Programmierkurs
Dr. Steffen Müthing
IWR, Universität Heidelberg

Aufgabenblatt 12
Abgabe 04. Februar 2019
ipk-exercises:2018-ws-48-gbd459e7

Aufgabenblatt 12

Allgemeine Hinweise:

- Für die Aufgaben auf diesem Übungsblatt müssen Sie am 04.02. votieren.

Aufgabe 1

Simulation von Mehrkörperproblemen

1 + 1 + 1 Punkte + 3 Bonuspunkte

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: Sie können auf jede der Teilaufgaben jeweils einzeln votieren.

Hinweis: Für die Darstellung der Simulation wird die SDL-Bibliothek [1] benötigt. Installationsanweisungen für diese Bibliothek finden Sie, sofern nötig, im zugehörigen Wiki [2]. Wenn Sie die von uns verteilte virtuelle Maschine verwenden, können Sie die Bibliothek und die Entwicklungspakete mit folgendem Befehl installieren, falls noch nicht geschehen:

```
[ipk@localhost ~]$ sudo dnf install SDL2-devel
```

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:

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

Alternativ können Sie auch von Hand kompilieren:

```
[ipk@localhost ~]$ ipkc++ -lSDL2 -o <Programmname> <Quelldateien>
```

(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 representiert. Ein solcher 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 `char`s, die dem Punkt eine Farbe aus dem RGB-Farbraum zuordnen.

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

```
Point a = ...;
Point b = ...;
double d = ...;
a = a + b;
a = a - b;
```

[1] https://www.libsdl.org/
[2] https://wiki.libsdl.org/Installation

```
a += b;
a -= b;
a = d*b;
a = b*d;
a *= d;
std::cout << a << std::endl;
```

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

```
class Point {
  ...
public:

  Point& operator*=(double d)
  {
    _x *= d;
    _y *= d;
    return *this;
  }
}
```

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, z.B. mit Hilfe des bereits zur Verfügung gestellten Headers `io.hh`, hundert zufällige Massepunkte innerhalb des Canvas, deren Geschwindigkeiten, Massen und zugeordnete Farben ebenfalls zufällig sind. 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. 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.
- Um später gute Parameter für die Simulation zu finden, sollten Sie Ihr Programm so schreiben, dass man die Parameter für die Zufallsverteilung einfach ändern kann und insbesondere für Positionen und Geschwindigkeiten der Punkte unterschiedliche Verteilungsparameter verwenden kann.

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

```
template<typename Force, typename Body>
void eulerStep (const Force& force, std::vector<Body>& bodies,
                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:

```
[](const auto& bodies, int i, double dt)
{
  // bodies is a std::vector<Body>, so you can iterate over it
  // 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 Graviation 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 hier angepasst werden kann, um eine sinnvolle Simulation zu bekommen. 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. Finden Sie eine 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.

Weitere Möglichkeiten für Wechselwirkungen (Bonus-Votierpunkte): 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 Massepunkte 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 $F = -k\delta\cdot x_i$. 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 $F_{\text{em}} = m_i \cdot (E + v_i \times B)$. 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.