SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 730

MEĐUOVISNI SUSTAVI ČESTICA

Luka Marković-Đurin

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 730

MEĐUOVISNI SUSTAVI ČESTICA

Luka Marković-Đurin

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 11. ožujka 2022.

ZAVRŠNI ZADATAK br. 730

Pristupnik:	**Luka Marković-Đurin (0036524734)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentorica:	prof. dr. sc. Željka Mihajlović

Zadatak: **Međuovisni sustavi čestica**

Opis zadatka:

Proučiti modele sustava čestica. Proučiti mogućnosti međudjelovanja više sustava čestica. Razraditi međuovisne sustave koji će utjecati jedan na drugi. Posebno obratiti pažnju na učinkovitost izračuna pri primjeni međuovisnosti. Načiniti testiranje na nizu primjera. Analizirati i ocijeniti ostvarene rezultate. Diskutirati upotrebljivost ostvarenih rezultata kao i moguća proširenja. Izraditi odgovarajući programski proizvod. Koristiti programski jezik C++ i grafičko programsko sučelje OpenGL. 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

Sustavi čestica predstavljaju kolekcije velikog broja manjih objekata koje nazivamo čestice. Jednu od prvih primjena sustava čestica možemo vidjeti u filmu iz 1982., *Star Trek II: The Wrath of Khan*, gdje su sustavi čestica iskorišteni izradu scene *Genesis Effect* [1]. Od 1980-ih sustavi čestica pronašli su svoju primjenu u mnoštvu video igara, animacija, digitalnih umjetnina te modela prirodnih fenomena poput vode, vatre, oblaka, kose, krzna…

Jedno od glavnih, ograničavajućih svojstava modeliranja korištenjem sustava čestica je velik broj objekata koje je potrebno ažurirati te prikazati u svakom koraku simulacije. Kako bi se umanjila računalna kompleksnost ovakvih modela, često se uvode određeni kompromisi. Čestice se prikazuju jednostavnim geometrijskim oblicima, sudari se provjeravaju samo s manjim brojem unaprijed određenih objekata, a interakcije s drugim sustavima zanemaruju.

Sustavi čestica koji ne zanemaruju interakcije s drugim sustavima nazivaju se međuovisni. U ovom radu prikazana je 2D implementacija međuovisnih sustava čestica, pojašnjeni su korišteni algoritmi te analizirane njihove performanse na različitim primjerima.

1. Implementacija sustava čestica

Implementacija rada ostvarena je u programskom jeziku C++, koristeći grafičku biblioteku *OpenGl* te matematičku biblioteku *OpenGlMathematics*.

1.1. Čestica

Česticu možemo opisati sljedećim osnovnim svojstvima:

- položaj
- brzina
- boja
- veličina
- rok trajanja

Osim osnovnih svojstava dodajemo i sljedeća svojstva:

- neprozirnost
- preostali rok trajanja
- aktivnost

Osnovna svojstva zadaju se pri stvaranju čestice, a dodatna svojstva se mijenjaju tijekom njene simulacije.

Čestica je implementirana kao `struct` s prethodno navedenim svojstvima (Programski kôd 1.1)

```
struct Particle
{
	glm::vec2 position;
	glm::vec2 velocity;
	glm::vec3 color;
	float opacity = 1.0f;
	float size;
	float lifetime = 1.0f;
	float lifeRemaining = 1.0f;
	bool active = false;
};
```

Programski kôd 1.1 – Implementacija objekta čestice

1.2. Sustav čestica

Sustav čestica pohranjuje sve njemu pripadne čestice te implementira metode za njihovo stvaranje, ažuriranje te prikazivanje na ekranu.

Sustav čestica implementiran je kao `class ParticleSystem` te implementira navedene funkcionalnosti. Čestice se u sustavu pohranjuju u podatkovnoj strukturi `std::vector<Particle>` čija se veličina inicijalno postavlja na broj čestica zadan pri inicijalizaciji sustava.

1.2.1. Stvaranje čestica

Zadatak ove metode je stvoriti objekt čestice s definiranim osnovnim svojstvima te je pohraniti u sustav čestica. Budući da vektor pri inicijalizaciji sustava napunjen s česticama, ovaj zadatak svodi se na određivanje indeksa čestice kojoj promijeniti svojstva na ona koja su predana ovoj metodi. Određivanje indeksa implementirano je na sljedeći način:

```
poolIndex = ++poolIndex % particlePool.size();
```

Nedostatak ovakve implementacije je to što se čestice koje su možda još aktivne mogu prebrisati s novom česticom, a da u sustavu ima ne-aktivnih čestica. Međutim ovakva implementacija poprilično je jednostavna te se pokazala dovoljno dobrom pri testiranju različitih primjera.

1.2.2. Ažuriranje čestica

Ažuriranje čestica uključuje mijenjanje svojstava svih aktivnih čestica sustava.

Kako bi se osigurala vremenska nezavisnost od brzine izvođenja programa, metodi se predaje parametar `timeStep` koji skalira promjene svojstava čestica. `timeStep` označava proteklo vrijeme u sekundama između dva uzastopna koraka simulacije.

Implementacija metode ažuriranja čestica prikazana je u programskom kôdu 1.2

```
void ParticleSystem::Update(float timeStep, const glm::vec2& acc)
{
   for (auto& particle : particlePool)
   {
      if (!particle.active)continue;
      if (particle.lifeRemaining <= 0)
      {
         particle.active = false;
         continue;
      }
      particle.lifeRemaining -= timeStep;
      particle.opacity = particle.lifeRemaining / particle.lifetime;
      particle.velocity += acceleration * timeStep;
      particle.position += particle.velocity * timeStep;
   }
}
```

Programski kôd 1.2 – Metoda ažuriranja čestica

1.2.3. Prikazivanje čestica

Zbog velikog broja čestica te njihove relativno male veličine, za prikaz jedne čestice koriste se samo dva trokuta strukturirana u kvadrat. Na kvadrat se primjenjuje tekstura koju je potrebno zadati pri inicijalizaciji sustava čestica.

Velik broj čestica rezultira velikim brojem poziva iscrtavanja grafičkog sustava te je stoga implementiran *Batch Rendering*. *Batch Rendering* je optimizacijska metoda kojom se smanjuje broj poziva iscrtavanja tako što se podaci o objektima koje je potrebno iscrtati grupiraju te iscrtavaju jednim pozivom iscrtavanja. Implementacija ove metode rezultira većom potrebom za memorijom no uvelike smanjuje broj poziva iscrtavanja te time drastično poboljšava performanse.

1.3. Razred Emitter

Razred `class Emitter` implementira metode stvaranja čestica određenog sustava temeljem potrebe i definiranih parametara. Inicijalizacija razreda vrši se predajom reference na razred sustava čestica. Neke od metoda koje implementira ovaj razred su

simulacija vatre, simulacija kiše te simulacija snijega temeljem određenih parametara. Ove metode korištene su pri implementaciji određenih primjera.

1.4. Razred ParticleHandler

Razred `class ParicleHandler` implementira metode detekcija i izvođenja interakcija čestica različitih sustava. Budući da rezultati interakcija ovise o potrebama specifičnih implementacija, prilagodba ovog razreda specifičnim potrebama ostavljena je korisniku programskog koda. Razred sadrži implementacije algoritama opisanih u poglavlju Testiranje i rezultati.

2. Međuovisni sustavi čestica

U stvarnom životu, čestice, odnosno atomi i molekule međusobnom interakcijom razmjenjuju energiju, a ta energija utječe na promjenu njihovih svojstava. Interakcija čestica dovodi do različitih kompleksnih makroskopskih pojava poput valova, temperaturnih gradijenata te kemijskih reakcija. Kako bi mogli računalno prikazati te modelirati ovakve pojave, uvodimo pojam međuovisnih sustava čestica.

Međuovisni sustavi čestica podrazumijevaju interakciju različitih sustava čestica te interakciju čestica istog sustava. Čestice se mogu mijenjati temeljem prisutnosti, broja te blizine susjednih čestica. Glavni problem međuovisnih sustava je velik broj čestica. Kako bi se odredili sudari, razmjena energije te ostali efekti interakcije bilo koje čestice potrebno je pretražiti te sumirati učinke preostalih N-1 čestica. Time je za izračun svakog koraka simulacije sustava s 1000 čestica potrebno približno 1000000 provjera. Opisani problem je kvadratne (N^2) vremenske složenosti.

2.1. Detekcija sudara

Detekcija sudara je računalni problem određivanja presjeka dvaju ili više objekata [2]. Sudari su jedan od osnovnih načina na koji čestice sustava vrše interakciju. Detekciju sudara možemo podijeliti u dvije glavne faze:

1. Određivanje svih parova objekata koji se mogu sudariti
2. Određivanje presjeka dvaju objekata

2.1.1. Široka faza (*eng. „broad phase“*)

Faza 1, poznatija po imenu „široka faza“ (*eng. „broad phase“*) predstavlja optimizaciju algoritama detekcije sudara. Temelji se na svojstvu da se sudar objekata ne može dogoditi ako se objekti ne nalaze u istom dijelu prostora. Ovo je jedan od glavnih načina na koji se pokušava utjecati na smanjenje kvadratne vremenske složenosti detekcije sudara.

2.1.2. Bliska faza (*eng. „narrow phase“*)

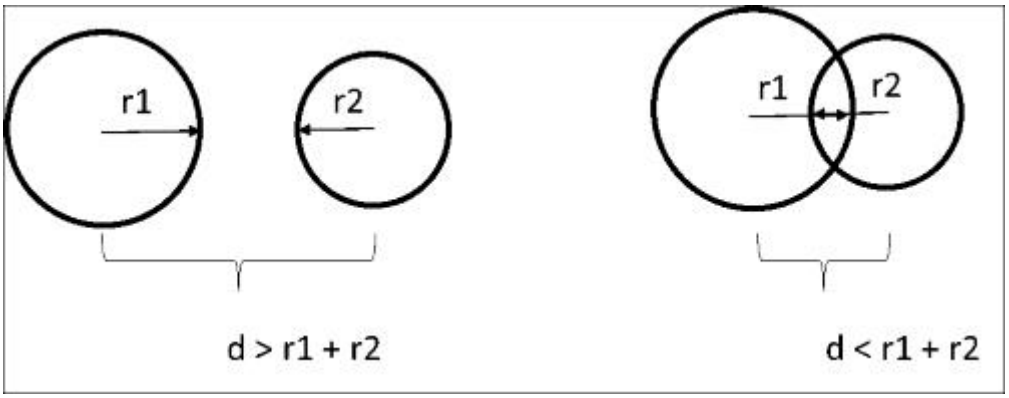
Faza 2, poznatija po imenu „bliska faza“ (*eng. „narrow phase“*) daje odgovor na pitanje: „Preklapaju li se dva objekta?“. Složenost algoritma koji daje odgovor na ovo pitanje, uvelike ovisi o obliku objekata. Za jednostavne oblike potrebno je obaviti nekoliko računskih operacija, a za složene potrebno je izdvojiti značajne računalne resurse. Zbog toga se određivanje presjeka kompleksnih objekata često pojednostavljuje aproksimacijom detekcije presjeka jednostavnih oblika koji ih omeđuju.

Zbog velikog broja čestica te njihove relativno male veličine, sve detekcije presjeka aproksimirane su detekcijom presjeka kružnica koji ih omeđuju.

Detekcija presjeka dviju kružnica svodi se na usporedbu udaljenosti središta kružnica s zbrojem njihovih polumjera. Razlikujemo sljedeće slučajeve:

1. Udaljenost je manja od zbroja polumjera → kružnice se preklapaju
2. Udaljenost je jednaka zbroju polumjera → kružnice se dodiruju u jednoj točki
3. Udaljenost je veća od zbroja polumjera → kružnice se ne dodiruju

Vizualizacija algoritma prikazana je na slici (Slika 2.1).



Slika 2.1 – Detekcija sudara dvije kružnice [3]

2.2. Naivan pristup detekciji sudara

Naivan pristup podrazumijeva zanemarivanje široke faze, odnosno implementaciju algoritma grube sile za provjeru međusobnih interakcija čestica. Prednost ovog algoritma je jednostavnost implementacije te količina korištene memorije. Za provjeru interakcije odnosno sudara čestica dva sustava, potrebno je proći kroz sve njihove čestice te provjeriti interakciju svih parova čestica sustava.

Neka je S1 = {a_1, a_2, a_3…} skup čestica sustava 1, a S2 = {b_1, b_2, b_3…} skup čestica sustava 2. Algoritam grube sile provjerava interakciju svih parova čestica (a_i, b_j).

Vremenska složenost algoritma je: |S1| * |S2|, odnosno $O(N^2)$.

Primjer implementacije naveden je u programskom kôdu 2.1.

```
NaiveCollision(ParticleSystem& ps1, ParticleSystem& ps2)
{
   for (auto& p1 : ps1.particlePool)
   {
      if (!p1.active) continue;
      for (auto& p2 : ps2.particlePool)
      {
         if (!p2.active) continue;
         glm::vec2 p1Center = p1.position +
            glm::vec2((p1.size / 2), (p1.size / 2));
         glm::vec2 p2Center = p2.position +
            glm::vec2((p2.size / 2), (p2.size / 2));
         float p1R = (p1.size) / 2;
         float p2R = (p2.size) / 2;
         if (glm::distance(p1Center, p2Center) <= p1R + p2R)
         {
            //do interaction
         }
      }
   }
}
```

Programski kôd 2.1 – Naivna implementacija metode detekcije sudara

2.3. Prostorna particija

Kako bi se smanjio broj parova čestica nad kojima se radi provjera sudara uvode se algoritmi prostorne particije. Prostorna particija podrazumijeva strukturirano dijeljenje prostora elemente te pridjeljivanje objekata pripadnom prostornom elementu. Provjeru sudara time je potrebno izvršiti samo na parovima objekata koji pripadaju istom prostornom elementu.

S obzirom na način dijeljenja prostora, korištene podatkovne strukture te način pridjeljivanja objekata određenom prostornom elementu, razlikujemo sljedeće algoritme: ujednačena mreža, četvero/oktalno stablo, k-d stabla, BSP stabla, hijerarhije volumnih obujmica.

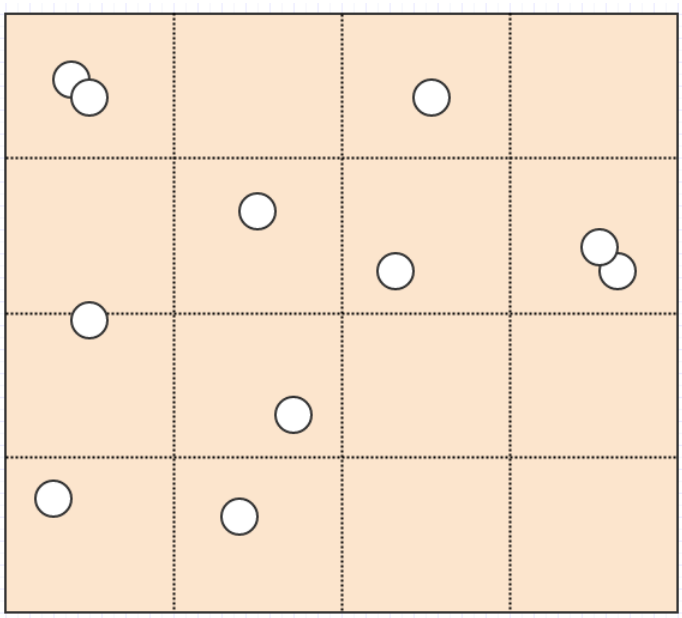
Navedeni algoritmi ne daju jednako dobre rezultate ovisno o scenariju u kojem su primijenjeni. Ujednačena mreža te četvero/oktalno stablo najprikladniji su algoritmi za primjenu optimizacije detekcije sudara mnoštva jednostavnih, dinamičkih objekata u otvorenom prostoru [4].

2.3.1. Ujednačena mreža (*eng. Uniform grid*)

Ujednačena mreža dijeli 2D prostor na kvadrate jednakih dimenzija koje nazivamo ćelije (Slika 2.2). Mreža se obično realizira kao dvodimenzionalna matrica u koju se pohranjuje lista objekata ili kao neki drugi oblik ugniježđenih struktura podataka. Umetanje objekata u mrežu temelji se na određivanju pripadnosti objekta određenoj ćeliji odnosno pripadnih indeksa retka i stupca ćelije. Posebnu pozornost potrebno je obratiti na slučajeve gdje se objekt nalazi na granici između dvaju ili više ćelija. Budući da takav objekt može biti u interakciji s bilo kojim objektom graničnih ćelija, umećemo ga u sve ćelije s kojima graniči. Određivanje ćelija kojima objekt pripada prikazano je u programskom kôdu 2.2.

```
int startRow = std::max(0, (int)(yPosUL / cellSize));
int startCol = std::max(0, (int)(xPosUL / cellSize));
int endRow = std::min((int)grid.size() - 1,
   (int)std::ceil(yPosBR / cellSize));
int endCol = std::min((int)grid[0].size() - 1,
   (int)std::ceil(xPosBR / cellSize));
// insert into the grid
for (int row = startRow; row <= endRow; row++)
{
   for (int col = startCol; col <= endCol; col++)
   {
      grid[row][col].push_back(&p);
   }
}
```

Programski kôd 2.2 – Umetanje objekta u pripadne ćelije



Slika 2.2 – Podjela prostora na ćelije [5]

Važnu ulogu u performansi algoritma ujednačene mreže nosi veličina ćelije. Za vrlo male veličine, mreža sadrži velik broj ćelija čime se značajno povećava prostorna složenost te broj koraka potreban za prolazak kroz mrežu. Za vrlo velike veličine, ćelije sadrže velik broj objekata te se algoritam približava kvadratnoj vremenskoj složenosti naivnog pristupa.

2.4. Nedeterministički pristup

Model međuovisnih sustava čestica može se dodatno pojednostaviti nedeterminističkim pristupom. Umjesto pohranjivanja liste čestica u svaki prostorni element, pohranjujemo samo broj čestica, energiju, temperaturu ili neki drugi bitan podatak. Ovime gubimo mnoge korisne informacije poput položaja, brzine, identiteta čestica…, no modeliranje interakcije je još uvijek moguće. Interakcije čestica ne temelje se više na egzaktnoj provjeri sudara već na vjerojatnosti temeljenoj na svojstvima koje pohranjujemo u prostorni element.

Budući da više ne pohranjujemo čestice u prostorne elemente, ne možemo točno odrediti interakcijske parove. Problem pretraživanja čestica i detekcije sudara postaje problem preciznosti i komunikacije sustava [6].

Algoritam interakcije dva sustava čestica možemo prikazati u nekoliko koraka:

1. Ažuriraj ujednačenu mrežu podacima sustava 1
2. Ažuriraj ujednačenu mrežu podacima sustava 2

3. Za svaku aktivnu česticu sustava 1 odredi vjerojatnost interakcije na temelju podataka pohranjenih u ćeliji (i po potrebi, okolnih ćelija)
4. Generiraj nasumičan broj između 0 i 1 te obavi interakciju nad česticom ako je dobiveni broj manji od vjerojatnosti interakcije
5. Ponovi korake 3. i 4. za čestice sustava 2
6. Resetiraj podatke u ujednačenoj mreži

Podaci koje pohranjujemo u ujednačenu mrežu te modeliranje vjerojatnosti interakcije ovisi o potrebama sustava koje nastojimo simulirati. Vjerojatnost interakcije čestice sustava 1 s česticom sustava 2 možemo primjerice odrediti omjerom broja čestica sustava 1 s ukupnim brojem čestica u ćeliji. Ili primjerice omjerom površine koju pokrivaju čestice sustava 2 te površine ćelije. Po potrebi u obzir treba uzeti i podatke susjednih ćelija, te njihove utjecaje prikladno skalirati.

Problemi ovakvog pristupa su nemogućnost određivanja točnog ishoda interakcija čestica. Budući da nam je pri izvođenju interakcije dostupna samo jedna čestica, ne možemo primjerice točno odrediti vektore brzine nakon sudara, odnosno gubimo svojstvo očuvanja količine gibanja. Zbog toga je ovakav model neprikladan za sustave kod kojih je potrebno precizno odrediti ishode interakcija. Kod određenih sustava, uz prikladno modeliranje vjerojatnosti interakcije ovaj model može davati zadovoljavajuće rezultate.

Prednosti ovakvog modela su linearna vremenska složenost (O(N)) te manja prostorna složenost u odnosu na standardne implementacije.

3. Testiranje i rezultati

U ovom dijelu opisani su primjeri na kojima su testirani implementirani algoritmi te su prikazani i analizirani rezultati.

Svi testovi izvršeni su na sljedećem sklopovlju:

- Procesor: AMD Ryzen 5 3600 6-Core Processor, 3.6 GHz
- RAM: 8GB, 2133 Mhz
- Grafička kartica: Radeon RX 580, 8GB (driver verzija: 30.0.13023.4001)

Kako bi odredili performanse algoritama, potrebno je odrediti svojstvo koje mjerimo. Program nema nikakvo ograničenje brzine izvođenja, odnosno broja koliko puta se čestice prikazuju i ažuriraju u sekundi. Time je jedini ograničavajući faktor brzine izvođenja, vrijeme potrebno za ažuriranje i prikazivanje svih čestica. To vrijeme predstavlja dobro svojstvo kojim možemo mjeriti performanse programa. Broj iscrtavanja po sekundi (*eng. frames per second - FPS*) definira se kao broj slika koje program prikazuje u sekundi. Ovo svojstvo obrnuto je proporcionalno prethodno opisanom svojstvu vremena. FPS znatno utječe na vizualnu percepciju programa te se često koristi za mjerenje njegove performanse. U nastavku su prikazane performanse algoritama temeljem ovog svojstva.

3.1. Juha čestica (*eng. particle soup*)

Juha čestica naziv je primjera u kojem se sve čestice svih sustava stvore na nasumičnim mjestima u sceni te s nasumičnim početnim brzinama. U obzir uzimamo interakcije čestica različitih sustava, a interakcije čestica istog sustava zanemarujemo. Sve sustave inicijaliziramo s istim početnim brojem čestica. Kako bi smo osigurali konstantan broj čestica i provjera, interakcije čestica samo provjeravamo te ne činimo ništa slučaju interakcije. Ovaj primjer dizajniran je kako bi se pokazale općenite performanse algoritama za određeni broj čestica.

U nastavku je prikazan prosječan izmjeren FPS za implementirane algoritme.

Tablica 3.1 prikazuje prosječan izmjeren FPS pri provjeri interakcija 2 međuovisna sustava.

Tablica 3.2 prikazuje prosječan izmjeren FPS pri provjeri interakcija 3 međuovisna sustava.

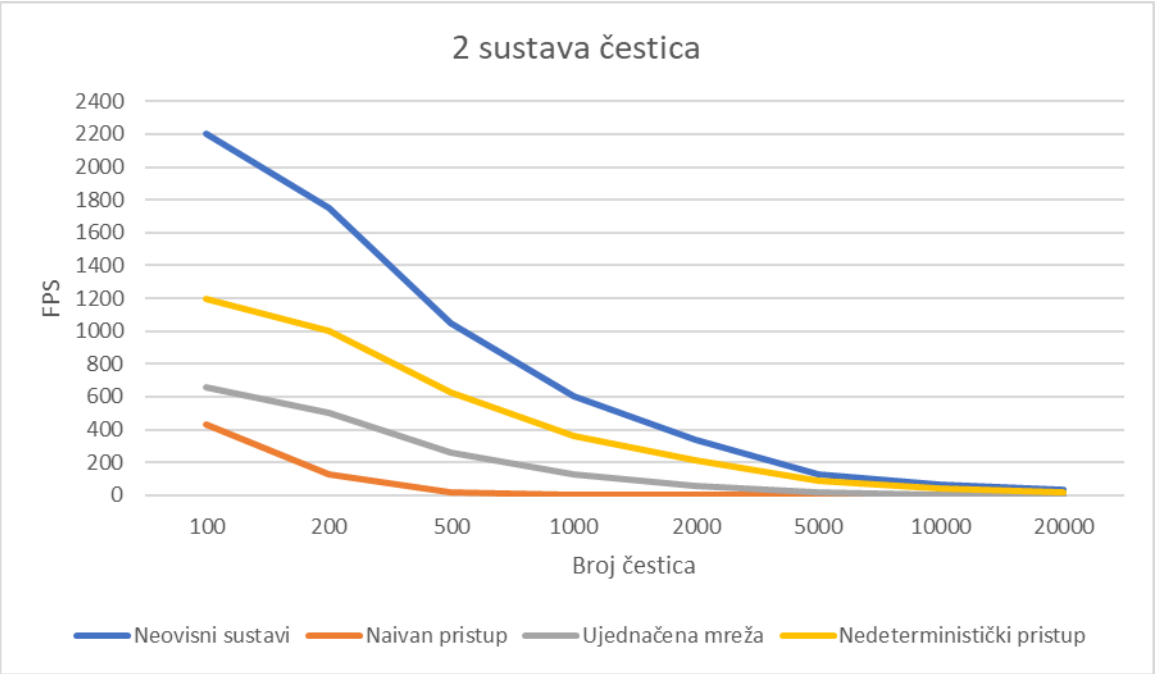
Podaci prikazani u tablicama, vizualno su predočeni na slici (Slika 3.1), odnosno na slici (Slika 3.2).

Tablica 3.1 – Prosječan FPS pri interakciji 2 sustava

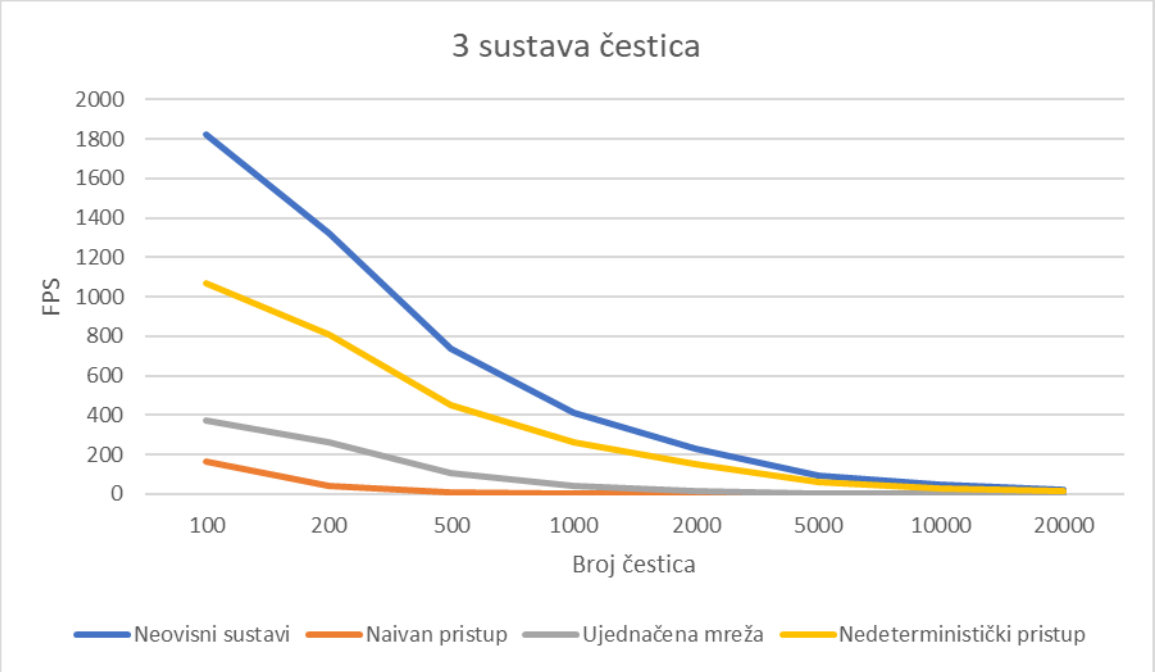
Broj čestica	Neovisni sustavi	Naivan pristup	Ujednačena mreža	Nedeterministički pristup
100	2200	430	660	1200
200	1750	125	500	1000
500	1050	22	260	630
1000	600	6	125	360
2000	340	-	55	215
5000	130	-	15	85
10000	65	-	5	45
20000	30	-	-	22

Tablica 3.2 – prosječan FPS pri interakciji 3 sustava

Broj čestica	Neovisni sustavi	Naivan pristup	Ujednačena mreža	Nedeterministički pristup
100	1820	165	375	1070
200	1320	44	260	810
500	740	8	105	450
1000	415	-	40	265
2000	230	-	12	150
5000	95	-	-	60
10000	45	-	-	30
20000	22	-	-	15



Slika 3.1 – graf prosječnog FPS-a pri interakciji 2 sustava



Slika 3.2 – graf prosječnog FPS-a pri interakciji 3 sustava

3.2. Nasumične emisije

Nasumične emisije naziv su primjera u kojem se čestice sustava kontinuirano stvaraju na nasumičnim mjestima u sceni. Brzina, te veličina čestica nasumično su određene u zadanom rasponu. Sve sustave inicijaliziramo s istim početnim brojem čestica, a stopu stvaranja čestica postavljamo kao desetina početnog broja čestica po sekundi. U obzir

uzimamo interakcije čestica različitih sustava, a interakcije čestica istog sustava zanemarujemo. Pri interakciji dviju čestica različitih sustava, uništavamo te čestice. Čestice koje izađu izvan okvira prozora uništavamo. Ovaj primjer dizajniran je kako bi se ispitala stabilnost performansi različitih algoritama.

Tablica 3.3 prikazuje varijacije raspona izmjerenog FPS-a pri provjeri interakcija 2 međuovisna sustava. Prvi stupac prikazuje broj čestica s kojima su sustavi inicijalizirani.

Tablica 3.3 – varijacija FPS-a temeljem broja čestica

Broj čestica	Naivan pristup	Ujednačena mreža	Nedeterministički pristup
100	380-550	430-455	1230-1260
500	65-75	250-260	775-805
1000	33-40	175-185	615-635
2000	16-18	135-140	465-490
5000	5-7	80-85	315-320
10000	-	53-56	225-250

Primjetno je da varijacije u izmjerenom FPS-u smanjuju proporcionalno s padom performansi. Ovo je očekivano ponašanje jer je vrijednost FPS-a teže precizno odrediti pri većim vrijednostima.

Naivan pristup ima značajne varijacije kada su sustavi inicijalizirani sa 100 čestica. Moguće objašnjenje je da su pri malom broju čestica interakcije manje vjerojatne, što dovodi do rasta broja aktivnih čestica, a samim time i potrebnih provjera interakcija. Budući da naivan pristup ima kvadratnu vremensku složenost, učinci na performanse su značajniji.

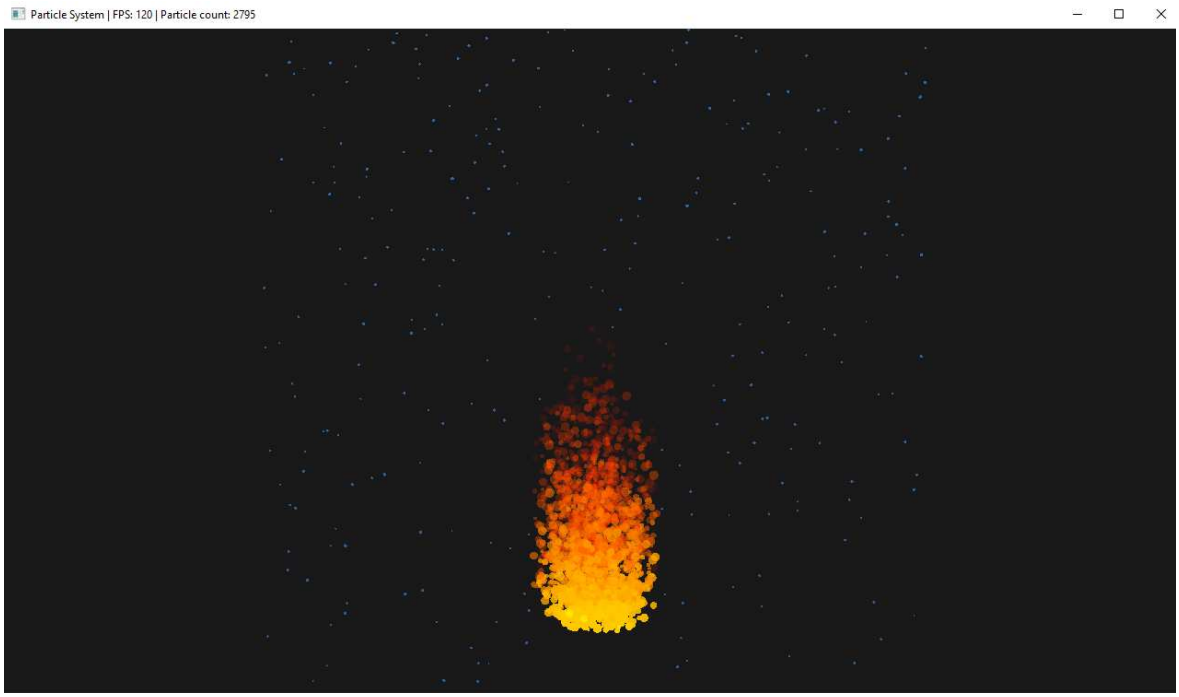
Temeljem podataka dobivenih u ovom primjeru može se zaključiti da algoritmi rade poprilično stabilno u ovakvom dinamičnom scenariju.

3.3. Vatra i kiša

Vatra i kiša naziv je primjera kojim se želi prikazati jedna od mogućih simulacija koju omogućuju međuovisni sustavi. Ovim primjerom se također želi ispitati performanse

algoritama kada je jedan od međuovisnih sustava koncentriran na malom prostoru. Primjer se sastoji od 2 sustava: vatra i kiša. Čestica vatre i četica vode pri međusobnoj interakciji nestaju. Za simulaciju vatre te kiše korištene su metode implementirane u razredu Emitter.

Sustav vatre inicijaliziran je s 5000 čestica, a sustav kiše s 2000 čestica. Čestice vatre stvaraju se stopom od 400 po sekundi, a čestice kiše stopom od 200 po sekundi. Izgled ovog primjera može se vidjeti na slici (Slika 3.3). Vatra je prikazana narančastim, a kiša plavim česticama.



Slika 3.3 – Interakcija kiše i vatre

Dobiveni rezultati su:

Naivan pristup: ~8 FPS

Ujednačena mreža: ~8 FPS

Nedeterministički pristup: ~125 FPS

Prosječan broj aktivnih čestica pri primjeni svakog algoritma je oko 2800.

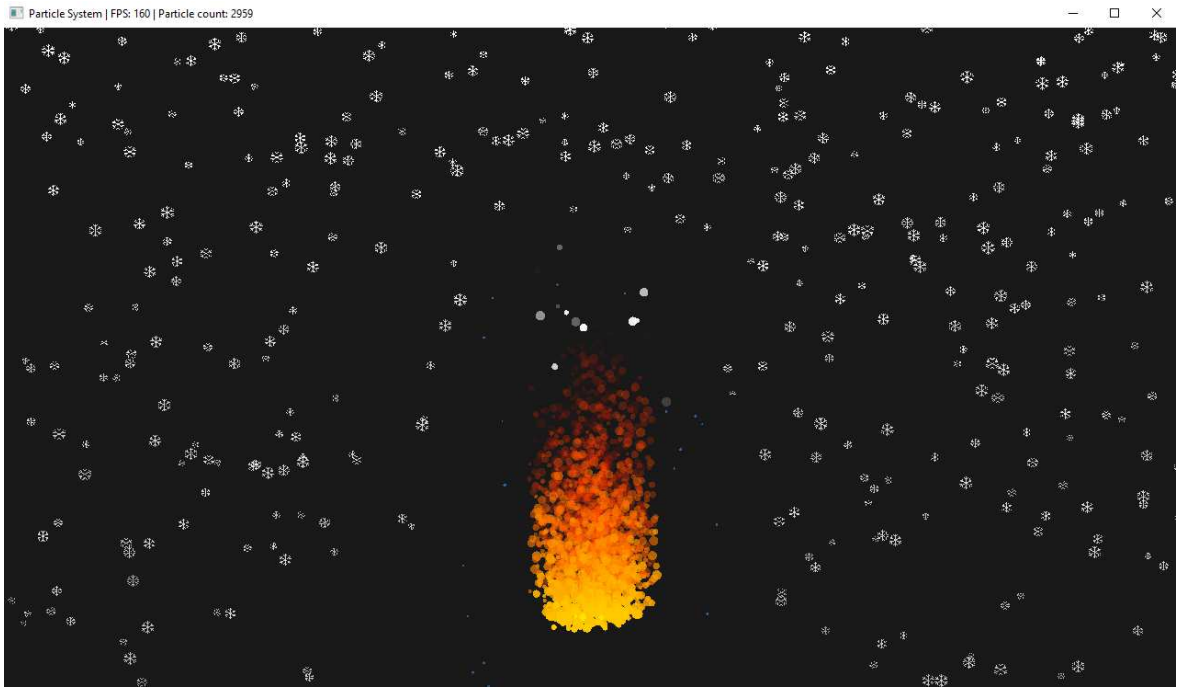
Algoritam ujednačene mreže daje značajno lošije rezultate u ovom primjeru. Razlog je velika koncentriranost sustava čestica vatre. Zbog toga se mnogo čestica nalazi u malom broju ćelija te je za interakciju potrebno provjeriti velik broj parova čestica.

3.4. Vatra i snijeg

Vatra i snijeg proširenje je primjera vatra i kiša. Glavni cilj ovog primjera je testirati rezultate nedeterminističkog pristupa, te na primjeru prikazati moguće modeliranje vjerojatnosti interakcije. Primjer se sastoji od 4 sustava: vatra, kiša, snijeg i vodena para. Svojstvo koje pohranjujemo u prostorne elemente su broj pojedinih čestica svakog sustava.

Sustav vatre inicijaliziran je s 5000 čestica, sustav kiše s 1000 čestica, sustav vodene pare s 500, a sustav snijega s 3000 čestica. Čestice vatre stvaraju se stopom od 400 po sekundi, a čestice snijega stopom od 50 po sekundi.

Izgled ovog primjera može se vidjeti na slici (Slika 3.4).



Slika 3.4 – Interakcija snijega i vatre

Čestica vatre pri interakciji s česticom vode ili snijega nestaje. Vjerojatnost ove interakcije računamo formulom:

$$p = \frac{rainCount + snowCount}{totalCount}$$

gdje su *rainCount* broj čestica kiše, *snowCount* broj čestica snijega, a *totalCount* ukupan broj svih čestica u prostornom elementu čestice vatre.

Čestici se vode pri interakciji s česticom vatre smanjuje preostali rok trajanja, a ako on dosegne 0, čestica nestaje te se na njenom mjestu stvara čestica vodene pare. Vjerojatnost ove interakcije računamo formulom:

$$p = \frac{fireCount}{totalCount}$$

gdje su *fireCount* broj čestica vatra, a *totalCount* ukupan broj svih čestica u prostornom elementu čestice kiše.

Čestica vodene pare pri interakciji s česticom snijega nestaje, a na njezinom se mjestu stvara čestica kiše. Vjerojatnost ove interakcije računamo formulom:

$$p = \frac{snowCount}{totalCount}$$

gdje su *snowCount* broj čestica snijega, a *totalCount* ukupan broj svih čestica u prostornom elementu čestice vodene pare.

Čestica snijega pri interakciji s česticom vatre ili česticom vodene pare nestaje, a na njezinom se mjestu stvara čestica kiše. Vjerojatnost ove reakcije računamo formulom:

$$p = \frac{effectiveCount}{effectiveTotal}$$

gdje je *effectiveCount* zbroj doprinosa čestica vatre i vodene pare u prostornom elementu čestice snijega te određenom broju okolnih prostornih elemenata koji se nalaze ispod. *effectiveTotal* je zbroj svih čestica svih prostornih elemenata koji su uzeti u obzir pri računanju vrijednosti *effectiveCount*. Algoritam izračuna ove vjerojatnosti prikazan je u programskom kôdu 3.1.

```
//consider particles in the vertical range of the current cell
float effectiveCount = fireCount + steamCount/2.0f;
float effectiveTotal = totalCount;
for (int r = 1; r <= 4; r++)
{
   int row = p.row + r;
   if (row > NDgrid2.size() - 1) break;
   for (int s = -4; s <= 4; s++)
   {
      int col = p.col + s;
```

```
        if (col < 0 || col > NDgrid2[0].size() - 1) continue;
        effectiveCount += NDgrid2[row][col].fireCount /
           (r * r); //further cells contribute less
        effectiveCount += NDgrid2[row][col].steamCount /
           (2.0f * (r*r));
        effectiveTotal += NDgrid2[row][col].fireCount +
           NDgrid2[row][col].rainCount +
           NDgrid2[row][col].steamCount +
           NDgrid2[row][col].snowCount;
}
```

Programski kôd 3.1 – Izračun vjerojatnosti reakcije temeljem okolnih ćelija

4. Moguća poboljšanja

4.1. Poboljšanja sustava čestica

Poboljšanja sustava čestica odnose se na mogućnost prikazivanja većeg broja čestica. Jedno od većih ograničenja prikazivanja većeg broja čestica je brzina prijenosa podataka o česticama s procesora na grafičko sklopovlje [7]. Kako bi se zaobišao ovaj problem, sustave čestica moguće je implementirati na grafičkoj kartici. Ovakve implementacije mogu efektivno koristiti svojstvo paralelizma grafičkih kartica za izračunavanje utjecaja sila na pojedinu česticu te detekcije sudara između jednostavnih geometrijskih oblika.

Algoritam slučajnih brojeva često je korišten u implementaciji ovog rada. U implementaciji je korišten algoritam `std::mt19937` koji daje dobru distribuciju nasumičnih brojeva no ne smatra se jednim od efikasnih algoritama. Implementacija efikasnijih metoda generiranja nasumičnih brojeva može imati značajan utjecaj pri simulaciji dinamičkih sustava.

4.2. Poboljšanje detekcije interakcije

Značajan dio računalnih resursa utrošen je u detekciju interakcije dvije čestice sustava. Poboljšanje detekcije interakcije odnosi se na implementaciju ostalih algoritama široke faze, te analizi njihovih performansi. Posebno zanimljiv algoritam je algoritam četvero stabla (*eng. quadtree*) koji adresira problem ujednačene mreže vidljiv u primjeru 15.

Poboljšanja je moguće napraviti i u modeliranju vjerojatnosti interakcije u nedeterminističkom pristupu.

Ostala poboljšanja odnose se na optimizaciju već implementiranih algoritama. Primjerice izbjegavanje ponovne izgradnje ujednačene mreže nakon svakog koraka simulacije.

Zaključak

Međuovisni sustavi čestica omogućuju računalno modeliranje raznih prirodnih pojava. Glavni problem međuovisnih sustava je velik broj čestica, a samim time i velik broj provjera interakcije među česticama. Pri implementaciji međuovisnih sustava posebnu pozornost treba obratiti na algoritme detekcije interakcije. U ovom radu predstavljeni su neki od algoritama detekcije interakcije te su analizirane njihove performanse na različitim primjerima. Od svih navedenih algoritama, najboljim se pokazao nedeterministički pristup. Ako nam pri modeliranju međuovisnih sustava nisu bitni egzaktni ishodi interakcija, uz pravilno modeliranje vjerojatnosti interakcije možemo dobiti zadovoljavajući model međuovisnih sustava. Nedeterministički pristup sa svojom linearnom vremenskom složenosti predstavlja vrlo efikasan algoritam detekcije interakcija. Time daleko nadmašuje algoritam grube sile, a predstavlja i značajno poboljšanje u odnosu na algoritam ujednačene mreže.

Literatura

[1] *History of computer animation (CGI)*, (2018, siječanj), Poveznica: https://computeranimationhistory-cgi.jimdofree.com/star-trek-2-the-wrath-of-khan-1982/; pristupljeno 8. lipnja 2022

[2] *Collision detection*, (25. ožujka 2022.), Poveznica: https://en.wikipedia.org/wiki/Collision_detection; pristupljeno 8.lipnja 2022.

[3] *Circular collision detection*, Poveznica: https://subscription.packtpub.com/book/game-development/9781783288199/5/ch05lvl1sec29/circular-collision-detection; pristupljeno 10. lipnja 2022.

[4] M Cihan Özer, *Uniform Grid Based*, Poveznica: http://www.mcihanozer.com/tips/computer-graphics/collision-detection-related/uniform-grid-based/; pristupljeno 8. lipnja 2022.

[5] *Efficient (and well explained) implementation of a Quadtree for 2D collision detection [closed]*, (4. studenog 2020.), Poveznica: https://stackoverflow.com/questions/41946007/efficient-and-well-explained-implementation-of-a-quadtree-for-2d-collision-det; pristupljeno: 8. lipnja 2022.

[6] Justin A. McCune, *Interdependent Particle Systems*, (1995.), Poveznica: https://people.ece.cornell.edu/land/OldStudentProjects/CS718/fall1995/mccune/final/report.html; pristupljeno 8. lipnja 2022.

[7] Lutz Latta, *Building a Million-Particle System*, (28. srpnja 2004.), Poveznica: https://www.gamedeveloper.com/programming/building-a-million-particle-system; pristupljeno 8. lipnja 2022.

Sažetak

U ovom radu objašnjen je pojam međuovisnih sustava čestica te njegove korisnosti pri modeliranju prirodnih pojava. Rad se usredotočuje na rješavanje glavnog problema međuovisnih sustava – detekcija interakcije. Predstavljeni su neki od algoritama detekcije interakcije te su analizirane njihove performanse na različitim primjerima. Također dan je primjer implementacije međuovisnih sustava čestica u programskom jeziku C++, koristeći grafičku biblioteku *OpenGl*, te su navedena moguća poboljšanja.

Ključne riječi: međuovisni sustavi čestica, sustavi čestica, čestice, detekcija sudara, detekcija interakcije, optimizacija detekcije sudara, optimizacija detekcije interakcije

Summary

This paper explains the term of interdependent particle systems and their usefulness in modelling natural phenomena. The paper focuses on solving the main issue of interdependent particle systems – interaction detection. It presents some of the algorithms of interaction detection and analyses their performance on a set of examples. It also provides an example implementation of interdependent particle systems in C++, using the graphics library OpenGl and lists possible improvements.

Keywords: interdependent particle systems, particle systems, particles, collision detection, interaction detection, collision detection optimization, interaction detection optimization

Privitak

Programski kod rada može se pronaći na poveznici: https://github.com/Vercy09/IPS