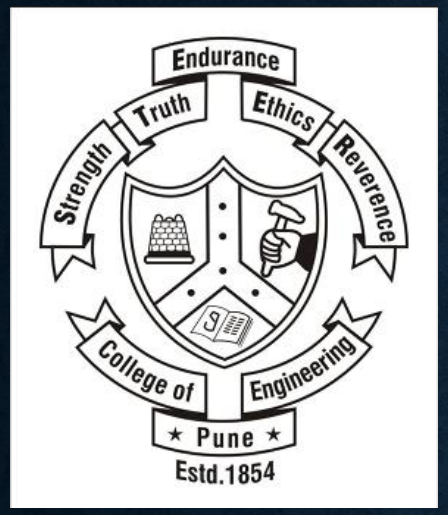


COLLEGE OF ENGINEERING PUNE

Course : Data Structures and Algorithms (DSA II)

Molecular Dynamics Simulation

Name : Vishwajit Kadam
MIS : 111903128
Div. : 2 (S3)

AY : 2020-21

Project Details

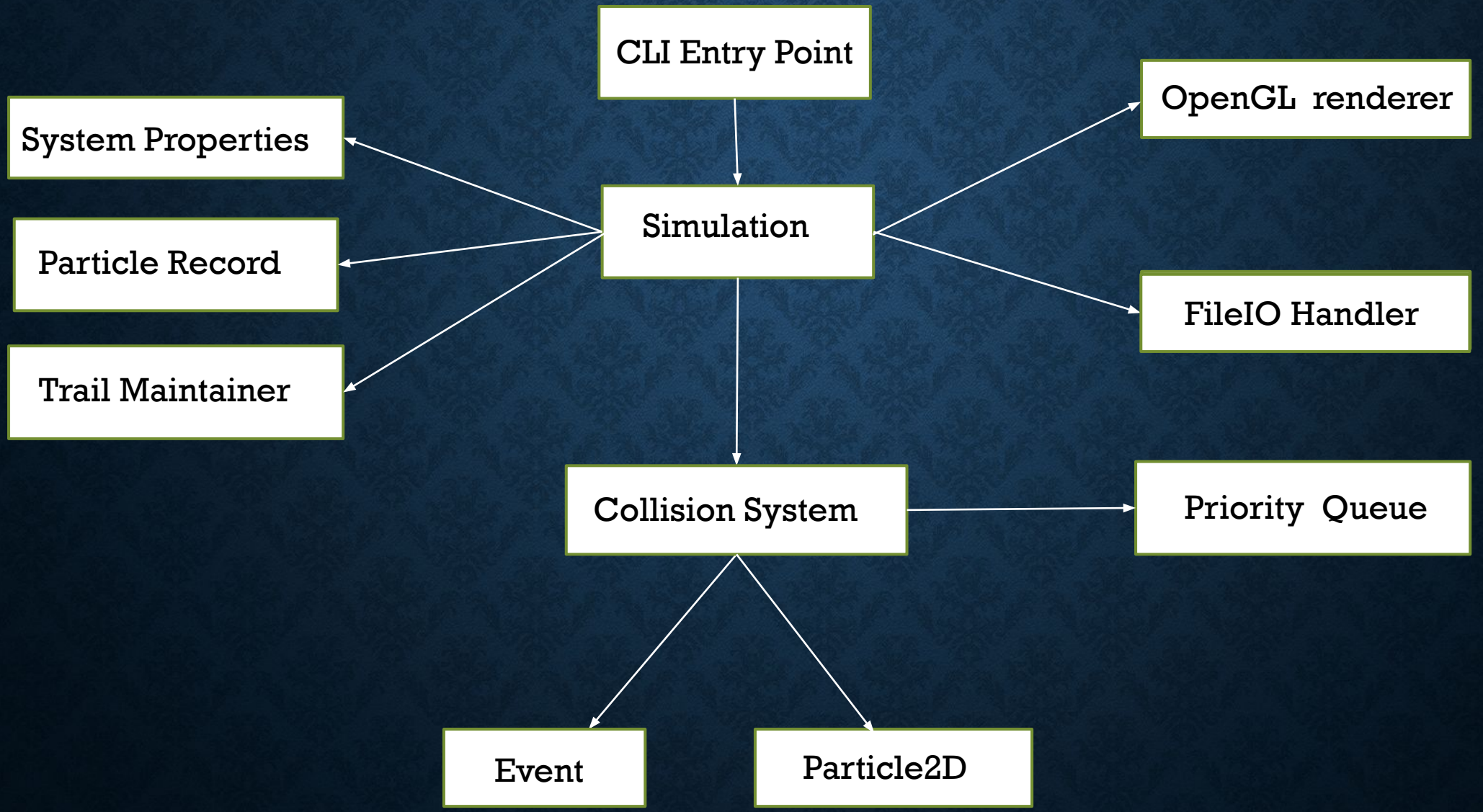
- Efficient Molecular Dynamics Simulator and Analyzer
- Uses event driven simulation technique to study behaviour of large number of particles .
- Tool to measure thermodynamic Properties statistically of system and verify physical laws.

- Data-structures used : Priority Queues, Linked-lists, arrays
- Algorithms used : Event Driven Simulation Algorithm, random System Generator

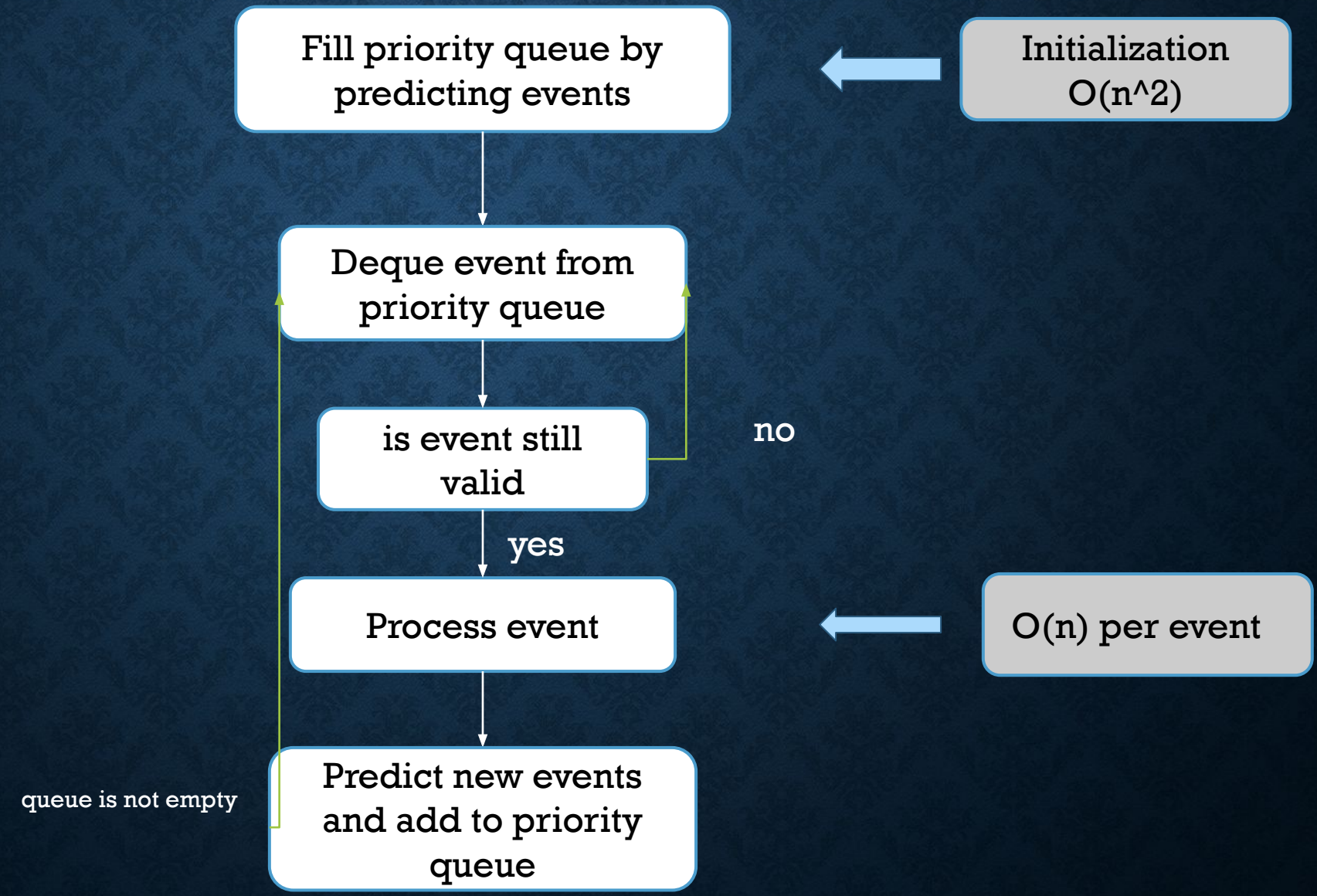
Functionalities

- simulate the collision system
- predict system state after a given a time
- calculate thermodynamic properties of system
- keep record of specific particles (useful if it's different than others, like heaviest particle in brownian motion)
- create a random collision system efficiently
- read system from and save results to file
- draw trails of specific particles as they move

Overall Structure



Event Driven Simulation Algorithm



Why Algorithm is efficient

Change state only when something interesting happens.

- Between collisions, particles move in straight-line trajectories.
- Focus only on times when collisions occur.
- Maintain PQ of collision events, prioritized by time.
- Delete min = get next collision.

Collision prediction : Given position, velocity, and radius of a particle, when will it collide next with a wall or another particle?

Collision resolution : If collision occurs, update colliding particle(s) according to laws of elastic collisions.

Random System Generation

Problem

Initialize positions of n particles (circles) each with radius r so that no two particles overlap. System must be random.

Solution

1. generate position randomly
2. check for overlaps with existing particle (O(n))

Generating position of particle randomly works well for small number of particles but as number grows, it becomes really hard to initialize positions without overlaps.

Best Case : O(n^2)
Worst case : Infinite !

So this approach is highly Inefficient

What our algorithm does

1. divide the region into 4 quadrants and assign n/4 particles to each and recursively solve problem till n > 40

2. Base Case : Divide region into a grid with distance equal to diameter and randomly choose grid intersection points to assign position to particles

Time Complexity : O(n*log(n))

Structure of the Collision System

Data structure used : array of particle pointers and priority queue of event pointers

```
typedef struct CollisionSystem{
  size_t  n;           // number of particles
  Particle **particles; // array of particle pointers
  PQueue *pq;          // priority queue of particles
  double t;            // time elapsed since start
} CollisionSystem;
```

Structure of the Particle

Particle structure stores properties of each 2D particle box of unit dimensions

```
typedef struct Particle{
  double rx, ry;    // position
  double vx, vy;    // velocity
  int count;        // number of collisions so far
  double radius;  // radius
  double mass;    // mass
  struct Color color;
} Particle;
```

Structure of the Event

Event stores collision counts of particles to check validity of events

```
typedef struct Event{
        double time;            // predicted time of event
        eventType type;         // type of event
        int particle1, particle2; // particles involved
        int countA, countB;     // collision counts at event
creation} Event;
```

Simulation ADT

This Structure holds all the properties of simulation.
Uses linked lists where dynamic data storing is required

```
typedef struct Simulation{
        CollisionSystem *cs;      // collision system to be simulated
        double limit;             // total time period of simulation
        TrailNode *traildata;     // Linked list of trails,
        bool pause;               // simulation state
        SystemProperties *sp;     // pointer to  system properties
        double       last_sampling_time;
        ParticleRecord *records;  // linked list of particle records
        int wallBalls;
} Simulation;
```

This structure maintains thermodynamic properties of system

```
typedef struct SystemProperties{
   double pressure;
    double temp;
   double collisionFreq;
    double rmsVel;
   double meanFreePath;
   int num;
   double *freePath; // array of mean free paths
double *freeTime; // array of mean time
} SystemProperties;
```

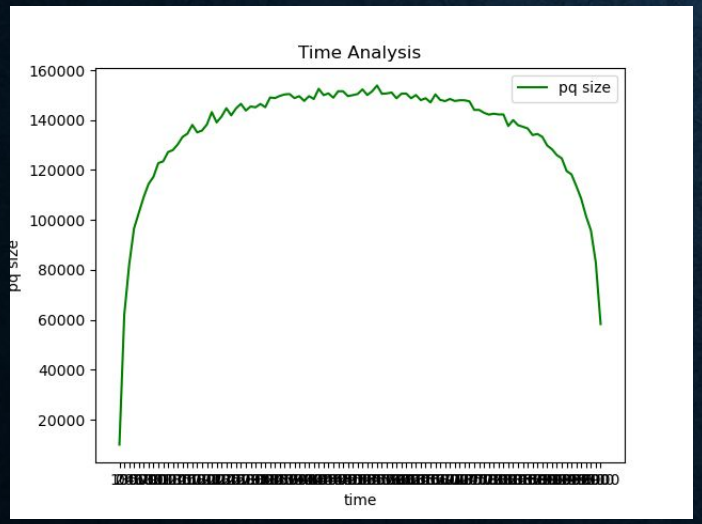
Experiments

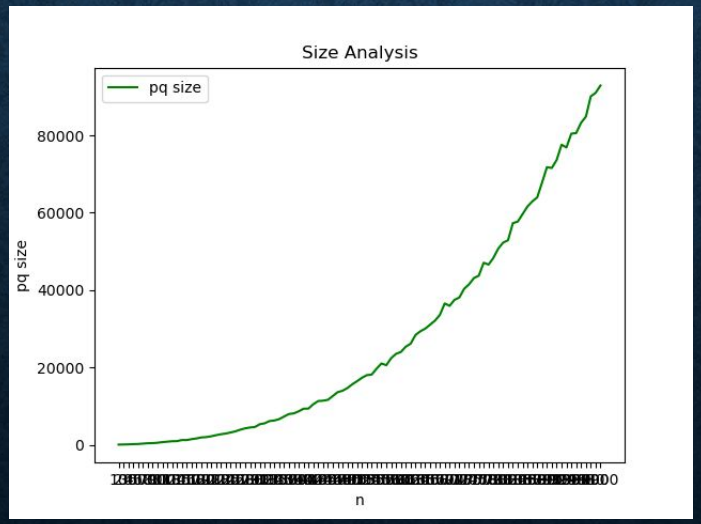
To determine optimal capacity of priority queue at initialization,

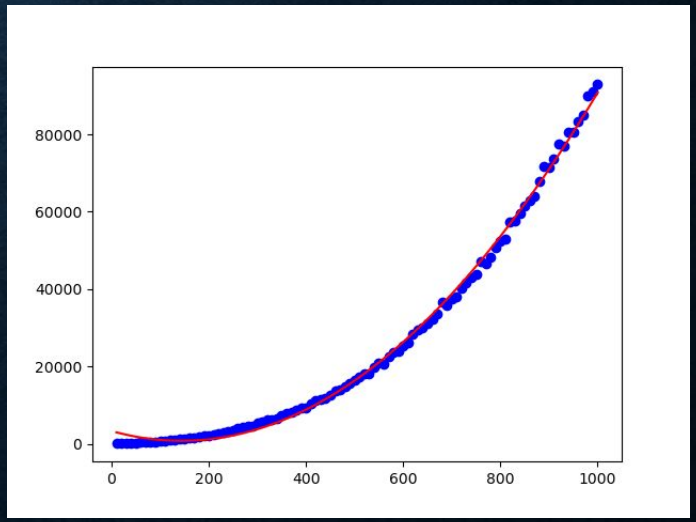
I simulated system for different values of n and recorded maximum pq capacity at any point during simulation.

using Linear Regression find best polynomial fit on data, result turned out to be

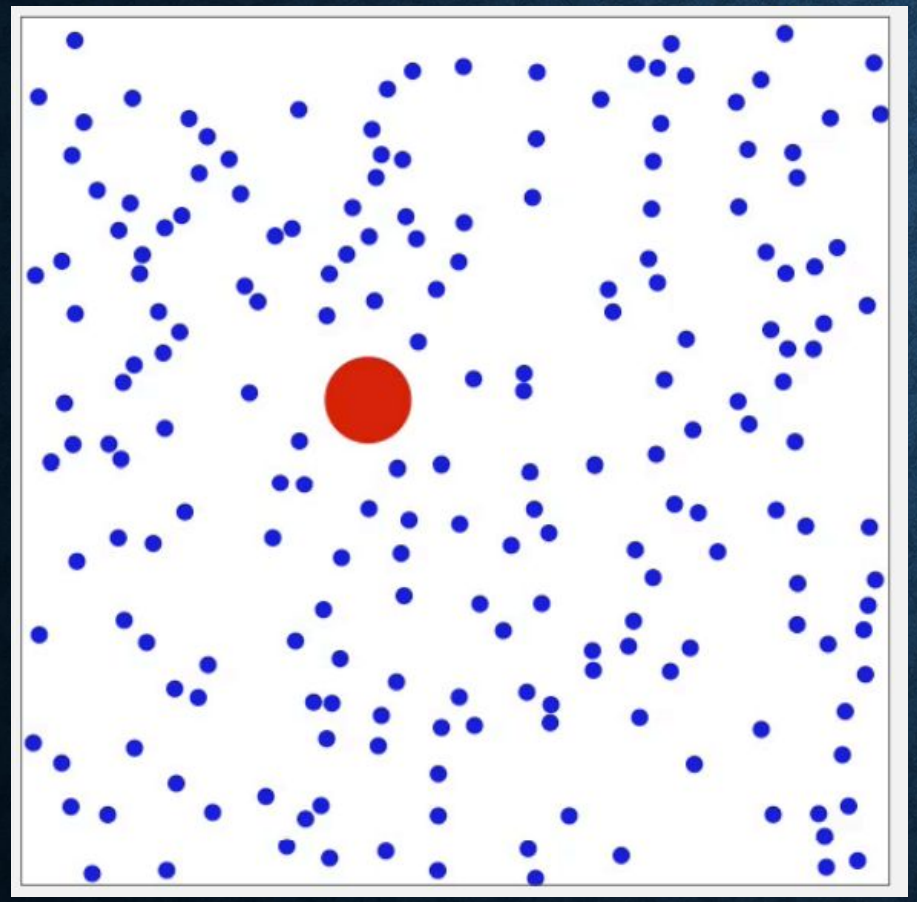
PQ size = 0.1228 * n^2 - 35.31 *n



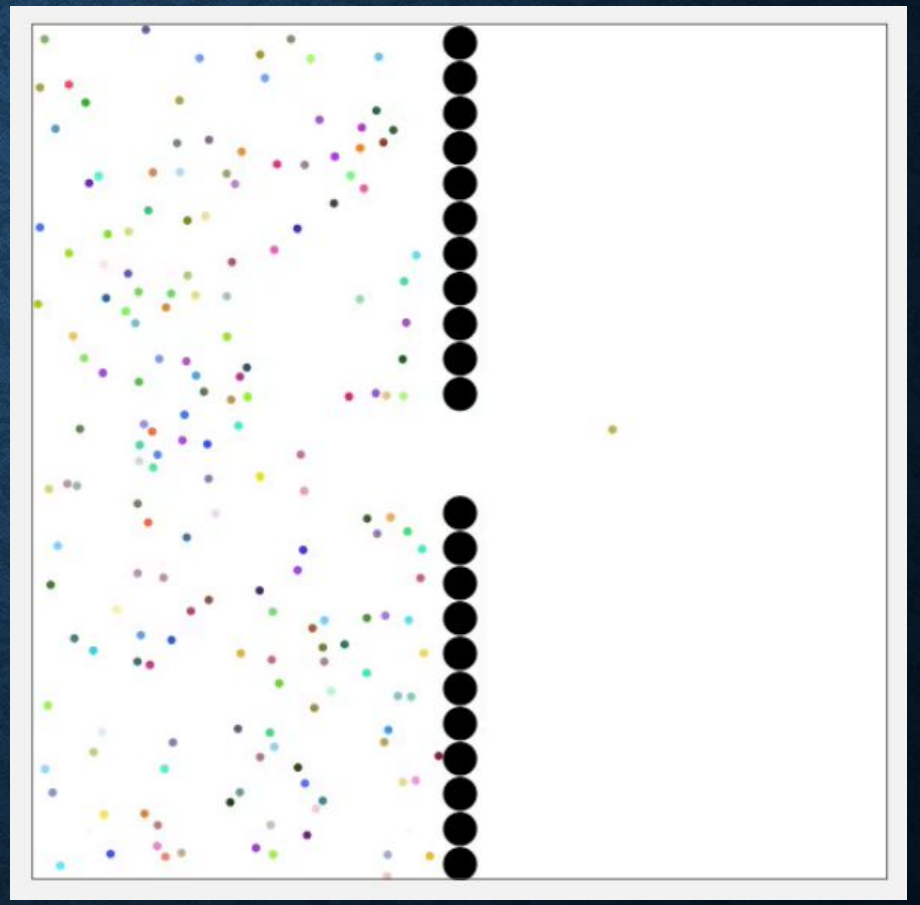




Screenshots



Brownian motion



Diffusion

Execute demo.sh file in code to see demo

Possible Improvements

- Implement cell method to reduce computation
- Parallelize processing using multiple cores using OpenCL library
- Extend this 2D engine to n- dimensions. Analyze collision of n dimensional hyperplanes

References

- 0405089.pdf (arxiv.org)
- Event-Driven Simulation (princeton.edu)
- Event-Driven Molecular Dynamics Simulation of Hard-Sphere Gas Flows in Microchannels (hindawi.com)