

Aufgabenblatt 11

Allgemeine Hinweise:

- Für die Aufgaben auf diesem Übungsblatt müssen Sie am 12.02. votieren.
- Infos zur Klausur:
 - Bis zum 17. Februar werden alle wichtigen Informationen bezüglich der Klausur kommuniziert. Sie werden die Informationen per Email erhalten. Außerdem werden die wichtigsten Informationen auf dem Klausurvorbereitungszettel zusammengefasst.
 - Falls Sie an der Klausur teilnehmen möchten müssen Sie sich in Müsli für die Klausur anmelden. Die Anmeldung finden sie (hoffentlich) unter <https://muesli.mathi.uni-heidelberg.de/lecture/view/1248>
 - Die Anmeldung muss bis zum Sonntag den 28. Februar erfolgen. In der oben angekündigten Email werden Sie natürlich noch mal auf die Anmeldung hingewiesen.

Aufgabe 1: Simulation von Mehrkörperproblemen

(3 Punkte)

Für diese Aufgabe gibt es jeweils einen Votierpunkt für die Teilaufgaben a), b) und c). Für die Implementierung einer der Teilaufgaben d), e) oder f) gibt es einen Bonus-Votierpunkt. Sie können maximal einen Bonus-Votierpunkt erreichen.

In unserem Sonnensystem bewegt sich eine Vielzahl von Himmelskörpern unter gegenseitiger Wechselwirkung, der Gravitation. Der Versuch, die Bahnbewegung der Planeten (oder anderer Objekte) vorherzusagen, führt mathematisch auf ein Mehrkörperproblem: mehrere punktförmige Massen, deren Beschleunigung jeweils von der Position der anderen Punktmassen abhängt.

Während das Zweikörperproblem (z.B. Erde — Sonne) noch analytisch lösbar ist, tritt ab drei Körpern chaotisches Verhalten auf, und die Bahnen lassen sich nur noch näherungsweise mit numerischen Methoden berechnen. In dieser Aufgabe wollen wir uns damit näher beschäftigen.

Hinweis: Für die Darstellung der Simulation wird die SDL-Bibliothek ¹ benötigt. Installationsanweisungen für diese Bibliothek finden Sie, sofern nötig, im zugehörigen Wiki ².

Unter Ubuntu können sie die Bibliothek mittels

```
1 apt install libsdl2-dev
```

installieren.

Sie können CMake verwenden, um die SDL-Bibliothek automatisch in Ihr Projekt einzubinden. Linken Sie dazu Ihr Programm mit der SDL-Bibliothek, indem Sie folgende Zeile in `CMakeLists.txt` einfügen:

```
1 target_link_libraries(<Programmname> SDL2)
```

Alternativ können Sie auch von Hand kompilieren:

```
1 g++ -o <Programmname> <Quelldateien> -lSDL2
```

- (a) **Vorbereitung und Visualisierung:** Schreiben Sie zunächst eine `class Point`, wie wir es schon öfters in Aufgaben hatten, und eine `class Body`, das einen Massepunkt repräsentiert. Ein solcher

¹<https://www.libsdl.org/>

²<https://wiki.libsdl.org/Installation>

Massepunkt soll zwei `Point` haben, einen für die Position und einen für die Geschwindigkeit, ein `double` für die Masse, und zusätzlich ein Array aus drei `ints`, die dem Punkt eine Farbe aus dem RGB-Farbraum zuordnen.

Erweitern Sie die Klasse `Point` mit eigenen Operatoren, so dass alle folgenden Ausdrücke funktionieren:

```

1      Point a = ...;
2      Point b = ...;
3      double d = ...;
4      a = a + b;
5      a = a - b;
6      a += b;
7      a -= b;
8      a = d*b;
9      a = b*d;
10     a *= d;
11     std::cout << a << std::endl;

```

Für die Operatoren, die rechnen und zuweisen, orientieren Sie sich am Beispiel von Multiplikation mit einem Skalar:

```

1      class Point {
2          ...
3      public:
4
5          Point& operator*=(double d)
6          {
7              _x *= d;
8              _y *= d;
9              return *this;
10         }
11     }

```

Es ist wichtig, dass Sie den Rückgabewert korrekt angeben!

Auf der Homepage finden Sie den Header `sdlwrapper.hh`, der eine Klasse `SDLCanvas` zur Verfügung stellt. Machen Sie sich mit der Funktionalität dieser Klasse vertraut, binden Sie sie in Ihr Programm ein, und erzeugen Sie einen Canvas. Erzeugen Sie einige Massepunkte mit Positionen, Geschwindigkeiten, Massen, und zugeordneten Farben. Unten finden Sie dafür zwei Beispiele. Benutzen Sie die bereitgestellte Klasse, um die Punkte zu visualisieren. Schreiben Sie dazu eine Funktion `displayBodies()`, die auf den Datentyp der Massepunkte templatisiert ist und `Canvas` und Massepunkte als Argumente bekommt. Nutzen Sie darin die Methoden `drawPixel()` und `display()` des `Canvas`-Objekts. Damit sich das Fenster nicht sofort schließt, müssen Sie *nach* dem Aufruf ihrer Funktion eine `while`-Schleife einfügen, die abwartet, bis die Escape-Taste gedrückt wurde (rufen Sie die Methode `windowClosed` auf, um herauszufinden, ob Escape gedrückt wurde).

Hinweise:

- Sie müssen hier wieder kontinuierliche Koordinaten in Pixelpositionen umrechnen, wie wir das bereits in früheren Aufgaben hatten. Wenn Sie ein Fenster wählen, das mindestens die Größe 1024×768 hat, können Sie die Koordinaten der Beispiele unten auch direkt verwenden, ohne umskalieren zu müssen. Sie müssen dann lediglich den Punkt $(0,0)$ in die Bildmitte verschieben. Achten Sie außerdem darauf, dass der Ursprung links oben ist: die x -Achse geht also wie gewohnt nach rechts, die y -Achse zeigt aber nach unten.
- Hier zwei Beispiele von Massepunkten, die Sie zum testen verwenden können. Das erste erzeugt ein Planetensystem mit (näherungsweise) geschlossenen Bahnen:

```

1      std::vector<Body> bodies = {

```

```

2         {{ 0., 0.}, { 0., 0.}, 1e3, {255, 0, 0}},
3         {{ 100., 0.}, { 0.,0.3}, 10., { 0,255, 0}},
4         {{-200., 0.}, { 0.,0.2}, 10., { 0, 0,255}},
5         {{ 0.,250.}, {-0.25, 0.}, 10., {255,255, 0}},
6     };

```

Das zweite Beispiel erzeugt vier Massenpunkte, die aufgrund der Symmetrie für alle Zeiten umeinander kreisen sollten. Warum geht das nach einiger Zeit schief?

```

1     std::vector<Body> bodies = {
2         {{ 150., 0.}, { 0.,-0.2}, 1e3, {255, 0, 0}},
3         {{ -150., 0.}, { 0., 0.2}, 1e3, { 0,255, 0}},
4         {{ 0., 150.}, { 0.2, 0.}, 1e3, { 0, 0,255}},
5         {{ 0., -150.}, {-0.2, 0.}, 1e3, {255,255, 0}},
6     };

```

(b) **Numerische Simulation und Schwerpunktsystem:** Schreiben Sie eine Funktion

```

1     template<typename Force, typename Body>
2     void eulerStep (const Force& force, std::vector<Body>& bodies,
3     double t, double dt)

```

die die Bewegung der Massepunkte im Zeitintervall $[t, t+dt]$ mit dem Euler-Verfahren simuliert. Nehmen Sie dazu an, dass Force ein Funktor mit folgendem Interface ist:

```

1     [] (const auto& bodies, int i, double dt)
2     {
3         // bodies is a std::vector<Body>, so you can iterate over it
4         // with a range-based for loop and access the i-th element
5         // with bodies[i]
6         double force = ...;
7         return force;
8     }

```

der die Beschleunigung a_i des i -ten Massepunktes liefert. Führen Sie ein Update der Positionen x_i und Geschwindigkeiten v_i der Massepunkte gemäß der folgenden Formeln durch:

$$x_i(t + \delta t) = x_i(t) + v_i(t) \cdot \delta t, \quad v_i(t + \delta t) = v_i(t) + a_i(t) \cdot \delta t$$

Je nachdem, wie die Verteilung der Anfangsgeschwindigkeiten und der Massen gewählt ist, driftet die Ansammlung aus Masseteilchen langsam aus dem Canvas. Das liegt daran, dass der Schwerpunkt

$$\bar{x}(t) = M^{-1} \cdot \sum_i m_i x_i(t), \quad M = \sum_i m_i$$

sich mit konstanter Geschwindigkeit fortbewegt. Verschieben Sie die Position der Teilchen so, dass der Schwerpunkt nach jedem Schritt in der Mitte des Canvas liegt (sie rechnen dann im Schwerpunktsystem).

Zum Testen können Sie einen Funktor verwenden, der einfach immer 0 zurückgibt.

(c) **Kräfte und Interaktion:** Schreiben Sie nun einen Funktor, der das obige Interface erfüllt. Wenn nötig, können Sie natürlich noch Variablen in die Capture-Liste einfügen. Der Funktor soll die Gravitation berechnen, die alle Punkte auf den i -ten Punkt ausüben. Die Kraft, die das j -te Teilchen auf das i -te ausübt, ist

$$F_{ij} = Gm_i m_j \|x_j - x_i\|^{-3} (x_j - x_i),$$

und daher ist die verursachte Beschleunigung

$$a_{ij} = Gm_j \|x_j - x_i\|^{-3} (x_j - x_i).$$

Dabei ist G die Gravitationskonstante, deren Wert von Ihnen angepasst werden kann. Beachten Sie, dass die Masse m_i jeweils auf sich selbst keine Wirkung hat, und daher bei der Berechnung ausgelassen werden muss.

Schreiben Sie eine **while**-Schleife, die abwechselnd die neuen Positionen und Geschwindigkeiten berechnet und die Positionen der Massenpunkte graphisch darstellt. Experimentieren Sie dazu mit der Gravitationskonstante G , der Schrittweite δt und der Größenordnung der Anfangsgeschwindigkeiten. Für die oben angegebenen Beispiele verwenden Sie $G = 0,01$ und $\delta t = 1$. Finden Sie eine weitere Konstellation aus einem sehr schweren Zentralgestirn und drei bis vier weiteren Objekten, die ein stabiles Sonnensystem darstellen.

Hinweis: Wenn Sie nach jeder Iteration die Methode `clear` aufrufen, wird stets nur die aktuelle Position der Punkte dargestellt. Falls Sie den Aufruf weglassen, bleiben bisher gezeichnete Pixel mit ihrer Farbe erhalten, so dass die Trajektorien der Massenpunkte sichtbar werden.

Bonus: Weitere Möglichkeiten für Wechselwirkungen. Neben der hier angesprochenen Gravitationswechselwirkung können Sie Ihren Programmcode ohne große Änderungen für viele weitere Simulationen verwenden. Dazu müssen Sie lediglich den `Force`-Templateparameter austauschen. Schreiben Sie Funktoren für folgende weiteren Kräfte (für die Beschleunigung müssen Sie jeweils durch m_i teilen):

- (d) Die Schwerkraft $F_g = m_i \cdot g$. Die Massenpunkte sind hier keine Planeten, sondern kleine Partikel oder Regentropfen. Setzen Sie Punkte, die unten aus dem Canvas fallen, in diesem Fall mit Geschwindigkeit Null am oberen Rand wieder ein.
- (e) Die elastische Kraft $F = -k \cdot \delta x_i$. Der Term δx_i ist dabei die Distanz von x_i zu einem Referenzpunkt. Falls dieser konstant ist, schwingen die Punkte unabhängig voneinander um ihren Punkt. Man kann aber auch einige der Punkte (oder alle) als Referenzpunkte voneinander festlegen, was zu gekoppelten Schwingungen führt.
- (f) Die elektromagnetische Kraft $F_{em} = m_i \cdot (E + v_i \times B)$. Hier übernimmt m_i die Rolle der Ladung statt Masse, und kann auch negativ gewählt werden. Das elektrische Feld E und das magnetische Feld B sind vorgegebene Vektorfelder. Damit die Bewegung im Zweidimensionalen stattfindet, sollte E in der Bildebene und B senkrecht dazu liegen. Diese Kraft können Sie auch mit der obigen Kraft F_{ij} kombinieren — sie steht dann für die elektrodynamische Wechselwirkung zwischen den Punkten, statt für die Gravitation.