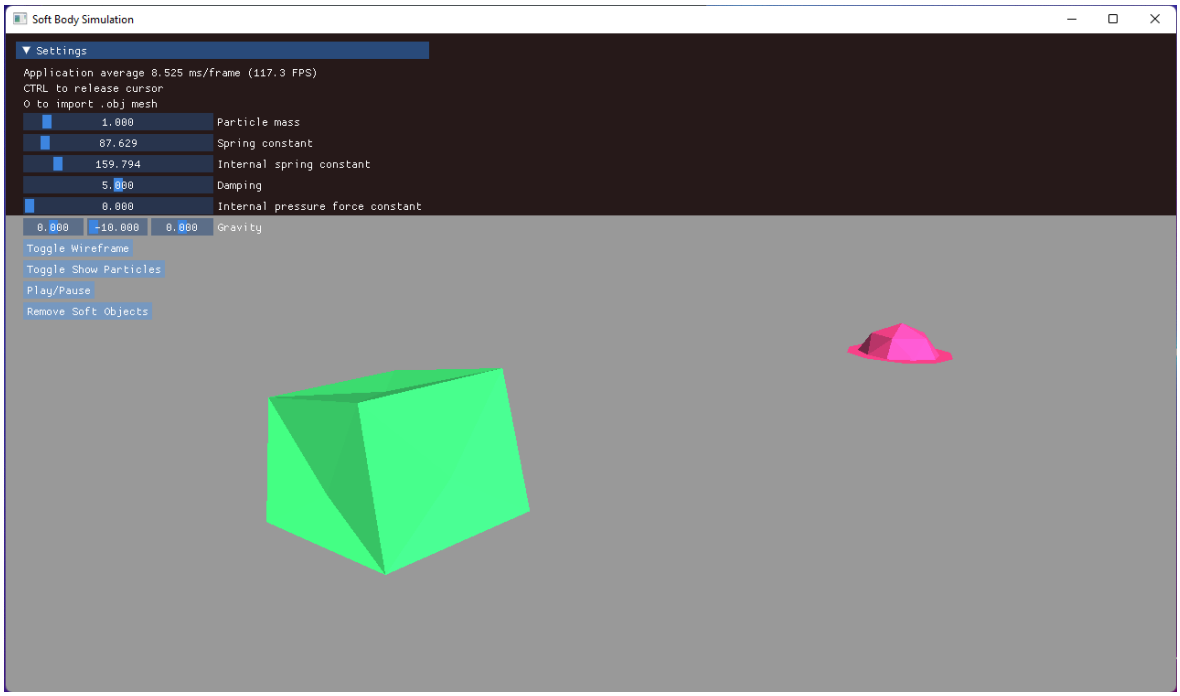
4.1. Rezultati uz promijenjene postavke

Ovisno o postavkama, tijela mogu biti plastična ili elastična. Uvećavanjem svih parametara dobiva se plastičnost, u suprotnom, elastičnost.



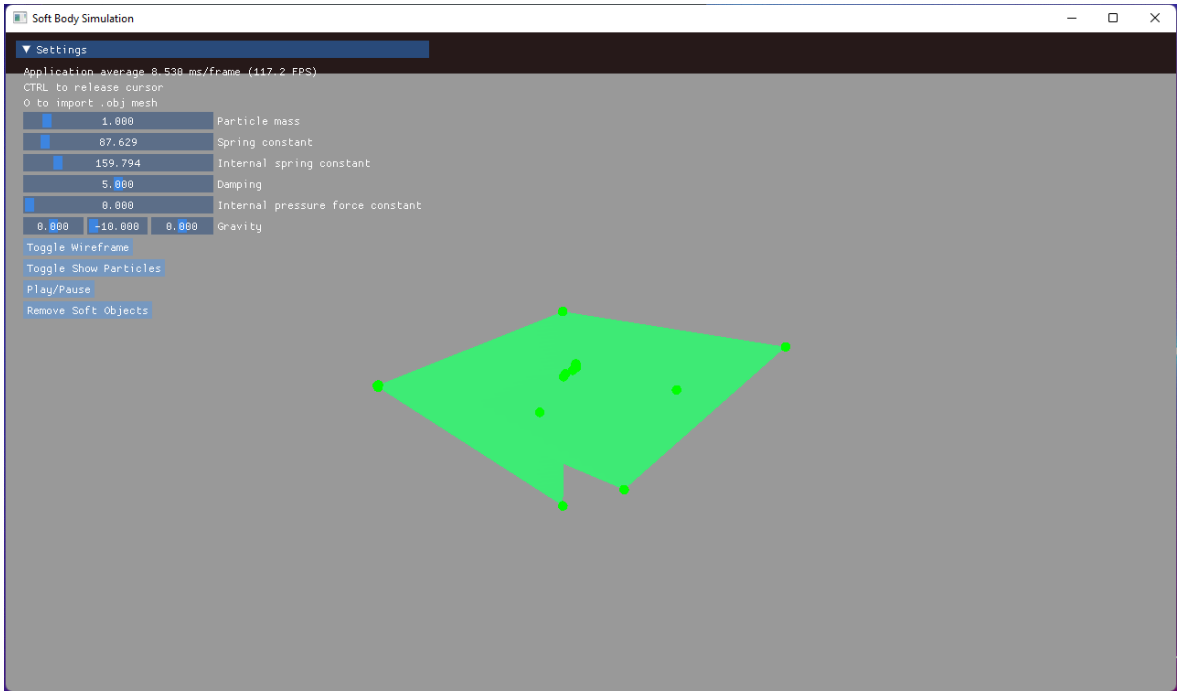
Slika 4.4 Niske vrijednosti konstanta opruga

Slika 4.4 prikazuje stanje simulacije za niske vrijednosti obje konstante opruga. Tijela su spljoštena, no i dalje djelomično zadržavanju svoj oblik zbog unutarnje pritisne sile.



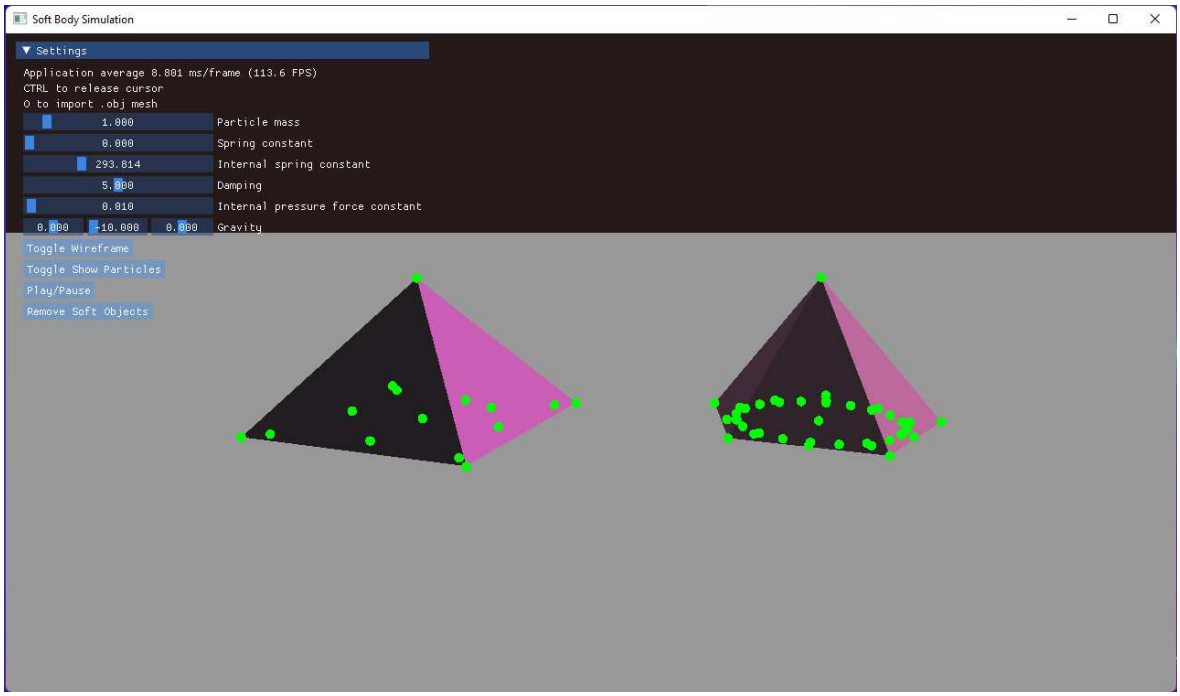
Slika 4.5 Simulacija bez unutarnje pritisne sile netom nakon onemogućavanja te sile

Ako maknemo i pritisnu silu, tijela se počinju urušavati. Ova situacija prikazana je na slici Slika 4.5.



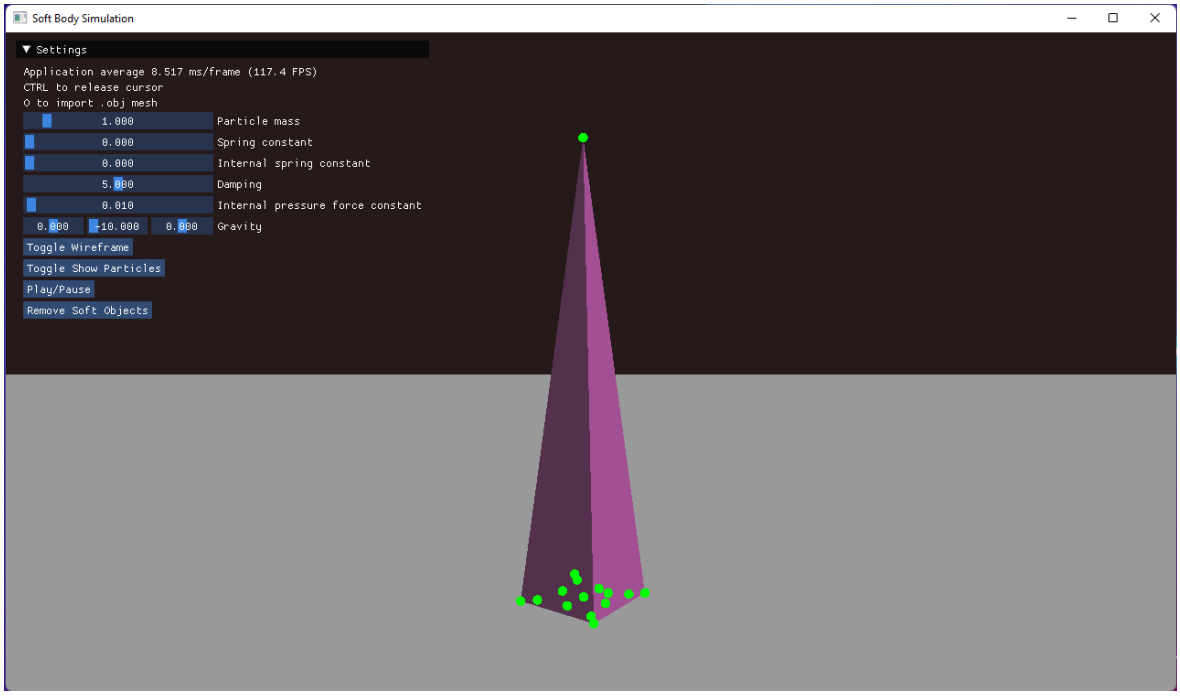
Slika 4.6 Kocka je spljoštena nakon nekog vremena

Nakon nekog vremena, tijela se spljošte u potpunosti kao na slici Slika 4.6.



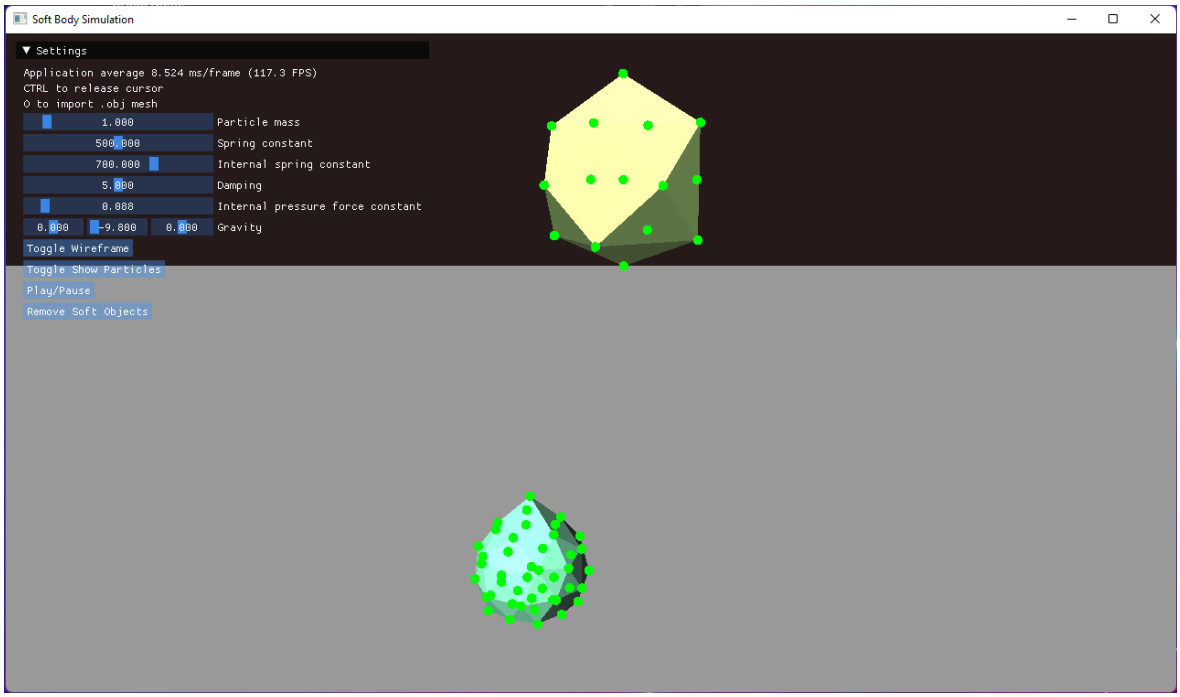
Slika 4.7 Simulacija s niskom vrijednošću konstante unutarnjih opruga i normalnom vrijednosti konstante unutarnje pritisne sile, ali bez ostalih opruga

Ako maknemo opruge koje povezuju susjedne čestice, tijela se spljošte na drukčiji način, kao na slici Slika 4.7. Tijela i dalje zadržavaju otprilike svoj volumen zbog unutarnje pritisne sile i unutarnjih opruga



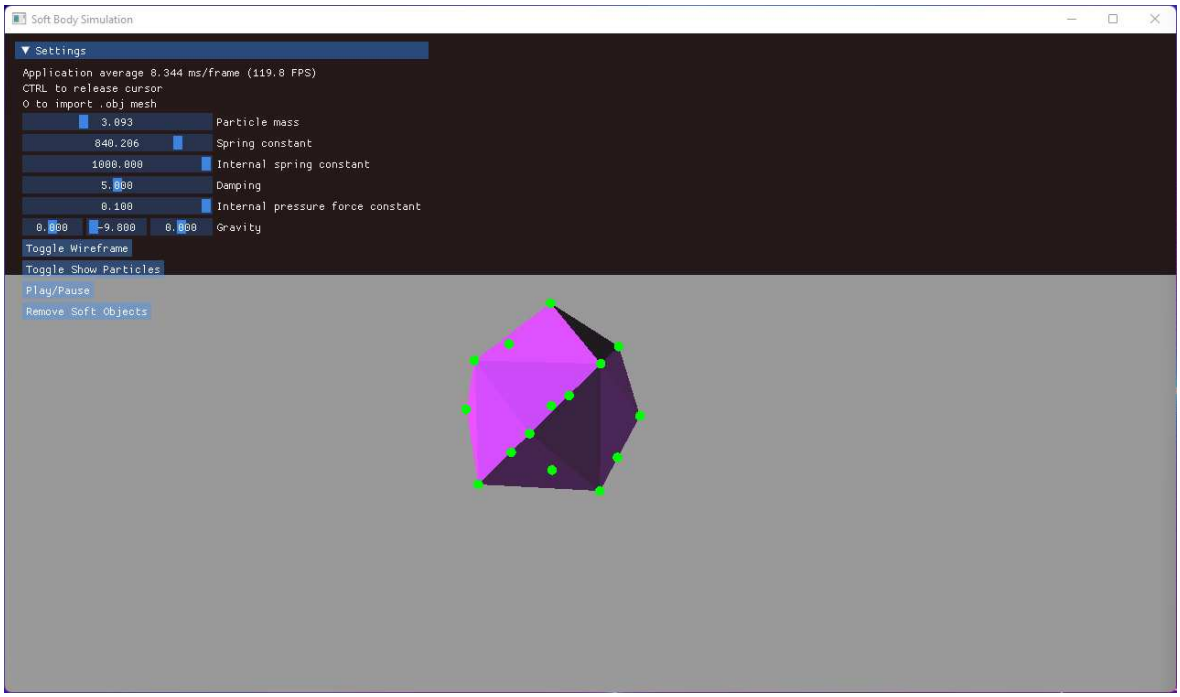
Slika 4.8 Simulacija sa samo unutarnjom pritisnom silom

Slika 4.8 prikazuje situaciju sa samo unutarnjom pritisnom silom. Dakle unutarnje opruge i unutarnja pritisna sila moraju djelovati zajedno.

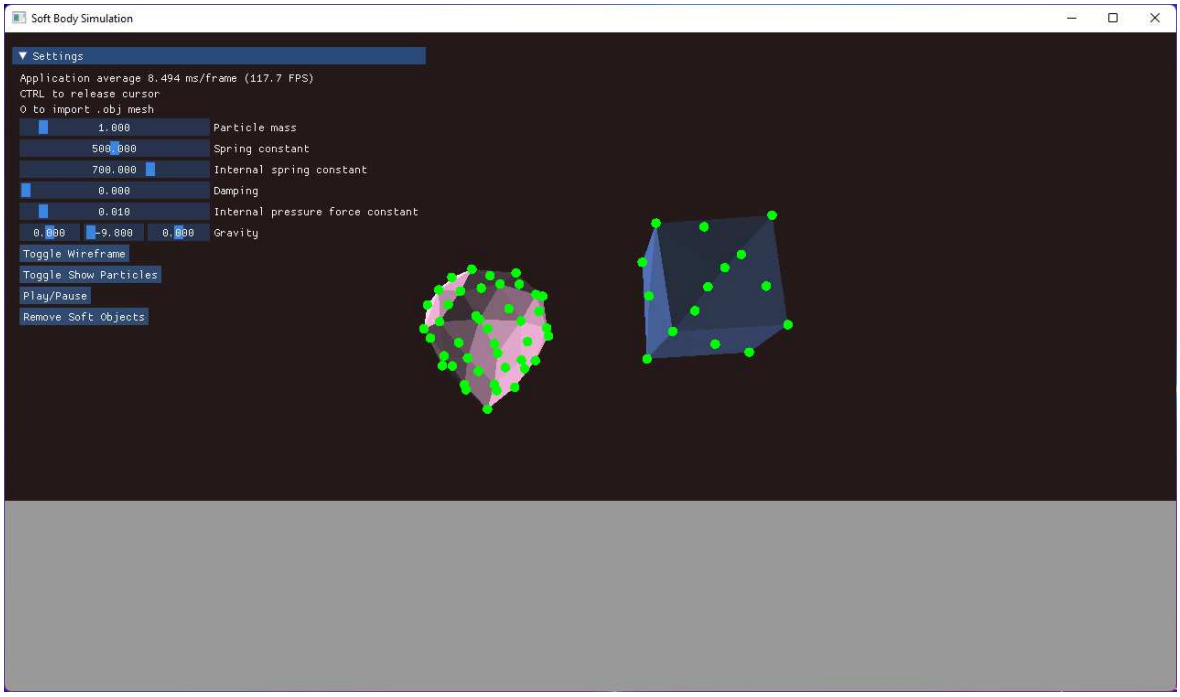


Slika 4.9 Veliki iznos unutarnje pritisne sile

Ako postavimo prevelik iznos konstante unutarnje pritisne sile (Slika 4.9), tijela su previše napuhana te se sustav ne ponaša očekivano – ne stabilizira se sam po sebi, već treba povećati ostale vrijednosti poput mase čestica da bi se stabilizirao (Slika 4.10).

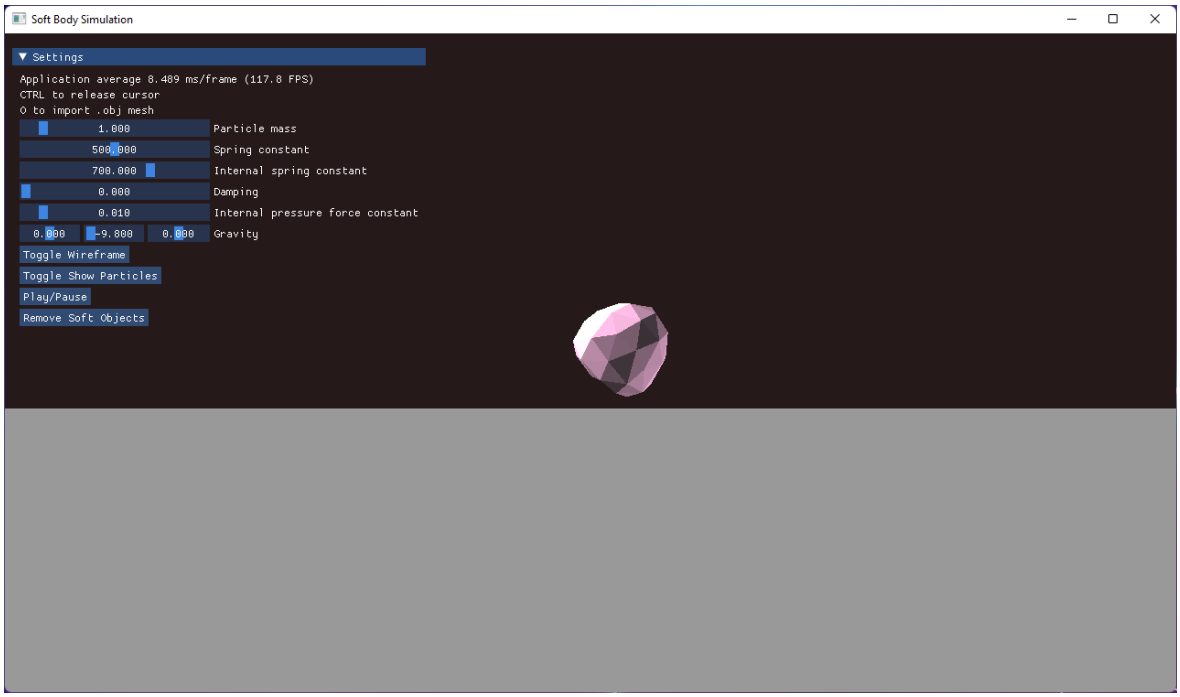


Slika 4.10 Povećane ostale vrijednosti kako bi se sustav stabilizirao



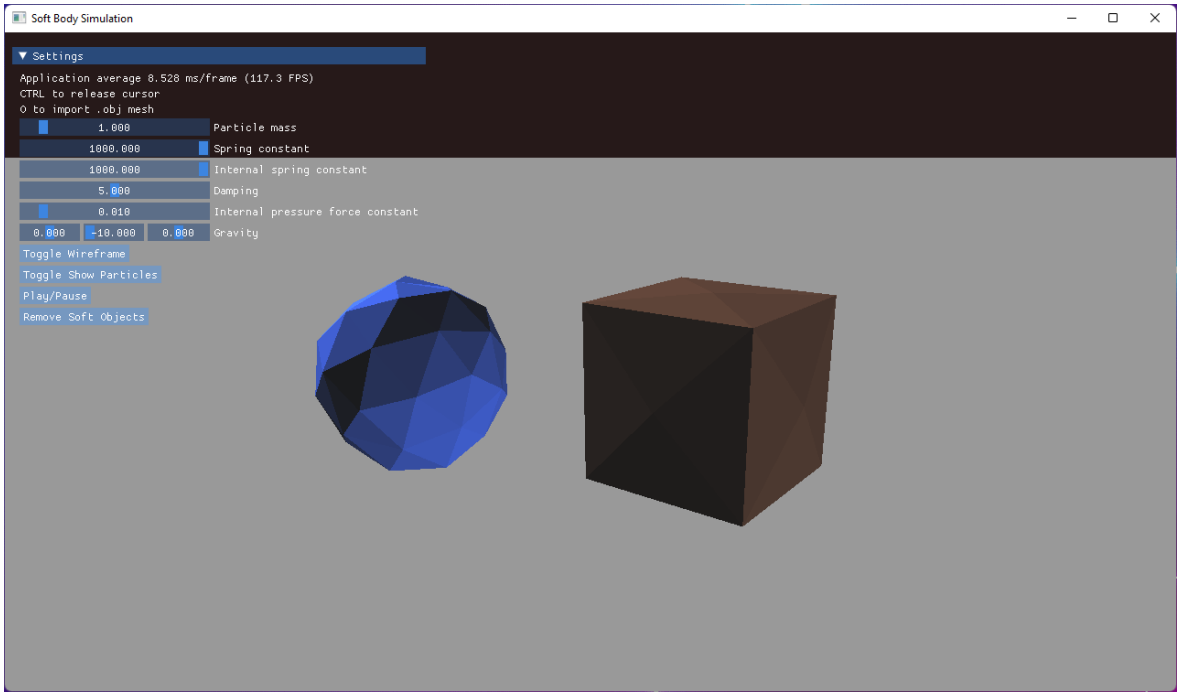
Slika 4.11 Simulacija bez prigušivanja

Slika 4.11 prikazuje situaciju bez prigušivanja netom nakon stvaranja novih mekih tijela. Nakon petnaestak sekundi, tijelo se i dalje nije stabiliziralo (Slika 4.12). Tijelo se će stabilizirati nakon nekoliko minuta, no to nije željeno ponašanje.



Slika 4.12 Simulacija nakon petnaestak sekundi bez usporavanja

Za visoke vrijednosti konstanta opruga, a normalnu vrijednost unutarnje pritisne sile, tijela se ponašaju čvršće, jače se odbijaju od tla, no ovo može uzrokovati manju stabilnost kod sudara s tlom (Slika 4.13). Sfera se ne može stabilizirati zato što prilikom svakog sudara s tlom opruge je dovoljno odbace za sljedeći sudar. Povećavanje konstante prigušenja ili mase čestica rješava ovaj problem.



Slika 4.13 Simulacija s visokim vrijednostima konstanti opruga

4.2. Problemi sustava masa i čestica

Najveći problem kod sustava masa i čestica je stabilnost. Za svaki model treba posebno nalaziti parametre za koje je sustav stabilan.

Složenost algoritma raste linearno s brojem vrhova u modelu, najskuplji izračun je računanje volumena i površine.

4.3. Ograničenja implementacije

Implementirani sustav funkcionira samo za konveksne modele. Kad bi se implementiralo računanje više konveksnih „središta“ za konkavni model te algoritam za računanje volumena koji radi i za konkavne modele, sustav bi se vjerojatno ponašao prirodno.

Aplikacija ima nekoliko manjih problema koji nisu vezani za algoritam simulacije, najveći problem je što kad se smanji *framerate*, svi objekti skoče. Ovo se događa prilikom učitavanja novog modela iz datoteke, a može se zaobići tako da se simulacija pauzira prije pokretanja postupka otvaranja.

Sudaranje s ostalim mekim tijelima je onemogućeno zbog vjerojatnih grešaka u kodu.

4.4. Moguće nadogradnje

Simulator bi se mogao nadograditi tako da podržava konkavne modele.

Višedretvenost se može implementirati tako da se prilikom simuliranja svakog okvira posao simuliranja mekih tijela podijeli na više dretvi, bilo to tako da se svakoj dretvi dodijeli simuliranje nekoliko tijela ili da se sam algoritam podijeli na više dretvi, što je također moguće zato što prilikom računanja elastičnih sila svakoj čestici mijenjamo samo njezinu silu, ne i lokaciju (koju koristimo za izračun sile) što znači da ne treba implementirati sinkronizaciju kojom bismo gubili na performansama.

Implementacija simulacije sudara s drugim mekim i krutim tijelima je također moguća i ne stvara probleme za višedretvenost iz prošlog poglavlja.

Implementacijom simulacije sudara također bismo mogli simulirati tkaninu.

S obzirom na to da simulacija kocke daje odlične rezultate, mogla bi se koristiti kao jednostavni sudarač za kompleksnije modele. Kocka simulira fiziku, a unutarnji model, npr.

glava robota, je samo pričvršćen za nju. Vrhovi glave robota bi se pomicali korištenjem trilinearne interpolacije između vrhova kocke.

Zaključak

Sustav masa i opruga daje vrlo dobre rezultate za jednostavne konveksne modele. Sustav svejedno ima niz problema zbog kojih npr. FastLSM daje bolje rezultate, poput problema stabilnosti te optimizacije za kompleksnije modele. Također, FastLSM sam po sebi ne ovisi o tome je li model konveksan ili konkavan, dok u slučaju sustava masa i opruga to stvara veći problem.

Potpunom implementacijom sustava masa i opruga mogao bi se dobiti uporabljiv simulator mekih tijela, no čak i tad, performanse ne bi bile dovoljno dobre za uporabu u stvarnom vremenu na kompleksnim modelima.

Cjelokupni kod aplikacije dostupan je na GitLab-u:

https://gitlab.com/davidcemeljic/springmasssimulation

Literatura

[1] https://www.cs.rpi.edu/~cutler/classes/advancedgraphics/S17/final_projects/haoxin_brandon.pdf

[2] http://www.ritsumei.ac.jp/~hirai/edu/2020/soft_robotics/Physics_Soft_Bodies_8up.pdf

[3] http://www.alecrivers.com/fastlsm/files/flsm.pdf.

[4] https://en.wikipedia.org/wiki/Mass-spring-damper_model#/media/File:Mass_spring_damper.svg.

[5] https://upload.wikimedia.org/wikipedia/commons/thumb/4/4a/FAE_visualization.jpg/375px-FAE_visualization.jpg.

Sažetak

Naslov: Simulacijski model mekih tijela

Ključne riječi: meka tijela, sustav masa i opruga, shape matching,

Ovaj rad daje pregled postojećih principa simulacija mekih tijela. Razvijeno je programsko rješenje za simulaciju ponašanja mekih konveksnih objekata. Detaljno je opisana implementacija sustava masa i opruga koji služi za simuliranje fizike mekih tijela. Opisane su prednosti i mane ovakvog principa, moguće nadogradnje te prikazani postignuti rezultati.

Summary

Title: Soft body simulation

Keywords: soft bodies, spring-mass system, shape matching

This paper gives an overview of existing soft body simulation systems. A software solution for simulation of behaviour of soft convex bodies has been developed. The implementation of a system of masses and springs that serves to simulate the physics of soft bodies is described in detail. The advantages and disadvantages of this principle are described, possible upgrades and the achieved results are presented.