Praktikum 1 zu Parallele Programmierung

Lukas Rothenberger

TECHNISCHE UNIVERSITÄT DARMSTADT

Wintersemester 2024/2025
05. November 2024

Allgemeines

- Die Standardsprache im Quellcode ist Englisch, weswegen auch die Typen/Methoden so benannt sind.
- Es gibt Fußnoten, wenn Konzepte das erste Mal verwendet werden; dort können Sie sich weiter informieren.
- *Es gibt Hinweise (zu gutem Programmierstil oder anderem) in kursiver Schrift. Diese werden nicht bewertet.*
- Stellen Sie sicher, dass Ihre Abgabe auf dem Hochleistungsrechner der TU Darmstadt lauffähig ist, andernfalls ist mit Punktabzug zu rechnen.

Bepunktung der Aufgaben

- Bei jeder Teilaufgabe stehen die erreichbaren Punkte dabei, welche die Anzahl an funktionalen Tests für die jeweige Teilaufgabe angibt.
- Für jeden funktionalen Test gilt: Sollte der Test fehlschlagen, nicht kompilieren oder länger als **eine Minute** brauchen, erhalten Sie keinen Punkt dafür. Sollte der Test innerhalb von **einer Minute** korrekt durchlaufen, erhalten Sie dafür einen Punkt. Ein paar Tests stellen wir Ihnen bereits.

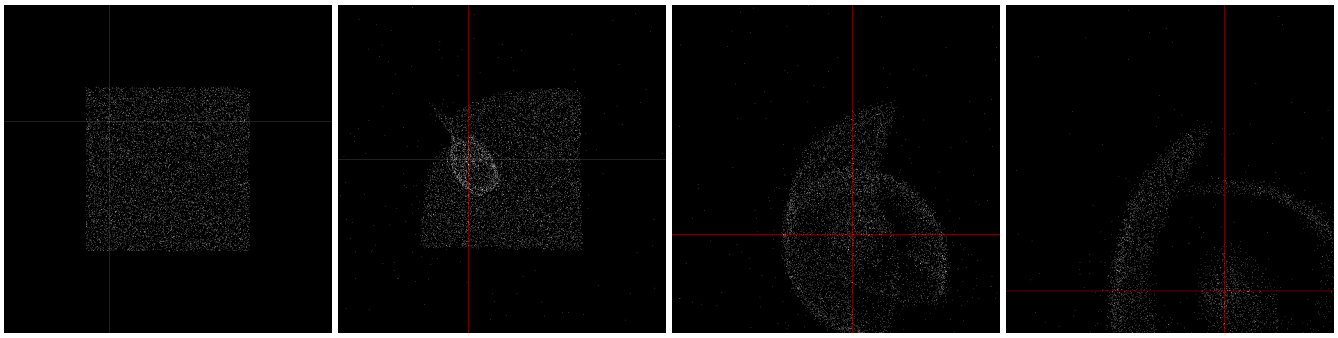
Zusätzliche Hinweise

- Nennen sie **keine** Ihrer Dinge (z.B. Funktionen, Dateien, Klassen) mit `ppws24` als Bestandteil vom Namen. Wir kodieren alle unserer Testfälle, Testbilder, Testfunktionen, etc. zufällig mit `ppws24` als zusätzlichen Schutz, damit sich diese nicht mit Ihren Dingen treffen.
- Ihnen steht frei weitere Funktionalität zu implementieren, solange nicht explizit anders geregelt.
- Achen Sie darauf, dass Sie nur standardkonformes C++ benutzen. Das Benutzen von compiler-, laufzeit-, oder betriebssystemspezifischen Erweiterungen und Funktionalitäten abseits der von uns vorgegebenen Dinge (Frameworks für die funktionalen Tests und die Laufzeittests, Benutzung von Typen wie `std::uint32_t` etc.) können zu Punktabzug führen.
- Bitte beachten Sie, dass die Tutor:innen keine bindenden Aussagen treffen.
- Beachten Sie, dass die Tutor:innen deren Sprechstunden Sie besuchen nicht notwendigerweise Ihre Abgaben bewerten.

Allgemeine Hinweise

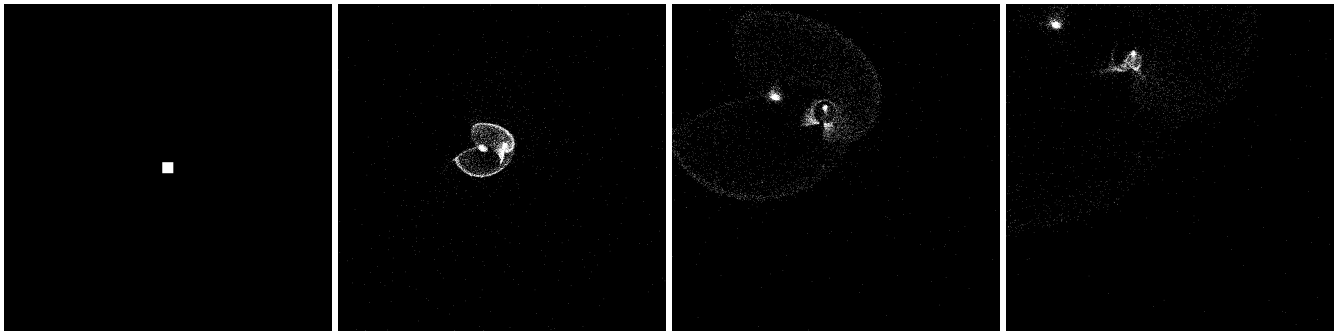
Wenn eine Methode einen Parameter nicht ändert, sollten Sie den als `const` klassifizieren – dies vermeidet Programmierfehler.[1]
Wenn eine Methode das aktuelle Objekt nicht ändert, sollten Sie diese als `const` klassifizieren. Dadurch sind die Methoden aufrufbar, auch wenn das Objekt als solches selbst `const` ist (siehe vorherigen Hinweis).[2]
Wenn eine Methode keine Exception werfen kann, sollten Sie diese als `noexcept` markieren.[3]
Wenn eine Methode einen Wert zurück gibt und ein logischer Fehler wäre, diesen nicht zu speichern, können Sie diese mit `[[nodiscard]]` markieren. Wird der Wert nicht beim Aufruf gespeichert, erzeugt dies eine Warnung.[4]

[1] `https://en.cppreference.com/w/cpp/language/cv`
[2] `https://en.cppreference.com/w/cpp/language/member_functions`
[3] `https://en.cppreference.com/w/cpp/language/noexcept_spec`
[4] `https://en.cppreference.com/w/cpp/language/attributes/nodiscard`



(a) Initialer Zustand (b) 200 simulierte Epochen (c) 600 simulierte Epochen (d) 900 simulierte Epochen

Abbildung 1: Beispielhafte Simulationsergebnisse eines zufällig erzeugten Systems von Himmelskörpern. Die Position eines sich bewegenden supermassiven Schwarzen Lochs ist in rot markiert. Die Kantenlänge der belebten Region im initialen Zustand beträgt $0.2Lj$. Epochenlänge von einem Monat.



(a) Initialer Zustand (b) 1500 simulierte Epochen (c) 5000 simulierte Epochen (d) 10000 simulierte Epochen

Abbildung 2: Beispielhafte Simulationsergebnisse eines zufällig erzeugten Systems von Himmelskörpern mit zwei supermassiven Schwarzen Löchern. Die größe des belebten Gebiets im initialen Zustand ist gleich der in Abbildung 1. Epochenlänge von einem Monat.

.

Aufgabe 1: Einführung

Das Ziel des ersten Praktikums wird es sein, eine simple, naive und sequentielle N-Body Simulation von Himmelskörpern (z.B. Planeten oder Sternen) zu implementieren. Beispielhafte Resultate einer solchen Simulation finden Sie in Abbildung 1 und 2 dargestellt.

Das Praktikum ist wie folgt strukturiert:

- Aufgabe 2 widmet sich der Vorbereitung der notwendigen Klassen
- Aufgabe 3 wird die Ausgabe und Darstellung der Ergebnisse behandeln
- Aufgabe 4 definiert die nötigen Grundlagen zur Berechnung der Physik
- Aufgabe 5 widmet sich der eigentlichen Simulation der Bewegungen der Himmelskörper

1a) Autoren

Wichtig: Tragen Sie in der Datei /source/authors_lab1.h zu jeder Teilaufgabe ein, wer sie bearbeitet hat!

1b) Kompilierungsfehler

Zu Beginn der Bearbeitung des ersten Praktikums ist mit Kompilierungsfehlern aufgrund fehlender bzw. noch nicht deklarierter und implementierter Klassen und Funktionen zu rechnen. Um die Zahl der dadurch erzeugten Compilerfehler zu reduzieren, können Sie z.B. temporär Teile der `main` Funktion in /source/main.cpp auskommentieren[5]. Sollten Sie weiterführende Probleme beim Kompilieren haben, wenden Sie sich bitte im Rahmen der Sprechstunden an die Tutoren.

[5] `https://en.cppreference.com/w/cpp/comment`

Aufgabe 2: Bereitstellungen der notwendigen (Daten-)Klassen (Gesamt: 6 Punkte)

Die Aufgabe beschäftigt sich hauptsächlich mit den Klassen um die Daten zu halten. Die hier geforderten Funktionen werden in genau dieser Form getestet; eine solche ist durch die `Teletype` Schrift gekennzeichnet. Ihnen steht frei, weitere Funktionalität zu implementieren.

2a) Vector2d - Klassenstruktur + Konstruktor + Index-Operator (1 Punkt)

Erstellen Sie die Klasse `Vector2d`, welche einen zweidimensionalen Vektor abbildet, in /source/structures/vector2d.h. Die Klasse soll zur späteren Verwendung mit einem `template` für Klassen versehen werden[6]. Über diesen Typparameter ('template' übersetzt) wird später der Typ übergeben, welcher für die Speicherung der einzelnen Komponenten verwendet wird. Stellen Sie einen `Standardkonstruktor`, sowie einen Konstruktor um beide Komponenten direkt zu setzen bereit. Letzterer soll zwei Argumente vom Typ des Typparameters entgegennehmen und die entsprechenden Komponenten des Vektors mit den gegebenen Werten initialisieren.

Implementieren sie den `Operator []`[7] um die Werte der einzelnen Komponenten auslesen zu können. Werfen Sie eine `std::exception` im Fall eines ungültigen Zugriffes. Stellen Sie die Methode `void set` bereit, welche zwei Argumente vom Typ des Typparameters entgegennimmt und die entsprechenden Komponenten des Vektors mit den gegebenen Werten überschreibt. Dabei soll das erste Argument der Komponente an Stelle 0, und das zweite Argument der Komponente an Stelle 1 zugewiesen werden.

Stellen Sie sicher, dass jedes Attribut stets initialisiert ist. Sie können beispielsweise Standardwerte direkt in Deklaration einfügen.
Normalerweise sind die privaten Felder einer Klasse unter den öffentlichen Feldern, da erstere 'nicht einsehbar' sind.
Stellen Sie den Typparameter als dependent name innerhalb der Klasse bereit.[8]

2b) Vector2d - Operatoren (1 Punkt)

Implementieren Sie die folgenden aus der Mathematik bekannten Operatoren der Klasse Vector2d:

- `+`: Addition zweier Vektoren
- `-`: Subtraktion zweier Vektoren
- `*`: Multiplikation des Vektors mit einem Skalar
- `/`: Division des Vektors durch ein Skalar

Implementieren Sie zusätzlich den den Gleichheitsoperator `==`. Zwei Vektoren sind genau dann gleich, wenn ihre Komponenten gleich sind.

Sie können alternativ zum Gleichheitsoperator auch den Spaceship-Operator bereitstellen: `<=>`[9]
Dieser bzw. andere können auch benutzt werden, um den Gleichheitsoperator implizit zu definieren.

2c) BoundingBox - Klassenstruktur und Konstruktor (1 Punkt)

Erstellen Sie die Klasse `BoundingBox` in /sources/structures/bounding_box.h. Diese soll die vier öffentlichen, nicht konstanten Felder `double x_min, x_max, y_min` und `y_max` sowie die in /sources/structures/bounding_box.cpp implementierten Methoden bereitstellen. Implementieren sie zudem einen Standardkonstruktor sowie einen Konstruktor, welcher Werte für die genannten Felder in der gegebenen Reihenfolge entgegennehmen und setzen kann.

[6] `https://en.cppreference.com/w/cpp/language/class`
[7] `https://en.cppreference.com/w/cpp/language/operators`
[8] `https://en.cppreference.com/w/cpp/language/type_alias`
[9] `https://stackoverflow.com/questions/47466358/what-is-the-spaceship-three-way-comparison-operator-in-c`

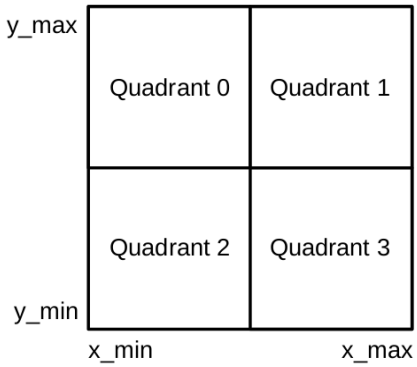


Abbildung 3: Zuweisungsschema der Quadranten.

2d) BoundingBox - contains + get_quadrant + throwing (1 Punkt)

Implementieren Sie die Methode `bool contains`, welche einen Positionsvektor vom Typ `Vector2d<double>` entgegennimmt und `true` zurückgibt, wenn diese innerhalb (inklusive der Grenzen) der BoundingBox liegt.

Richtungs- und Positionsangaben vom Typ `Vector2d<double>` sind grundsätzlich so zu interpretieren, dass die Komponente an der Stelle 0 der X-Richtung, und entsprechend Komponente 1 der Y-Richtung entspricht.

Stelle Sie zudem die Methode `get_quadrant` bereit, welche den index eines Quadranten als `std::uint8_t` entgegen nimmt und den entsprechenden Quadranten als Objekt vom Typ `BoundingBox` zurückgibt. Verwenden Sie zum Zuweisen der Quadranten das in Abbildung 3 dargestellte Schema. Die Grenzen der Quadranten dürfen dabei überlappen. Werfen Sie eine `std::exception` im Fall eines ungültigen Zugriffs.

2e) Universe - Datenstruktur (1 Punkt)

Im weiteren Verlauf des Praktikums wird es zur Aufgabe werden eine simple Simulation der Bewegung von Himmelskörpern zu implementieren. Erstellen Sie vorbereitend dafür die Klasse `Universe`, welche Beschreibungen der Himmelskörper verwalten soll. Genauer sollen die folgenden Felder bereitgestellt werden:

- `num_bodies` - Gesamtanzahl der Himmelskörper als `std::uint32_t`
- `current_simulation_epoch` - Anzahl der bereits simulierten Epochen als `std::uint32_t`
- `weights` - Massen der Himmelskörper als `double` in kg
- `forces` - Momentan auf die Körper wirkende Kräfte als `Vector2d<double>` in N
- `velocities` - Momentane Geschwindigkeiten der Körper als `Vector2d<double>` in m/s
- `positions` - Momentane Positionen der Körper als `Vector2d<double>` in m

Verwenden Sie Vektoren vom Typ `std::vector` um die jeweiligen Informationen der Körper zu verwalten. So soll Beispielsweise `weights` die Gewichte aller simulierten Körper beinhalten.

Im folgenden soll grundsätzlich gelten, dass die Reihenfolge der Informationen über die Himmelskörper in den bereitgestellten Vektoren stets identisch ist, und alle Vektoren somit stets die selbe Länge aufweisen. Beispiel: Ist die Masse eines bestimmten Körpers an Stelle n des Vektors `weights` abgelegt, so soll seine Position an Stelle n des Vektors `positions` zu finden sein.

Zusätzlich zu den genannten Feldern soll die Klasse einen Standardkonstruktor bereitstellen, welcher ein leeres Universum in Epoche Null abbildet.

Hinweis: Wir haben für Sie zu Testzwecken zwei Generatoren für Universen vorbereitet. Sie können diese via /source/input_generator/input_generator.h verwenden. Zudem sind beide Generatoren bereits in /source/main.cpp eingebunden und können mittels der --universe-generator flag umgeschaltet werden.

2f) Universe - get_bounding_box (1 Punkt)

Fügen Sie der Klasse `Universe` eine Methode `get_bounding_box` hinzu, welche keine Argumente annimmt und ein Objekt vom Typ `BoundingBox` zurückgibt. Die erzeugte BoundingBox soll alle Himmelskörper beinhalten, dabei aber die kleinst mögliche Größe aufweisen.

Zur Erinnerung: Richtungs- und Positionsangaben vom Typ Vector2d<double> sind grundsätzlich so zu interpretieren, dass die Komponente an der Stelle 0 der X-Richtung, und entsprechend Komponente 1 der Y-Richtung entspricht.

Aufgabe 3: Ausgabe des Systemzustandes (Gesamt: 3 Punkte)

3a) Sichern des Universums (1 Punkt)

In /source/utilities/import.hpp haben wir Ihnen die Funktion `load_universe` zum Einlesen eines Systemzustandes aus einer Datei bereitgestellt. Implementieren Sie die analoge, statische Funktion `void save_universe` in /source/utilities/export.hpp, welche den Pfad zur Ausgabedatei als `std::filesystem::path` sowie eine Referenz auf das zu sichernde Universum als Argumente bekommt, und den aktuellen Zustand des übergebenen Universums in der beschriebenen Datei sichert. Für Werte vom Typ `double` ist dabei eine Präzision von 6 Stellen nach dem Komma zu verwenden. Die erzeugte Datei muss der folgenden Struktur entsprechen, wobei n die Anzahl der Himmelskörper repräsentiert:

```
### Bodies
<n>
### Positions
<body_1.x> <body_1.y>
...
<body_n.x> <body_n.y>
### Weights
<body_1>
...
<body_n>
### Velocities
<body_1.x> <body_1.y>
...
<body_n.x> <body_n.y>
### Forces
<body_1.x> <body_1.y>
...
<body_n.x> <body_n.y>
```

Beispiele für gesicherte, vorgegebene Universen können Sie im Ordner /input finden. Diese können sie im Folgenden ebenfalls verwenden, um Mittels des `--load-universe-path` Kommandozeilenarguments im Gegensatz zu den standardmäßig zufällig generierten Universen reproduzierbare initiale Zustände zu erhalten.

3b) Plotten des Universums - mark_pixel, mark_position (1 Punkt)

In /source/plotting/plotter.h haben wir Ihnen eine Klassenstruktur bereitgestellt, welche im weiteren Verlauf zum Plotten des Systemzustandes verwendet werden können soll. Intern erzeugt diese ein Bitmap mittels der in /source/image bereitgestellen Klassen. Implementieren Sie die beiden Methoden `mark_pixel` und `mark_position` in /source/plotting/plotter.cpp gemäß der folgenden Spezifikationen.

mark_pixel: Setzt den Pixel im `BitmapImage image` an der durch `x` und `y` beschriebenen Position auf die durch `red`, `green` und `blue` beschriebene Farbe. Ist der beschrieben Pixel außerhalb des durch `plot_width` und `plot_height`

beschriebenen Bildes, werfen Sie bitte eine passende `std::exception`. Tipp: Bedenken Sie, dass das Zählen mit Null beginnt.

mark_position: Setzt den Pixel im `BitmapImage image` der die via `position` beschriebene Position im Universum repräsentiert auf die durch `red, green` und `blue` beschriebene Farbe. Ist die übergebene Position außerhalb des durch `plot_bounding_box` beschriebenen Bereichs des Universums soll keine Änderung des Bildes stattfinden.

3c) Plotten des Universums - highlight_position, add_bodies_to_image (1 Punkt)

Um die Resultate Ihrer Simulationen anschaulich und interpretierbar darstellen zu können, implementieren Sie bitte nun die beiden Methoden `highlight_position` und `add_bodies_to_image` in /source/plotting/plotter.cpp bzw. /source/plotting/universe.cpp gemäß der folgenden Spezifikationen.

highlight_position: Fügt ein Kreuz bestehend aus einer vertikalen und einer horizontalen Linie farbiger Pixel, beschrieben durch übergebene RGB-Werte, in das `BitmapImage image` ein, dessen Zentrum genau an der durch `position` beschriebenen Position im Universum liegt. Ist die übergebene `position` außerhalb des dargestellen Bereichs (`plot_bounding_box`) des Universums soll keine Änderung des Bildes stattfinden.

add_bodies_to_image: Fügt Himmelskörper in das `BitmapImage image` ein. Jeder Körper soll durch genau einen weißen (R:255, G:255, B:255) Pixel an der Position des Körpers repräsentiert werden. Mehrere Körper können dabei durch einen einzigen Pixel dargestellt werden. Befindet sich ein Körper außerhalb des dargestellten Bereichs (`plot_bounding_box`) des Universums soll keine Änderung des Bildes stattfinden.

Aufgabe 4: Physik (Gesamt: 2 Punkte)

Um die notwendigen physikalischen Berechnungen der simulierten Körper zu vereinfachen, sollen im folgenden die notwendigen Konstanten und Funktionen bereitgestellt werden.

4a) Gravitation (1 Punkt)

Die zur Berechnunge der Anziehungskraft zweier Körper notwendigen Konstanten und Funktionen sind in /source/physics/gravitation.h zu implementieren.

Definieren Sie zunächst die statische, Gravitationskonstante `gravitational_constant`, deren Wert an dieser Stelle als $G = 6.67430 * 10^{-11} \frac{m^3}{kg \cdot s^2}$ angenommen wird. Tipp: Sie können die Wissenschaftliche Notation[10] in C++ verwenden. Beachten Sie dabei, dass z.B. `1e-4` den Zahlenwert 10^{-4} repräsentiert.

Implementieren Sie die statische Funktion `double gravitational_force`, welche die Anziehungskraft zwischen zwei Körpern berechnet. Dazu nimmt Sie die folgenden Argumente in der dargestellten Reihenfolge entgegen:

- Masse m_1 des ersten Körpers in kg
- Masse m_2 des zweiten Körpers in kg
- Distanz d zwischen den beiden Körpern in m

Die Anziehungskraft F soll wie folgt berechnet werden: $F = G \cdot \frac{m_1 \cdot m_2}{d^2}$

[10] `https://cplusplus.com/reference/ios/scientific/`

4b) Mechanik (1 Punkt)

Analog zur Anziehungskraft sollen nun Funktionen zur Berechnung der Bewegungsmechanik bereitgestellt werden. Implementieren sie dazu die folgenden beiden statischen Funktionen in /source/physics/mechanics.h.

calculate_acceleration: Die Funktion besitzt einen `Typparameter T` und soll die auf einen Körper wirkende Beschleunigung als `Vector2d<T>` zurück geben. Sie erhält die auf einen Körper einwirkende Kraft F in N als `Vector2d<T>`, sowie die Masse des Körpers m in kg als Argumente. Die Beschleunigung a soll mittels $a = \frac{F}{m}$ berechnet werden.

calculate_velocity: Die Funktion besitzt einen `Typparameter T` und soll den Geschwindigkeitsvektor v eines Körpers als `Vector2d<T>` nach Beschleunigung mit a über eine Zeitspanne von t Sekunden berechnen. Die Funktion erhält die Ausgangsgeschwindigkeit des Körpers v_0 sowie a und t als Argumente. Die Geschwindigkeit v des Körpers nach der Beschleunigung kann wie folgt berechnet werden: $v = v_0 + a \cdot t$

Aufgabe 5: Naive N-Body Simulation (Gesamt: 9 Punkte)

Nachdem alle notwendigen Vorbereitungen getroffen wurden, kann nun die eigentliche Simulation der Interaktion der Himmelskörper in /source/simulation/naive_sequential_simulation.cpp implementiert werden.

Hinweis: Der Einfachheit halber werden keine Kollisionen von Körpern betrachtet. Daher ist damit zu rechnen, dass schwere Körper leichtere stark beschleunigen und "wegschießen", sofern sie sich zu nahe gekommen sind. Dieses Verhalten weicht von der Erwartung ab, dass der leichteren in den schwereren stürzt und eine Kollision stattfindet, wird an dieser Stelle jedoch als korrekt angesehen.

5a) Berechnen der wirkenden Kräfte (3 Punkte)

Zunächst widmen wir uns der naiven Berechnung der auf die Körper einwirkenden Kräfte. Implementieren Sie hierzu die Methode `calculate_forces`, welche die Anziehungskräfte zwischen jedem Paar von Körpern berechnet und die auf jeden Körper einwirkenden Kräfte aufsummiert. Für n Körper berechnet sich die auf einen Körper i einwirkende Kraft F_i also wie folgt:

$$F_i = \left(\sum_{j=1}^{i-1} f(i,j) \right) + \left(\sum_{j=i+1}^{n} f(i,j) \right)$$

, wobei $f(i,j)$ die Anziehungskraft zwischen Körper i und j beschreibt.

Die berechneten Kräfte sollen anschließend im Vector `forces` des als Referenz übergebenen Universums `universe` abgelegt werden. Achten Sie hier und in den folgenden Teilaufgaben insbesondere darauf, die Reihenfolge der Einträge in den Feldern von `universe` nicht zu ändern.

5b) Berechnen der Geschwindigkeiten (2 Punkte)

Definieren Sie zunächst in /source/simulation/constants.h die statische Konstante `double epoch_in_seconds`, welche die Dauer einer zu simulierenden Epoche beschreibt. Nehmen Sie als länge einer Epoche der Simulation $2.628 \cdot 10^6$ Sekunden an.

Implementieren sie die Methode `calculate_velocities`, welche die zuvor berechneten, auftretenden Kräfte verwendet um die Geschwindigkeitsänderungen aller Körper über eine simulierte Epoche zu berechnen. Berechnen Sie dazu zunächst die auf einen Körper wirkende Beschleunigung, und anschließend die neue Geschwindigkeit des Körpers. Sie können die aus /source/physics bekannten Funktionen verwenden. Die auftretenden Kräfte können innerhalb einer Epoche als konstant angenommen werden. Legen sie die berechneten Geschwindigkeiten analog zur vorherigen Aufgabe in `universe` ab.

5c) Berechnen der Bewegungen (2 Punkte)

Die Bewegungen aller Körper sollen von der zu implementierenden Methode `calculate_positions` bestimmt werden. Berechnen Sie dazu zunächst die Bewegung s eines Körpers innerhalb einer Epoche mittels $s = v \cdot t$, wobei v die

Geschwindigkeit des Körpers und t die Länge der Epoche in Sekunden darstellt. Um die neue Position p' eines Körpers am Ende einer Epoche zu berechnen, können Sie $p' = p_0 + s$ verwenden, wobei p_0 die Position des Körpers zu Beginn der Epoche darstellt. Legen sie die berechneten Positionen der Körper analog zur vorherigen Aufgabe in `universe` ab.

5d) Simulieren einer Epoche (1 Punkt)

Implementieren Sie die Simulation einer Epoche in `simulate_epoch`. Dazu sind die folgenden Schritte notwendig:

- Update der wirkenden Kräfte
- Update der Körpergeschwindigkeiten
- Update der Körperpositionen
- Inkrementieren des Zählers für simulierte Epochen des übergebenen Universums `universe`
- Erzeugen eines Systemplots, falls die Epochennummer ohne Rest durch `plot_intermediate_epochs` teilbar ist und `create_intermediate_plots` aktiviert ist. Sie können die Methode `write_and_clear` der Klasse `Plotter` verwenden, um die aktuelle Bitmap des Plotters in eine Datei zu schreiben. Dabei wird automatisch eine Seriennummer an die erzeugte Datei angehängt, und die interne Bitmap zurückgesetzt. Standardmäßig werden erzeugte Plots im Buildordner unter `simulation_results` abgelegt.

5e) Vollständige Simulation (1 Punkt)

Implementieren Sie schließlich die Methode `simulate_epochs`, welche für die konsekutive Simulation mehrerer Epochen verantwortlich ist. Das Argument `num_epochs` der Methode beschreibt dabei die Anzahl der zu simulierenden Epochen.

Aufgabe 6: Optional: Generieren eines Videos der Simulation

Falls Sie das Plotten von Zuständen während einer Simulation aktiviert haben, können Sie die erzeugten Bitmaps im Anschluss zu einem Video zusammenführen. Dieser Schritt ist optional und im Anschluss an die eigentliche Simulation durchzuführen. Sie können dazu zum Beispiel das von uns bereitgestellte, auf ffmpeg[11] basierende Script `create_mp4.sh` verwenden. Der von uns vorgeschlagene Weg ist:

- Wechseln in das Ausgabeverzeichnis der Simulation (Default: `simulation_results`)
- Ausführen des Scripts via `source <Pfad zum Script>`

[11] `https://www.ffmpeg.org/`

Hinweise zur Abgabe

Hier ein paar zusätzliche Formalitäten zu Ihrer Abgabe. Falls Sie diese nicht beachten, ist es möglich, dass Sie keine Punkte für eine oder alle Aufgaben erhalten.

Angabe der Autorschaft

Geben Sie in der Datei /source/authors.h an, wer von Ihnen an welcher Teilaufgabe mitgearbeitet hat. Mehrere Namen pro Teilaufgabe sind ok, trennen Sie diese z.B. mit Komma. Wir fordern eine Teilnahme von allen! Sollten Sie nicht an dem Praktikum mitgearbeitet haben (basierend auf den Angaben), erhalten Sie 0 Punkte auf das ganze Praktikum.

Hochladen des Quellcodes

Zippen Sie folgende Dateien und laden Sie das zip-Archiv in Moodle hoch:

- Den Ordner /cmake/
- Den Ordner /source/
- Die Datei CMakeLists.txt

Sie sind alle für die korrekte Abgabe verantwortlich. Zu spät eingereichte Dateien, Dateien mit fehlenden Lösungen, etc., liegen in allein Ihrer Verantwortung.

Die Ausführbarkeit der Abgabe auf dem Lichtenberg-Hochleistungsrechner ist entscheidend, da dieses System zur Bewertung verwendet wird. Sollte Ihre Abgabe auf dem Lichtenberg-Hochleistungsrechner nicht kompilierbar und ausführbar sein, ist mit Punktabzug zu rechnen.

Hinweise zum Lichtenberg

Der Lichtenberg-Hochleistungsrechner ist aufgeteilt in sogenannte Loginknoten und Rechenknoten, wobei sich alle das gleiche Dateisystem teilen. Erstere sind mit **ssh** und **scp** zu erreichen und werden benutzt um Rechenaufgaben für letztere zu kreieren. Wenn Sie eine sinnvolle Linux-Distribution verwenden, kann /bin/bash nativ **ssh** und Sie können das Dateisystem des Lichtenberg direkt in Ihrem Explorer (z.B. Nemo) einbinden. Wenn Sie Windows benutzen, empfiehlt sich das Terminal bzw. PuTTy für **ssh** und WinSCP für **scp**.

Sobald Sie eingeloggt sind, sollten Sie zusätzliche Softwarepakete laden. Das erreichen Sie beispielsweise mit **module load git/2.40.0 cmake/3.26.1 gcc/11.2.0 cuda/11.8 boost/1.81.0**. Kopieren Sie die Dateien auf den Hochleistungsrechner und kompilieren Sie dort das Programm.

Wenn Sie eine Rechenaufgabe lösen möchten, müssen Sie dafür SLURM benutzen. Effektiv führen Sie den Befehl **sbatch <filename>** aus, wobei <filename> hier eine Datei ist, die, sobald Ihr Programm Zeit und Resourcen zugeteilt bekommen hat, auf den Rechenknoten ausgeführt wird. Ein beispielhaftes Skript sieht so aus:

```
#!/bin/bash

#SBATCH -J parprog
#SBATCH -e /home/kurse/kurs00084/<TUID>/stderr/stderr.parprog.%j.txt
#SBATCH -o /home/kurse/kurs00084/<TUID>/stdout/stdout.parprog.%j.txt
#SBATCH -C avx512
#SBATCH -n 1
#SBATCH --mem-per-cpu=1024
#SBATCH --time=5
#SBATCH --cpus-per-task=1
#SBATCH -A kurs00084
#SBATCH -p kurs00084
#SBATCH --reservation=kurs00084

module load git/2.40.0 cmake/3.26.1 gcc/13.1.0 cuda/11.8 boost/1.81.0

echo "This is job $SLURM_JOB_ID"
```

Der Kurs ist ab dem 05. November 2024 freigeschaltet. Alle Informationen erhalten Sie auch hier: `https://www.hrz.tu-darmstadt.de/hlr/nutzung_hlr/zugang_hlr/loginknoten_hlr/index.de.jsp`

Sollten Sie große Ausgabedaten erzeugen, legen Sie diese bitte in `/work/scratch/kurse/kurs00084/TU-ID` ab, um Quotaüberschreitungen in `/work/kurse/kurs00084` zu vermeiden.
Wichtig: Dateien auf `/work/scratch/kurse/kurs00084/TU-ID` werden 8 Wochen nach der Erstellung gelöscht. Zudem wird von dem Verzeichnis kein Backup angelegt.

Einrichten einer alternativen Entwicklungsumgebung zum Lichtenberg Hochleistungsrechner

Die von uns vorgeschlagene, alternative Entwicklungsumgebung basiert auf Linux Cinnamon Mint 22 [12]. Zur Einrichtung der Referenzumgebung empfehlen wir das Erstellen einer virtuellen Maschine, zum Beispiel mittels VirtualBox[13]. Die folgenden Schritte zur Einrichtung können ebenfalls unter den meisten Ubuntu-basierten Distributionen ausgeführt werden.

- Installieren Sie ein geeignetes Betriebssystem (möglicherweise in einer virtuellen Maschiene) und schließen Sie die Installation ab
- Öffnen Sie eine Kommandozeile (Standard: STRG + ALT + T)
- Aktualisieren Sie ihr System mittels:
  ```
  sudo apt update
  sudo apt upgrade
  ```
- Installieren Sie die zum kompilieren des Praktikums benötigten Pakete:
  ```
  sudo apt install build-essential git cmake -y
  ```
- Installieren Sie die zum Bearbeiten von Aufgabe 6 benötigten Pakete:
  ```
  sudo apt install ffmpeg -y
  ```

[12] https://linuxmint.com/edition.php?id=316
[13] https://www.virtualbox.org/