c3particles
Modellierung eines Partikelsystems in C++

Praktikumsabschlusspräsentation

Rosalie Kletzander

Institut für Informatik, LMU München

6.8.2017

Zielsetzung

- Partikelsystem: Simulation (oft in der 3d Grafik) von chaotischen Systemen und natürlichen Phänomenen

- C++ als Werkzeug: mathematisch Ausdrucksstark, formale Konzepte lassen sich sauber definieren

- Ziel: Entwicklung einer Programmierabstraktion für Newton'sche Mechanik in Parntikelsystemen

Newton'sche Gesetze

"Ein Körper verharrt im Zustand der Ruhe [..] sofern er nicht durch einwirkende Kräfte zur Änderung seines Zustands gezwungen wird."

Newton'sche Gesetze

"Ein Körper verharrt im Zustand der Ruhe [..] sofern er nicht durch einwirkende Kräfte zur Änderung seines Zustands gezwungen wird."

"Das Verhältnis zwischen der Masse eines Objekts, seiner Beschleunigung und der angewandten Kraft ist F = m*a"

Newton'sche Gesetze

"Ein Körper verharrt im Zustand der Ruhe [..] sofern er nicht durch einwirkende Kräfte zur Änderung seines Zustands gezwungen wird."

"Das Verhältnis zwischen der Masse eines Objekts, seiner Beschleunigung und der angewandten Kraft ist F = m*a"

$\rightarrow$ brauchen *Kräfte*, um Beschleunigungen zu erreichen

Physikalische Grundlagen

$$a = F/m \tag{1}$$

Physikalische Grundlagen

$$a = F/m \tag{1}$$

$$\vec{v}(t) = \int (\vec{a})\mathrm{d}t = \vec{a} * t + C_v \tag{2}$$

Physikalische Grundlagen

$$a = F/m \tag{1}$$

$$\vec{v}(t) = \int (\vec{a})\mathrm{d}t = \vec{a} * t + C_v \tag{2}$$

$$\vec{s}(t) = \int (\vec{v})\mathrm{d}t = \int (\vec{a} * t + C_v)\mathrm{d}t \tag{3}$$

$$= \frac{\vec{a} * t^2}{2} + C_v + C_s \tag{4}$$

Physikalische Grundlagen

Das Superpositionsprinzip:

"Wirken auf ein Objekt mehrere Kräfte, so addieren sich diese vektoriell zu einer resultierenden Kraft auf."

$$\overrightarrow{F_{res}} = \overrightarrow{F_1} + \overrightarrow{F_2} + ... + \overrightarrow{F_n} \tag{5}$$

Modellierung

- Modellierung des Partikelsystems basiert stark auf der Physik

- besteht aus Newton'schen Objekten (Partikeln) und Kräften
$\rightarrow$ Konzepte

- die Expressions müssen die Gesetze der Physik erfüllen können

Modellierung

Expressions für "Newton'sches Objekt"

- $F = m * a \rightarrow$ "apply_force(NO)", oder "$NO << Kraft$"

- Berechnung der Position $\rightarrow$ "update(NO)"

Modellierung

Expressions für "Kraft"

- Auslöser für eine Kraft → "calc_force(NO, NO, ff)"

- Superpositionsprinzip → "accumulate(f1, f2, ..)"

Implementierung

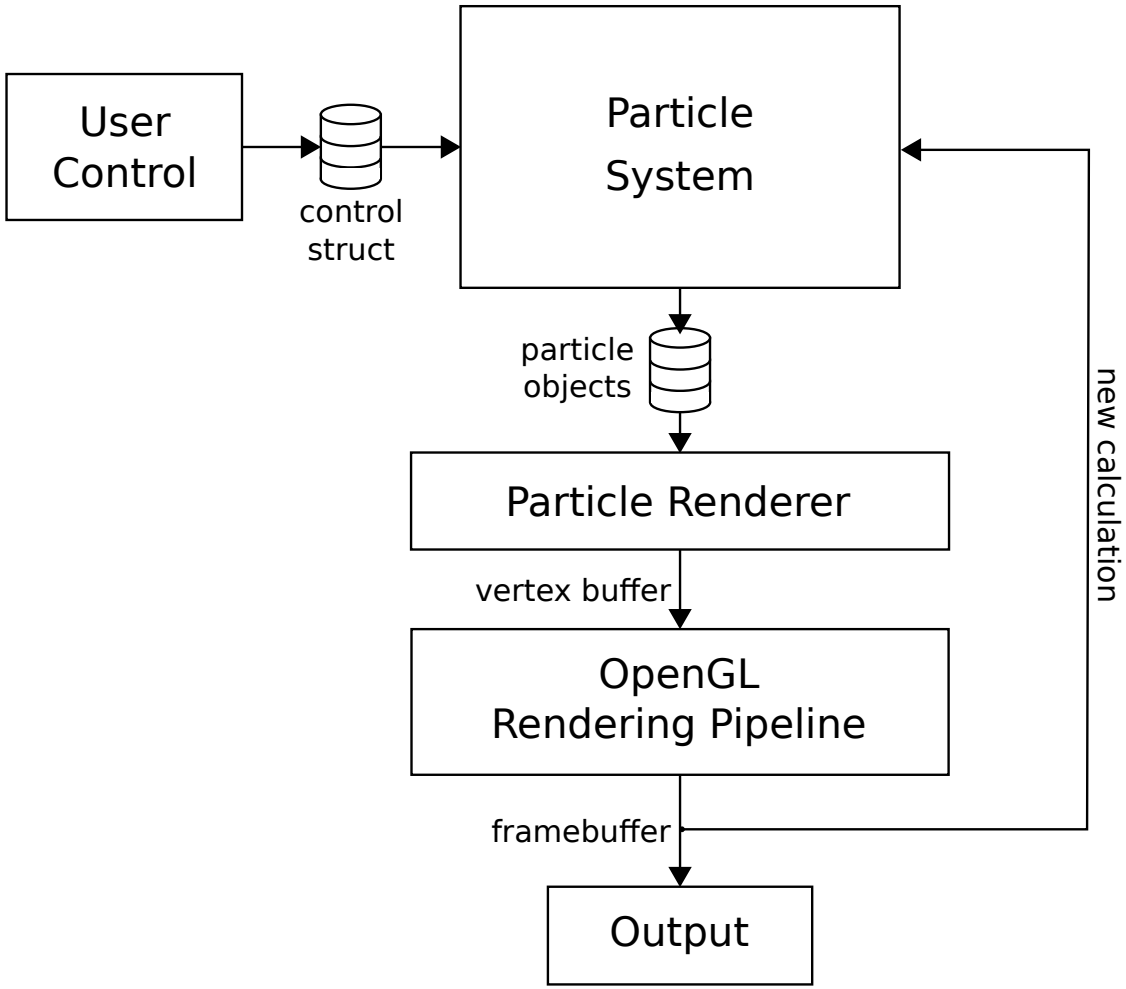

Figure: **Systemdiagramm**

Implementierung

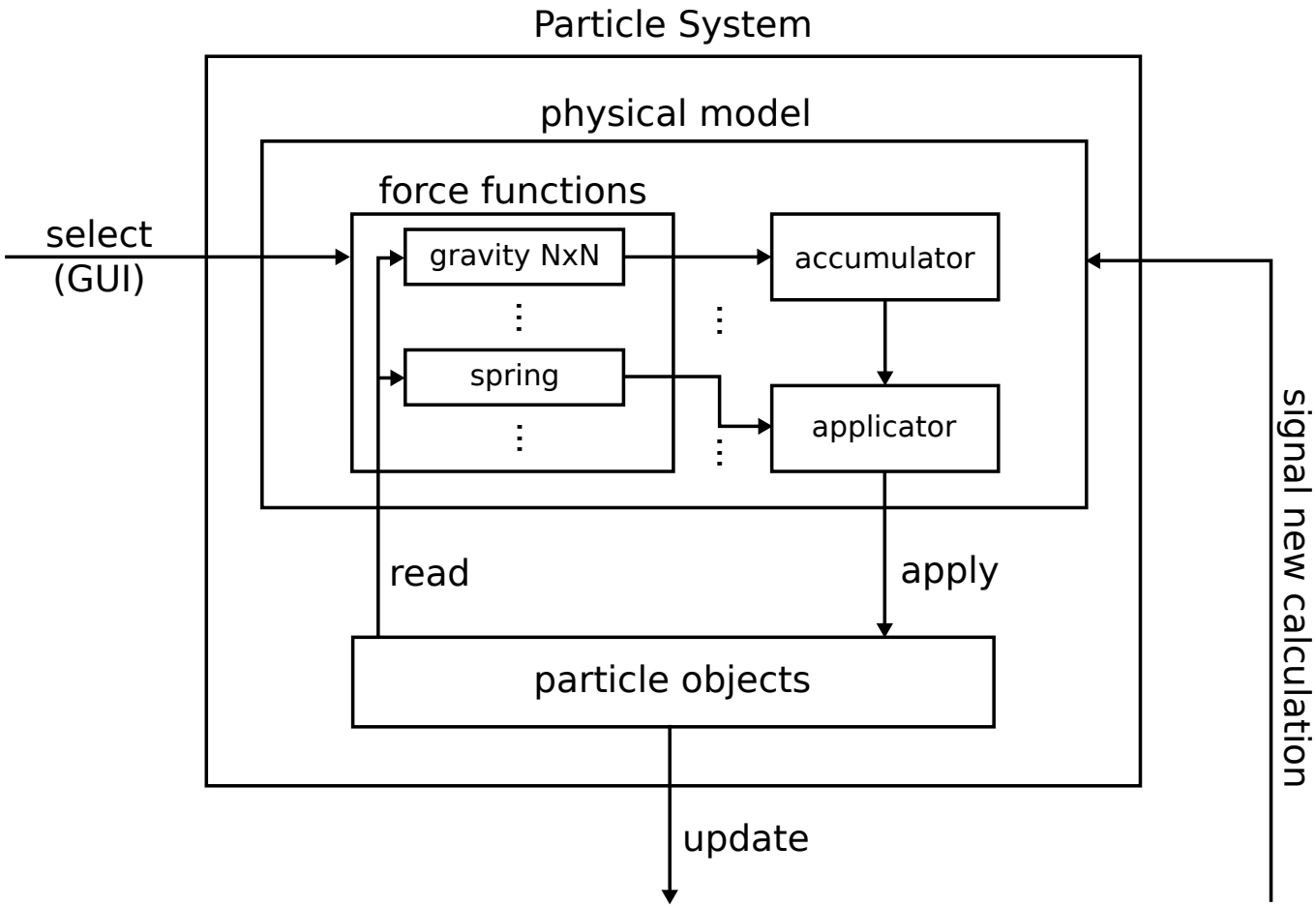

Figure: Das Partikelsystemmmodell im Detail

Implementierung

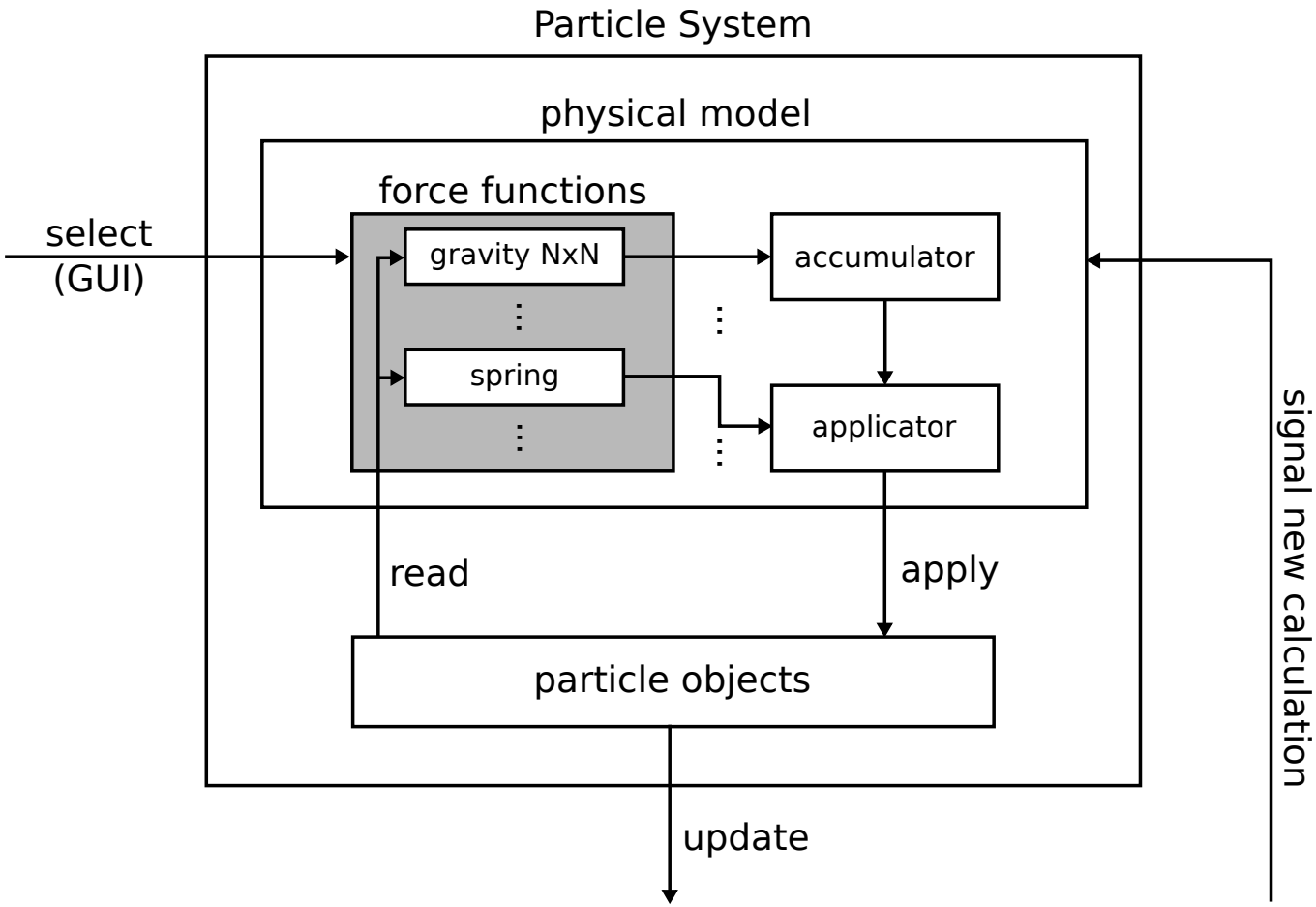

Figure: Die Kraftfunktionen (basierend auf calc_force)

Implementierung

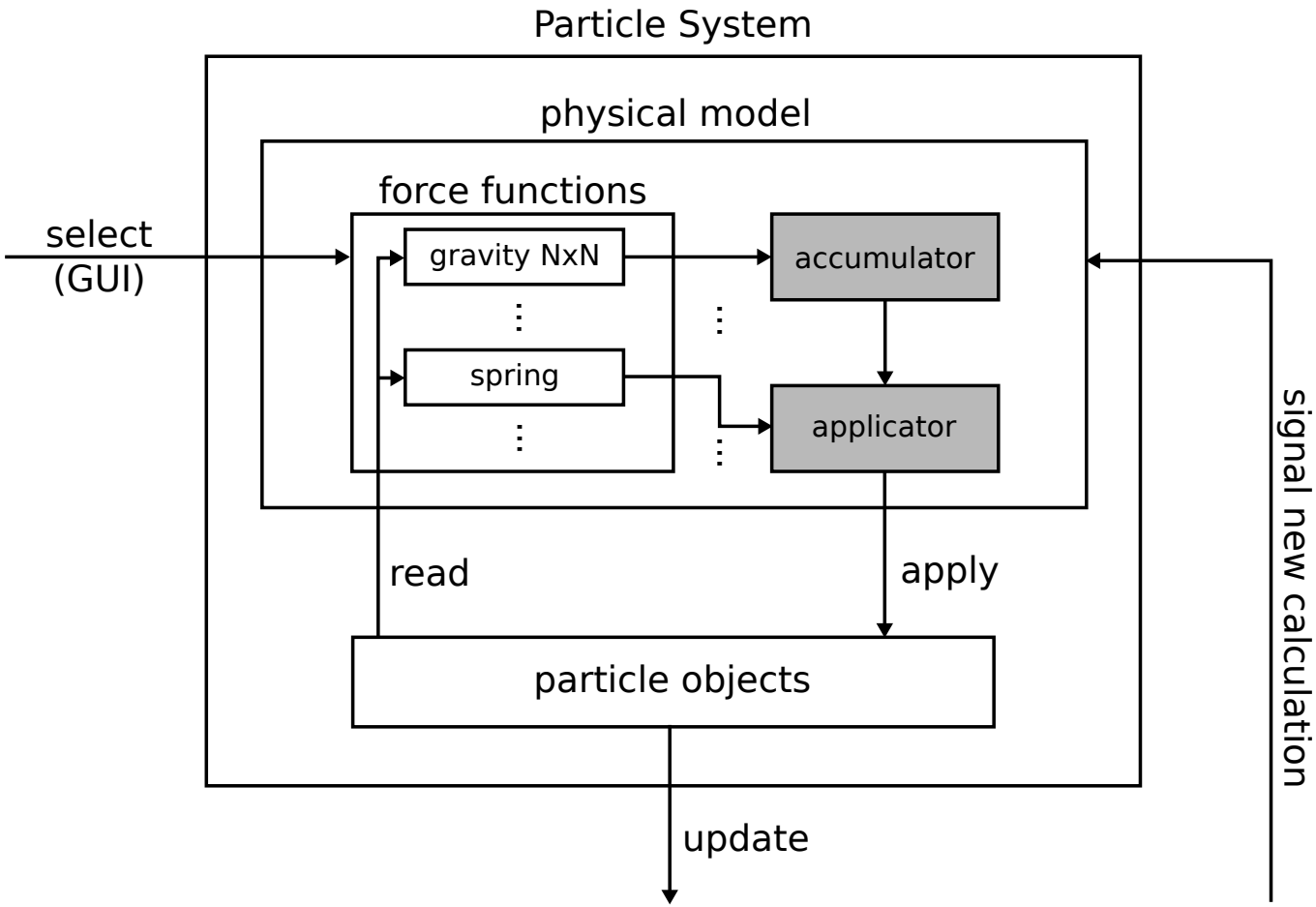

Figure: Externe vs Inter-Partikel Kräfte

Implementierung

Listing 1: Beispiel von Berechnung und Anwendung von Kräften

```
1  //iterate over all particles in the particle system
2  std::for_each(ps.begin(), ps.end(), [&ps](c3p::Particle&
       p)
3  {
4     // simple user-defined attraction force
5     p << calc_force(p, Particle(), [p](const Particle &,
          const Particle &)
6     {
7        glm::vec3 direction =
             glm::normalize(glm::vec3(0,0,0) - p.location());
8        return direction * 0.1;
9     });
10
11    // spring force from virtual particle at (0,0,0) to
         each particle
12    p << spring(p, Particle(0,0,0), {spring_constant,
         spring_length);
13
```

Implementierung

Listing 2: Beispiel von Berechnung und Anwendung von Kräften

```
1     ...
2
3      //gravitational forces between particles
4     p << c3p::accumulate(p,
5                          ps.particles(),
6                          {ps.g_constant()},
7                          c3p::gravity);
8  }
```

Implementierung

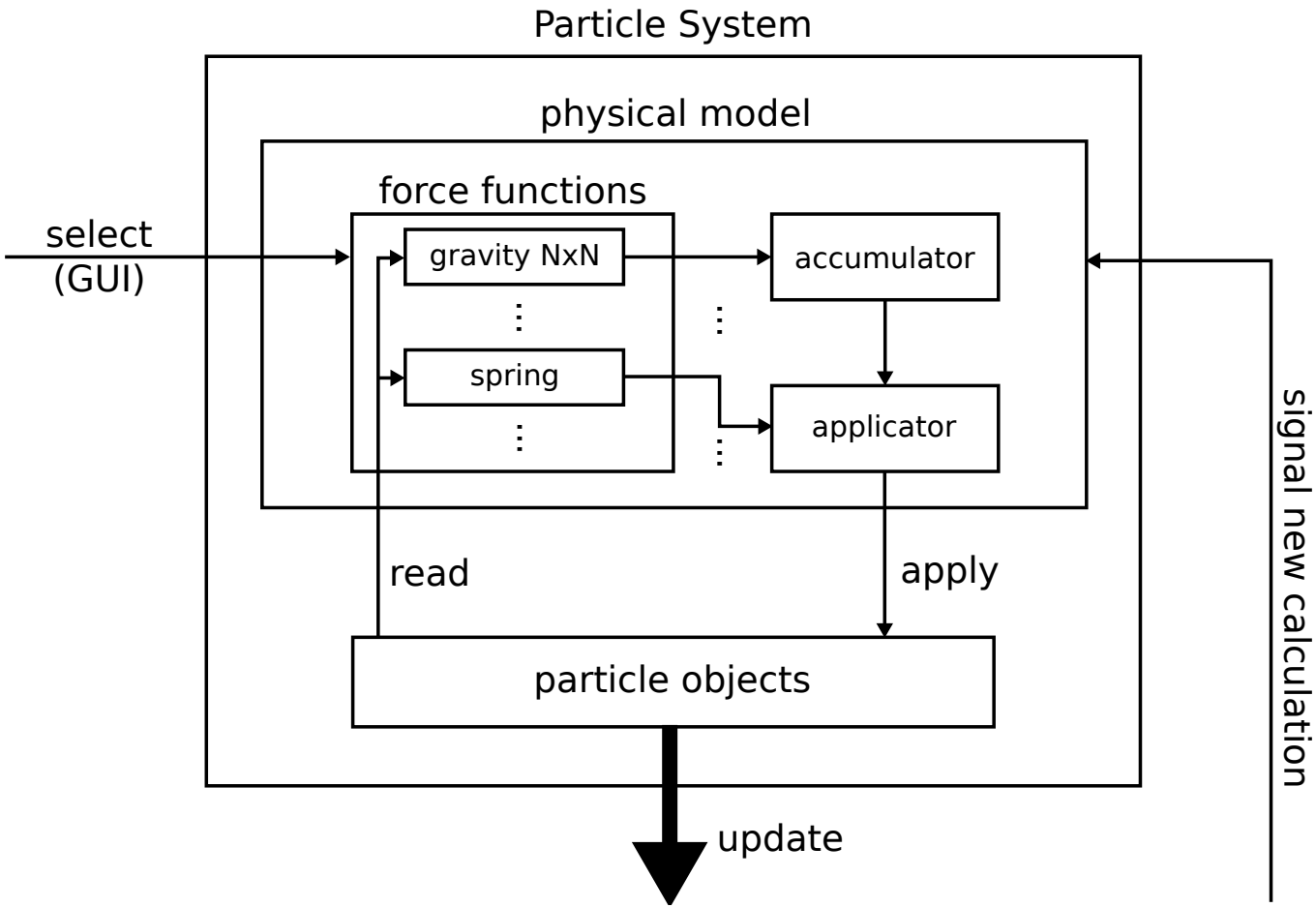

Figure: Aktualisieren der Geschwindigkeit und des Orts

Listing 3: Update Funktion

```
1  void update(Particle &p)
2  {
3     //v(t) = a*t + v(t-1)
4     p.velocity = p.acceleration * 1.0f + p.velocity;
          //deltaT = 1.0
5
6     //s(t) = (a*t^2)/2 + v(t) + s(t-1)
7     p.location = (p.acceleration * 1.0f) / 2.0f +
          p.velocity + p.location;
8
9     //acceleration is not accumulative, but recalculated
          at each time step
10    p.acceleration = {0, 0, 0};
11 }
```

Mögliche Optimierungen

- Parallelisierung der Iterationsschleifen über die Partikel

Mögliche Optimierungen

- Parallelisierung der Iterationsschleifen über die Partikel

- Offloading auf die GPU

Mögliche Optimierungen

- Parallelisierung der Iterationsschleifen über die Partikel

- Offloading auf die GPU

- Partitionierte, semi-symmetrische Kräftematrix

Zusammenfassung

Physik →

$$\vec{s}(t) = \frac{\vec{a} * t^2}{2} + C_v + C_s$$

$$F = m * a$$

$$\vec{F_{res}} = \vec{F_1} + \vec{F_2} + \ldots \vec{F_n}$$

Konzepte →

"Newton'sches Objekt"

"*NO << Kraft*"

"update(NO)"

"Kraft"

"calc_force(NO, NO, ff)"

"accumulate(f1, f2, ..)"

Implementierung