Optimizacija detekcije sudara u čestičnim sustavima

Računalna animacija - 3. laboratorijska vježba

Mario Jalšovec, 0036529377

Siječanj 25, 2023

1. Pokretanje

Nakon kloniranja repozitorija potrebno inicijalizirati potrebe *git submodule,* napraviti *build* direktorij i generirati prokejt koristeći *cmake*. To se može napraviti sljedećim isječkom koda.

```
git submodule update --init --recursive
mkdir build
cd build
cmake ..
cmake --build . --config Release
../bin/PIKProject
```

1. 1 Komande za pokretanje projekta

2. Opis rada

U radu se opisuju i isprobavaju tehnike optimizacije detekcije sudara među česticama unutar čestičnoga sustava, s ciljem poboljšanja ukupne izvedbe. Teorijsko polazište istražuje važnost detekcije sudara u simuliranju realnih interakcija čestica te raspravlja o računalnim kompleksnostima postojećih algoritama. Faza implementacije integrira istražene tehnike u jednostavni sustava čestica radi smanjenja računalnog opterećenja i poboljšanja učinkovitosti detekcije sudara. Sveobuhvatna analiza rezultata pokazuje značajna poboljšanja u računalnom vremenu, preciznosti i stabilnosti. Optimizirana detekcija sudara među česticama ne samo da povećava odziv sustava, već i poboljšava realizam interakcija čestica.

3. Korišteni načini optimizacije

3.1 Struktura podataka mreže

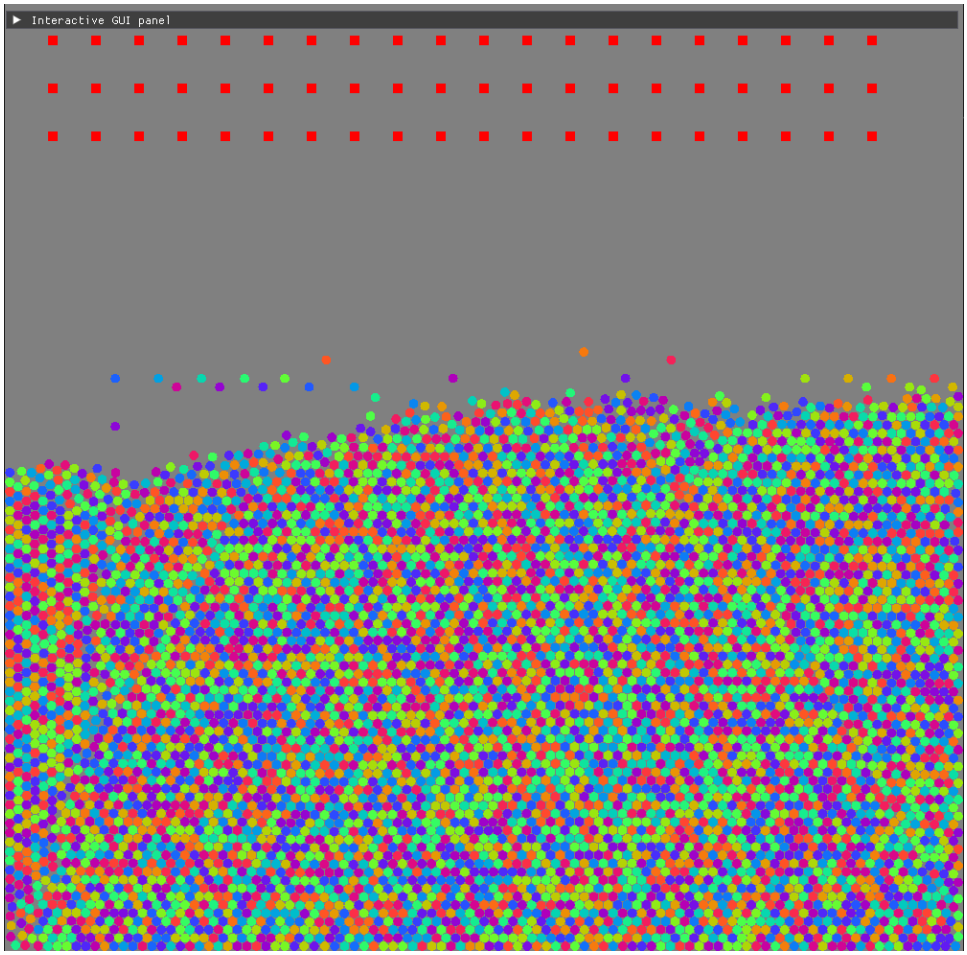
Glavna ideja korištenja strukture podataka mreže je podjela prostora na nezavisne segmente i time značajno smanjiti broj potrebnih provjera između čestica. To je moguće zato jer ako pretpostavimo da je veličina čestice manja od veličine čilije znamo da se čestice unutar jedne ćelije mogu sudarati samo s česticama unutar iste ćelije ili okolnim ćelijama. Jačina optimizacije kod strukture podataka mreže jako ovisi o veličini ćelije, broju ćelija i veličini čestica. Ako fiksiramo da je svaka čestica veličine jedne ćelije onda složenost algoritma postaje O(n). To je zato jer još uvijek trebamo iterirati kroz svaku ćeliju, ali radimo samo fiksan broj usporedbi s susjednim ćelijama koje ne ovise o globalnom broju čestica nego samo o broju unutar čestica unutar ćelije.

3.2 Paralelizacija algoritma

U području algoritama detekcije sudara, računalni zahtjevi često zahtijevaju istraživanje tehnika paralelizacije kako bi se iskoristila snaga modernih višejezgrenih i distribuiranih računalnih sustava. Ovo poglavlje istražuje teorijske osnove paralelizacije algoritama detekcije sudara temeljenih na rešetki, rasvjetljujući prirodne prednosti i strategije koje se koriste za učinkovito paralelno procesiranje.

4. Simulacija

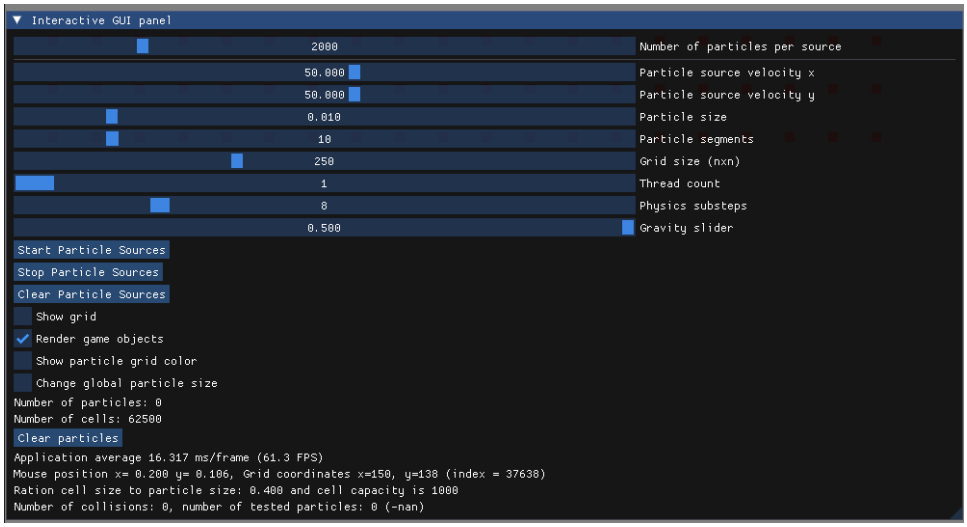
Na slici 4.1 je prikazano kako izgleda simulacija i interaktivni GUI



Slika 4.1 Prikaz simulacije

4.1. Korištenje simulacije

Glavne postavke koje je moguće mijenjati kod mjerenje performanci je veličina mreže koja specificira veličinu širine i duljine mreže koja će se koristiti u simulaciji, broj dretvi koji specifirica koliko dretava će se koristiti za simulaciju. Još je moguće mjenjati broj koraka za izračun fizike.



Slika 4.2. GUI unutar simulacije

5. Implementacija

5.1 Razvojna okolina

Ovaj projekt je izrađen u programskom jeziku C++ koristeći grafičko sučelje OpenGL, a za GUI je korištena biblioteka ImGui.

5.2 Implementacija mreže

Mreža je implementirana kao zasebna klasa koja u sebi ima vektor ćelija gdje svaka ćelija ima polje identifikacijskih brojeva ćestica.

```
class CollisionCell {
public:
    static constexpr uint32_t cell_capacity = 1000;

    uint32_t objects_count = 0;
    uint32_t objects[cell_capacity] = {};
    float maxRadiusRatio = 0.0f;

    CollisionCell() = default;

    void addObject(uint32_t objectId, float radiusRatio = 0.0f) {
        objects[objects_count] = objectId;
        objects_count += objects_count < cell_capacity - 1;
        maxRadiusRatio = radiusRatio > maxRadiusRatio ? radiusRatio : maxRadiusRatio;
    }
};

class CollisionGrid: public Grid<CollisionCell>{
public:

    CollisionGrid(int width, int height) : Grid<CollisionCell>( width_: width,  height_: height) {}


    int addObject(uint32_t x, uint32_t y, uint32_t object, float radiusRatio) {

        gridData[x * height + y].addObject( objectId: object, radiusRatio);
        return x * height + y;
    }
```

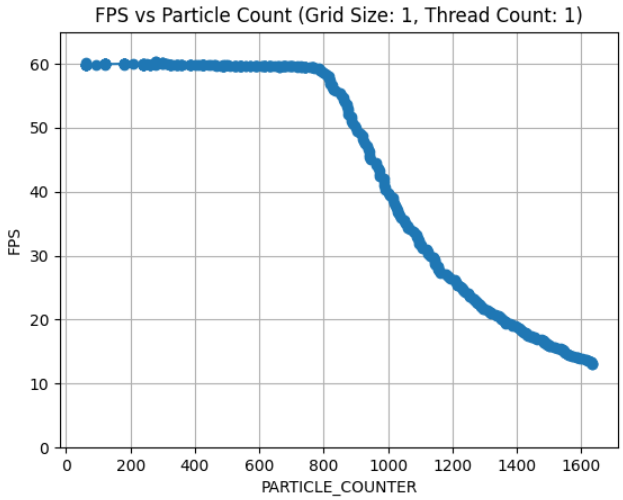
Slika 5.1 Prikaz klase CollisionCell i CollisionGrid

5.3 Implementacija paralelizacije

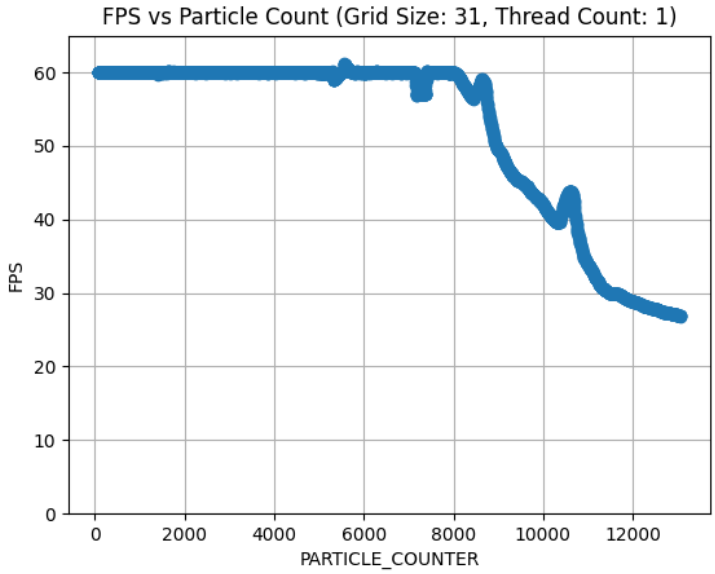
Paralelizacija je implementirana na način da se na početku mreža podijeli na jednak broj segmenata ovisno o broju dretvi i onda se svakoj dretvi samo zadaje početni i završni broj segmenta ćelija koje će obrađivati.

```
void PhysicsEngine::solveCollisions() {
    if(threadCount > 1 ) {
        const uint32_t thread_count = threadCount;

        const uint32_t thread_zone_size = ceil((grid.width * grid.height) / thread_count);

        std::vector<std::thread> mythreads;
        for (int i = 0; i < thread_count; i++) {
            uint32_t const start = i * thread_zone_size;
            uint32_t const end = start + thread_zone_size;
            mythreads.emplace_back(&PhysicsEngine::solveCollisionThreaded, this, start, end);
        }
        auto originalthread = mythreads.begin();
        //Do other stuff here.
        while (originalthread != mythreads.end()) {
            originalthread->join();
            originalthread++;
        }
        if (grid.width * grid.height % thread_count != 0) {
            solveCollisionThreaded(grid.width * grid.height - (grid.width * grid.height % thread_count),
                                   grid.width * grid.height);
        }
    }
    else {
        solveCollisionThreaded(0, grid.width * grid.height);
    }
```

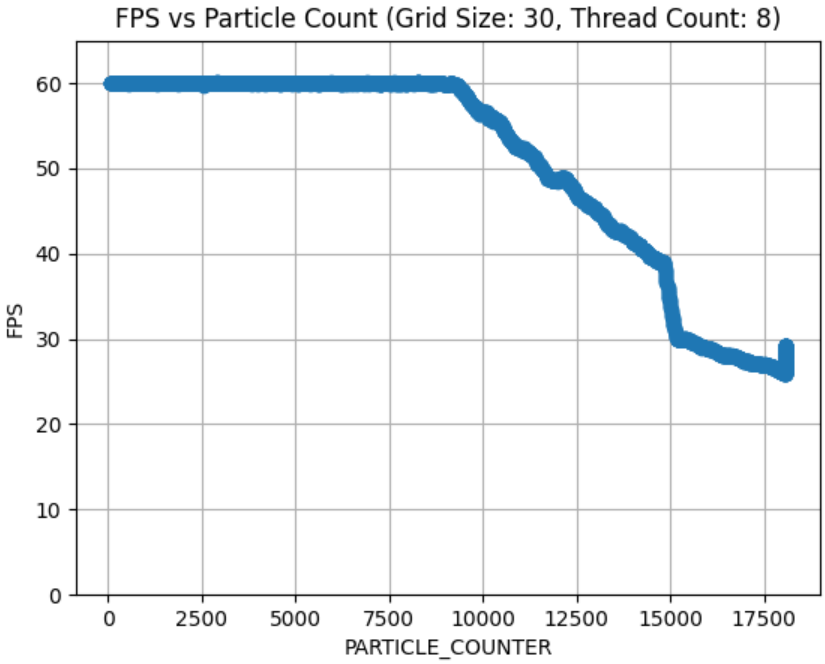
6. Rezultati



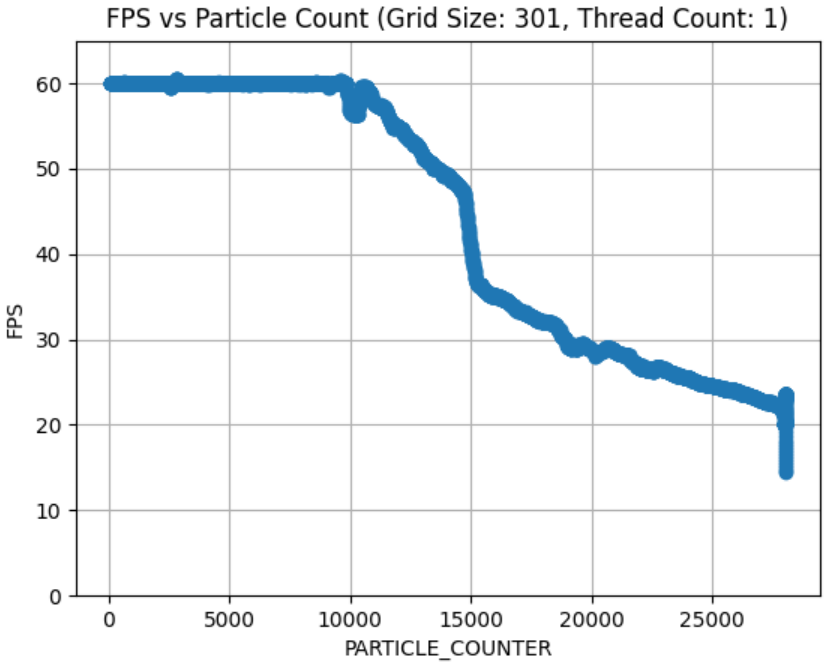
Slika 6.1. Graf promjene FPS-a pri veličini mreže 1x1 i jednom dretvom



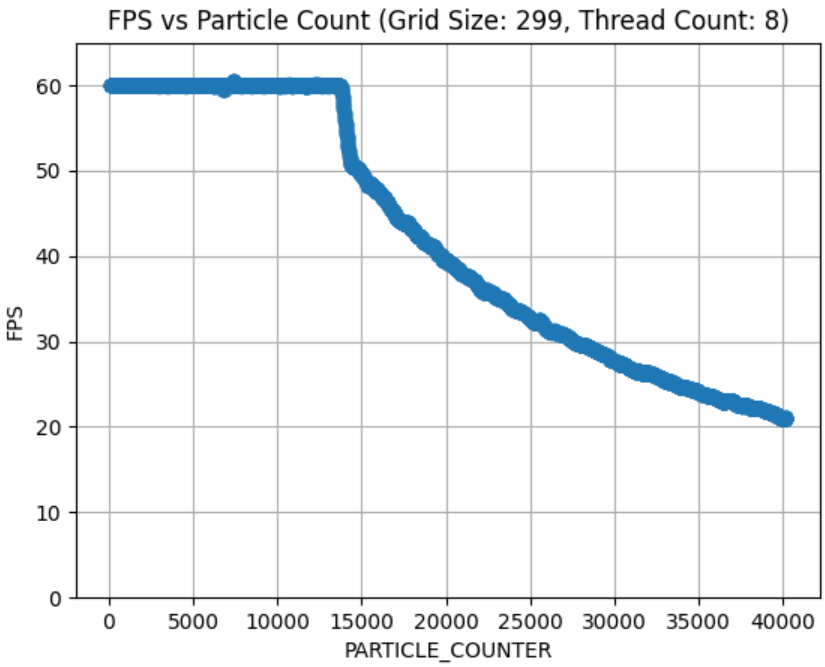
Slika 6.2. Graf promjene FPS-a pri veličini mreže 31x31 i jednom dretvom



Slika 6.3. Graf promjene FPS-a pri veličini mreže 30x30 i osam dretvi



Slika 6.4. Graf promjene FPS-a pri veličini mreže 301x301 i jednom dretvom



Slika 6.5. Graf promjene FPS-a pri veličini mreže 299x299 i osam dretvi

Referentni slučaj kada imamo samo jednu ćeliju unutar koje razrješavamo sve sudare može se vidjeti na Slika 5.1 gdje FPS pada ispod 30 za 1100 objekata

Rezultati pokazuju značajno ubrzanje performanci kada povećamo veličinu mreže s 30x30 na 300x300 to možemo vidjeti usporedbom Slika 5.1 gdje je vidljivo da FPS pada ispod 30 za 11000 objekata, dok na Slici 5.3 vidljivo da FPS pada pada ispod 30 tek za 20000 objekata, što su otprilike dvostruko bolje performace.

Kada se uspoređuju rezultati prilikom korištenja više dretvi može se vidjeti da performance rastu.

7. Zaključak

U sažetku, ovaj projekt pokazuje da integracija algoritama detekcije sudara temeljenih na rešetki i tehnika paralelizacije donosi značajna poboljšanja u obradi velikog broja čestica. I dok ove optimizacije poboljšavaju računalnu učinkovitost, prikazivanje brojnih čestica postaje značajan izazov u simulacijama u stvarnom vremenu.

Budući napredak mogao bi uključivati implementaciju optimiranih algoritama na grafičkim procesorima (GPU), iskorištavajući njihove mogućnosti paralelnog procesiranja. Ovaj prijelaz na GPU-ove ima potencijal za dodatno poboljšanje performansi, omogućavajući simulaciju još većih i složenijih sustava čestica u stvarnom vremenu.

U osnovi, seminar ističe učinkovitost algoritama temeljenih na rešetki i paralelizacije, pružajući uvid u potencijalna poboljšanja u upravljanju i prikazivanju obimnih simulacija čestica. Izgledi za implementaciju na GPU-ima obećavaju proširenje granica simulacija u stvarnom vremenu i interaktivnih okruženja.