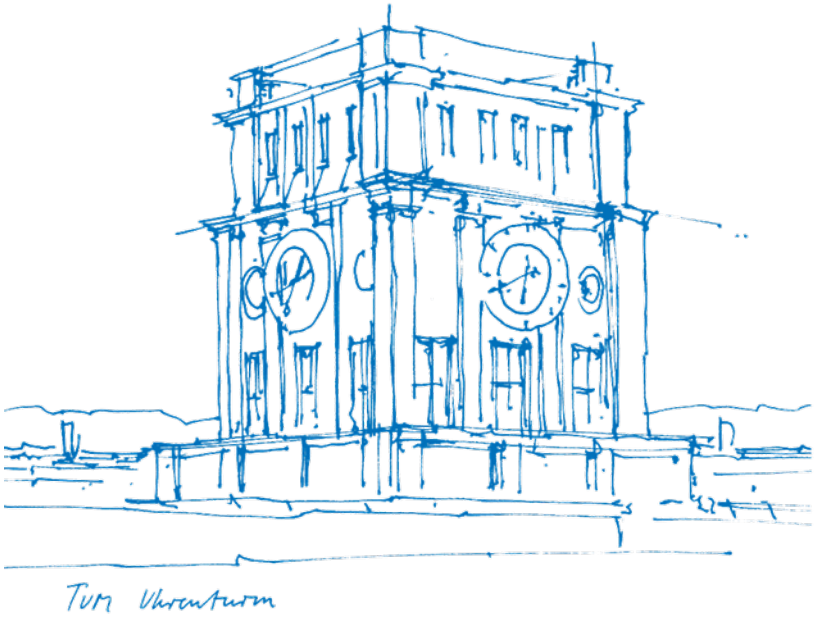
Molecular Dynamics - Assignment 3

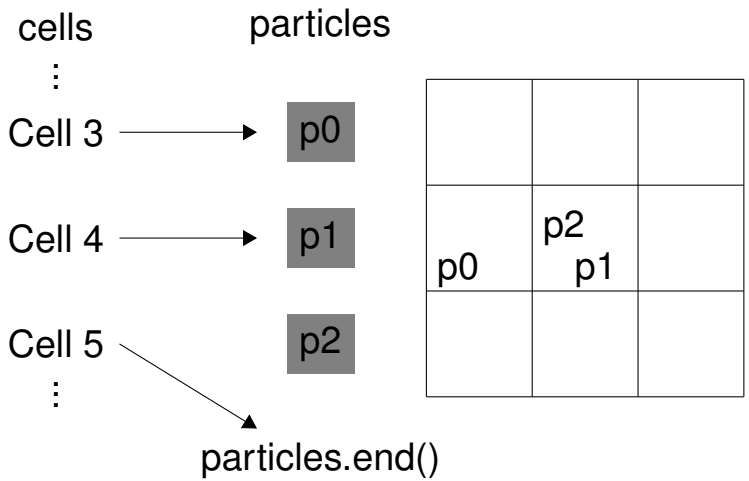
Alex Hocks Jan Hampe Johannes Riemenschneider
Technische Universität München
TUM CIT
Lehrstuhl für wissenschaftliches Rechnen
3. Dezember 2022



The Cell Data-Structure - Approach 1

Idea:

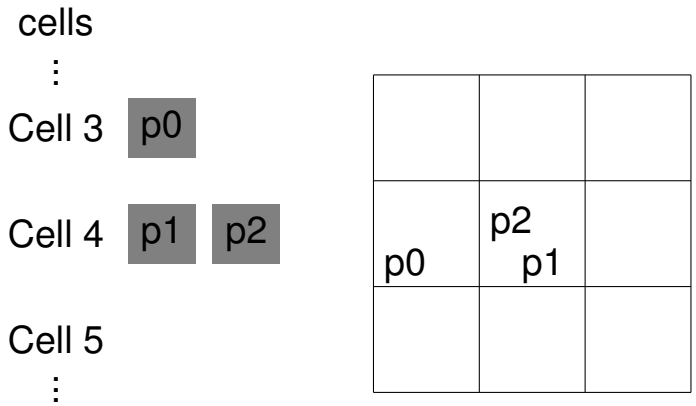
- Sort Particles in accordance to their Cell Position
- save which part of the particles-Vector corresponds to which cell



The Cell Data Structure - Approach 2

Idea:

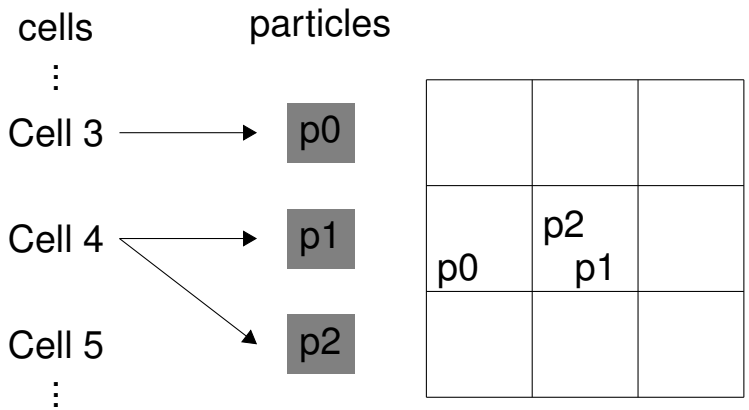
Approach 1.1 stored multiple virtual vectors in one vector $\rightarrow$ let's actually store the particles in vectors corresponding to their cell



The Cell Data Structure - Approach 3

Idea:

- Each Cell only keeps references to their members
- No sorting or copying of entire particles required



Approach Comparison

Approach 1	Approach 2	Approach 3
+ Easy to implement	+ Easy to implement	+ Easy to implement
+ Interface for old Assignments remains unchanged	+ New Implementation of some methods needed	− Interface for old Assignments remains unchanged
− Expensive struct swaps during sorting	− Expensive struct copies with potential reallocs needed	+ References are cheap
+ Direct access to particles for calculations	+ Direct access to particles for calculations	+ Dereferencing needed

Approach Comparison

Approach 1	Approach 2	Approach 3
+ Easy to implement	+ Easy to implement	+ Easy to implement
+ Interface for old Assignments remains unchanged	+ New Implementation of some methods needed	− Interface for old Assignments remains unchanged
− Expensive struct swaps during sorting	− Expensive struct copies with potential reallocs needed	+ References are cheap
+ Direct access to particles for calculations	+ Direct access to particles for calculations	+ Dereferencing needed

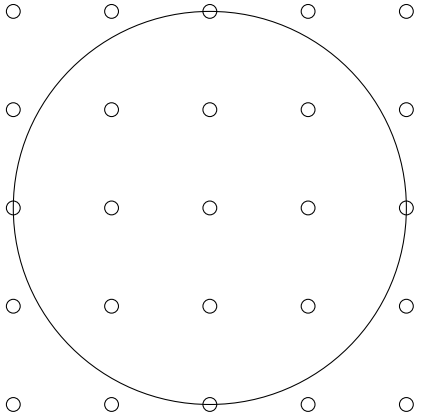
In the end we decided to implement approach 3.

Spheres

Expansion of the Body-struct utilized in Assignment 2

```
enum Shape {cuboid, sphere};
struct Body {
        Shape shape;
        Eigen::Vector3d fixpoint;
        Eigen::Vector3d dimensions;
        double distance;
        double mass;
        Eigen::Vector3d start_velocity;
};
```

$$\sqrt{x^2 + y^2 + z^2} <= r$$
$$\iff x^2 + y^2 + z^z <= r^2$$



ParticleContainer's new methods

```
class ParticleContainer{
        ...

        void forAllPairsInSameCell(void (*fun) (Particle& p1, Particle& p1));

        void forAllPairsInNeighbouringCell(void (*fun) (Particle& p1, Particle& p1));

        void forAllCells(void (*fun)(...));

        void forAllDistinctNeighbouringCells(void (*fun) (...));

        ...
}
```

- Functionality of first two methods is sufficient but hard to optimize
- Functionality of last two methods results in higher cohesion, but potential for runtime improvement

IO

input Loader gets chosen at compile time

Input parsing - Definition of Body

```
enum Shape {cuboid, sphere};

struct Body {
        Shape shape;
        Eigen::Vector3d fixpoint;
        Eigen::Vector3d dimensions;
        double distance;
        double mass;
        Eigen::Vector3d start_velocity;
};
```

CI/CD

- Protection of master branch

CI/CD

- Protection of master branch
- Deployment of CI/CD pipeline for *all* branches

CI/CD

- Protection of master branch
- Deployment of CI/CD pipeline for *all* branches

The pipeline consist of:

- library installation

CI/CD

- Protection of master branch
- Deployment of CI/CD pipeline for *all* branches

The pipeline consist of:

- library installation
- build process

CI/CD

- Protection of master branch
- Deployment of CI/CD pipeline for *all* branches

The pipeline consist of:

- library installation
- build process
- sanitizers

CI/CD

- Protection of master branch
- Deployment of CI/CD pipeline for *all* branches

The pipeline consist of:

- library installation
- build process
- sanitizers
- unit tests for every major component

CI/CD

```
name: Build and Gtest
on:
push:
branches:
- '**'        # matches every branch
pull_request:
branches: ["master"]
env:
#Customize CMake build type here from [Debug;Release;RelWithDebInfo;MinSizeRel]
BUILD_TYPE: Release
```

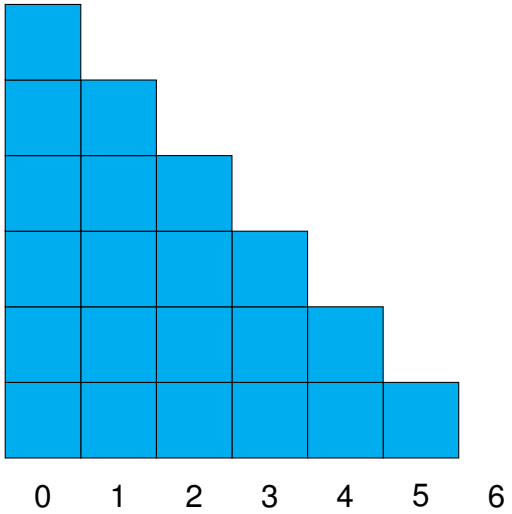
Logging

format:

```
[time][level::context] message
```

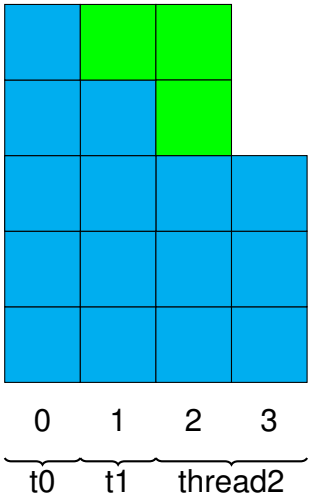
- 6 different log-levels available
- log-level can be chosen via command line input
- use of log functions since we don’t log in performance critical areas
- no need to deactivate logging at compile time

Performance optimization - "Gaussian multithreading"



- Idea: Force calculation can be multithreaded quite easily
- Evenly distribute Particle-pairs among multiple threads
- One rectangle represents one necessary force-calculation where $\min(p1, p2)$ is the number displayed below

Performance optimization - "Gaussian multithreading"



- Create "Gaussian rectangle“ as good as possible and redistribute the resulting blocks
- Threads use personal accumulators
- Accumulators get summed up in the end

In the end outsourcing the problem to OpenMP turned out to be much easier