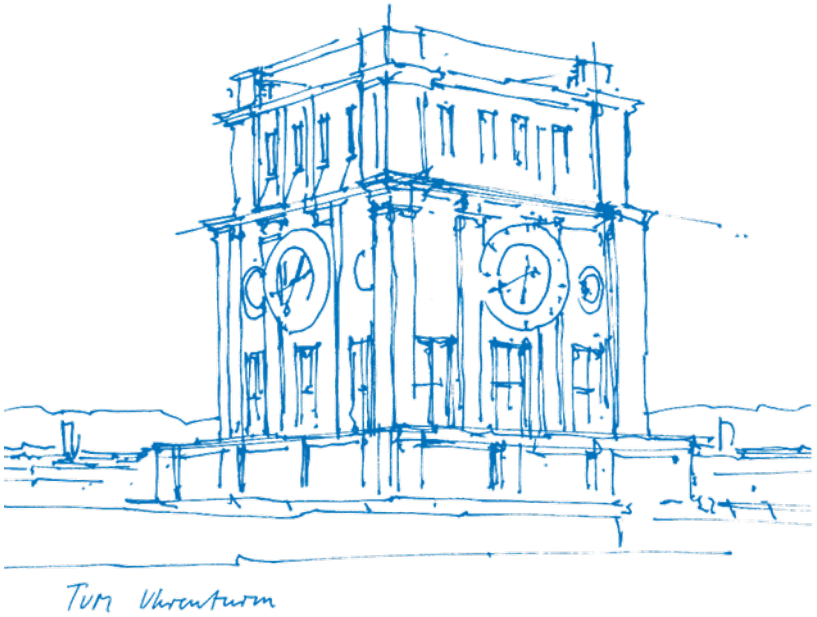
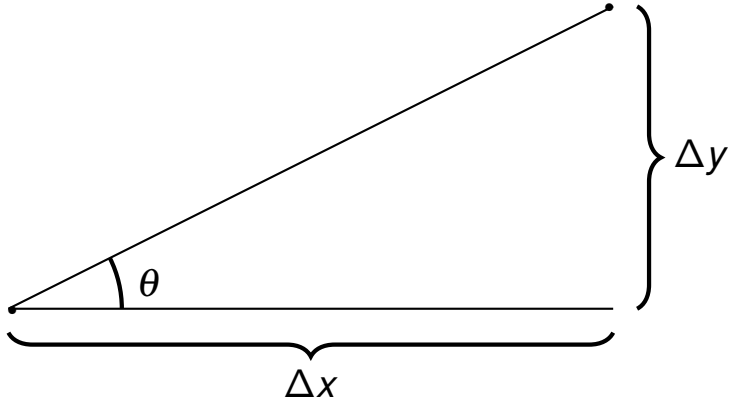
Titel der Präsentation bearbeiten

Alex Hocks Jan Hampe Johannes Riemenschneider
Technische Universität München
@Fakultät@
@Lehrstuhlname@
3. November 2022



2 Dimensional Force Calculation



$$\Delta x = x_2 - x_1 \tag{1}$$

$$\Delta y = y_2 - y_1 \tag{2}$$

$$|F| = \frac{m_1 m_2}{\Delta x^2 + \Delta y^2} \tag{3}$$

$$F_x = \cos(\theta) \cdot |F| = \Delta x \cdot \frac{m_1 m_2}{(\Delta x^2 + \Delta y^2)^{3/2}} \tag{4}$$

$$F_y = \sin(\theta) \cdot |F| = \Delta y \cdot \frac{m_1 m_2}{(\Delta x^2 + \Delta y^2)^{3/2}} \tag{5}$$

Utilizing $F_{ij} = -F_{ji}$

The naive approach ($n \cdot (n-1)$ Force calculations):

```
for all Particles p:
        for all Particles p'!=p:
                computeF(p,p')
```

Utilizing $F_{ij} = -F_{ji}$

The naive approach ($n \cdot (n-1)$ Force calculations):

```
for all Particles p:
        for all Particles p'!=p:
                computeF(p,p')
```

Utilizing $F_{ij} = -F_{ji}$

The naive approach ($n \cdot (n-1)$ Force calculations):

```
for all Particles p:
        for all Particles p'!=p:
                computeF(p,p')
```

Utilizing $F_{ij} = -F_{ji}$

The naive approach ($n \cdot (n-1)$ Force calculations):

```
for all Particles p:
        for all Particles p'!=p:
                computeF(p,p')
```

Utilizing $F_{ij} = -F_{ji}$

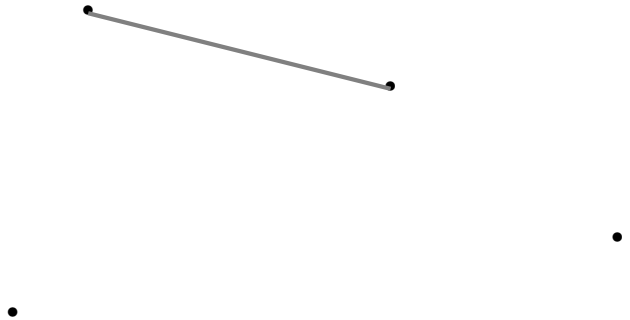
A better approach ($\frac{1}{2}n \cdot (n-1)$ Force calculations):

```
for all ParticlePairs (p,p'):
        computeF(p,p')
```

Utilizing $F_{ij} = -F_{ji}$

A better approach ($\frac{1}{2}n \cdot (n-1)$ Force calculations):

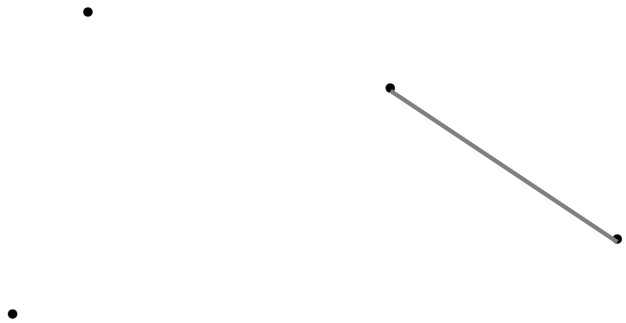
```
for all ParticlePairs (p,p'):
        computeF(p,p')
```



Utilizing $F_{ij} = -F_{ji}$

A better approach ($\frac{1}{2}n \cdot (n-1)$ Force calculations):

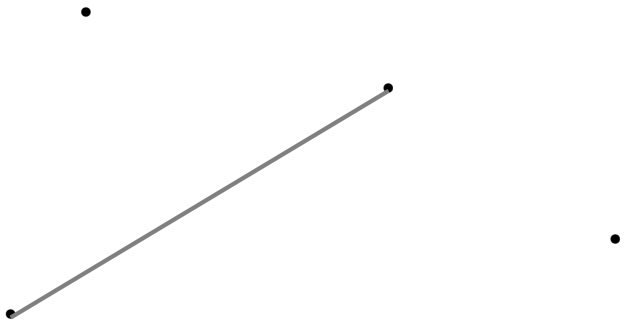
```
for all ParticlePairs (p,p'):
        computeF(p,p')
```



Utilizing $F_{ij} = -F_{ji}$

A better approach ($\frac{1}{2}n \cdot (n-1)$ Force calculations):

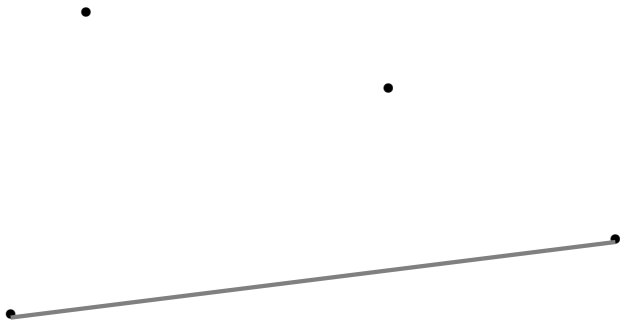
```
for all ParticlePairs (p,p'):
        computeF(p,p')
```



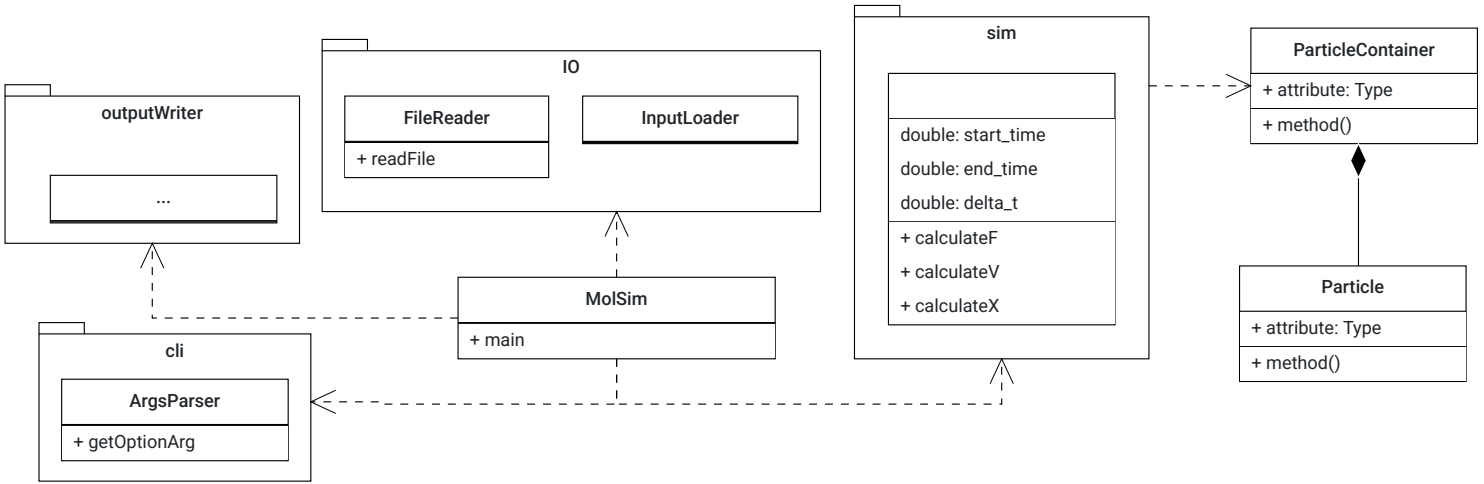
Utilizing $F_{ij} = -F_{ji}$

A better approach ($\frac{1}{2}n \cdot (n-1)$ Force calculations):

```
for all ParticlePairs (p,p'):
        computeF(p,p')
```



Refactoring



Use of the Program

```
 ./MolSim --help
Welcome to MolSim Help
Usage:
MolSim <input-files> <options>
Options:
--help, -h              Prints this screen.
-dt <value>             Sets delta time to <value>. If -dt is not specified default value is used.
-et <value>             Set end time to <value>. If -et is not specified default value is used.
-o <name>               Set base name of output files. DO NOT USE A PATH! Default is 'result'.
-of <path>              Set path to output folder. Default is ./output
```

Generic IO

```
template <typename LOCATOR, void (*LOAD)(LOCATOR, std::list<Particle>&)>
class InputLoader {
        private:
        std::list<Particle> buffer;
        LOCATOR locator;
        public:
        explicit InputLoader(LOCATOR loc) : locator(loc) {}
        InputLoader(const InputLoader& i) = delete;
        void reload() { LOAD(locator, buffer); }
        void getParticles(std::vector<Particle> &buf) {...}
};
```

Refactoring surprises

- Task: Verify the correctness of the program after major refactoring

Refactoring surprises

- Task: Verify the correctness of the program after major refactoring
- Idea: string-compare the new output with the old output. If the program works as intended it should be the same

Refactoring surprises

- Task: Verify the correctness of the program after major refactoring
- Idea: string-compare the new output with the old output. If the program works as intended it should be the same
- Observation: Very slight differences in the outputs generated (unoberservable in the videos)

Refactoring surprises

- Task: Verify the correctness of the program after major refactoring
- Idea: string-compare the new output with the old output. If the program works as intended it should be the same
- Observation: Very slight differences in the outputs generated (unoberservable in the videos)

What happened:

Refactoring surprises

- Task: Verify the correctness of the program after major refactoring
- Idea: string-compare the new output with the old output. If the program works as intended it should be the same
- Observation: Very slight differences in the outputs generated (unoberservable in the videos)

What happened:

- floating point operations are not associative $\rightarrow$ both outputs were „correct“

Refactoring surprises

- Task: Verify the correctness of the program after major refactoring
- Idea: string-compare the new output with the old output. If the program works as intended it should be the same
- Observation: Very slight differences in the outputs generated (unoberservable in the videos)

What happened:

- floating point operations are not associative $\rightarrow$ both outputs were „correct“
- probably some compiler magic

Refactoring surprises

- Task: Verify the correctness of the program after major refactoring
- Idea: string-compare the new output with the old output. If the program works as intended it should be the same
- Observation: Very slight differences in the outputs generated (unoberservable in the videos)

What happened:

- floating point operations are not associative $\rightarrow$ both outputs were „correct“
- probably some compiler magic
- Funfact: Compiling the exact same code with the exact same compiler settings will result in the same output

Refactoring surprises

- Task: Verify the correctness of the program after major refactoring
- Idea: string-compare the new output with the old output. If the program works as intended it should be the same
- Observation: Very slight differences in the outputs generated (unoberservable in the videos)

What happened:

- floating point operations are not associative $\rightarrow$ both outputs were „correct“
- probably some compiler magic
- Funfact: Compiling the exact same code with the exact same compiler settings will result in the same output
- First-hand encounter of the inaccuracies of numerical programming