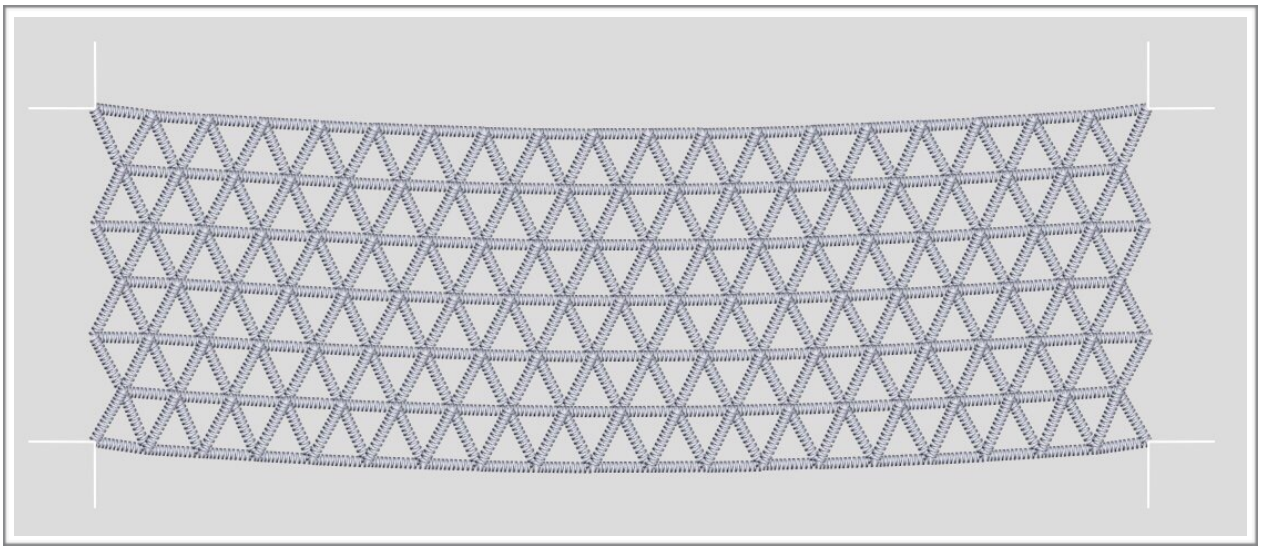
Многочастичный симулятор



Классическая механика

Егор Старобинский

Весна 2014

Оглавление

Многочастичный симулятор

Классическая механика

Аннотация

Рассматриваемой задачей этого проекта является определение поведения задаваемой механической двухмерной системы[1] из частиц, сил и ограничений. Для этого был создан интернет-сайт с программой, позволяющей найти решение уравнения движения системы (физический движок). Также на базе этой программы был реализован пользовательский интерфейс с возможностями создания и редактирования исходной системы и визуализацией её поведения.
В процессе решения широко применялись знания из классической механики и программирования, благодаря чему была реализована возможность моделирования тел из большого числа частиц и ограничений (интерес с точки зрения механики) и консоль управления с упрощёнными интерпретируемыми командами (интерес с точки зрения программирования). Был внедрён ряд математических методов как для вычисления решения уравнения движения, так и для анализа разрешённой системы. Были проведены проверки получившегося движка на предварительно решённых задачах.

Объединение физического движка и пользовательского интерфейса получило название "Многочастичный симулятор", имеет открытый исходный код и выполнено без использования готовых решений по теме проекта.

Научная новизна

Созданный движок позволяет человеку без специальных знаний в области программирования проводить моделирование собственных систем. Не требуется установки никакого дополнительного софта,

программа запускается при помощи браузера как на компьютерах, так и на телефонах, планшетах, телевизорах класса Smart.

Визуализация не применяет технологию WebGL, благодаря чему многократно увеличивается диапазон устройств, способных запустить сайт с полноценной функциональностью.

Применён метод нахождения периода движения частицы по участку траектории путём последовательного разбиения поля на сетки с возрастающей плотностью ячеек (идея позаимствована из теории слов).

Преимущества реализации

Движок представляет все тела как набор частиц со связями и применяет базовый двухшаговый метод численного интегрирования Верле и другие математические методы для разрешения уравнения движения в короткое время.

По возможности упрощено интегрирование в систему новых алгоритмов анализа (требует знаний в программировании на javascript как для реализации алгоритма, так и для его интегрирования), при этом все пользователи оперируют только актуальной версией программы.

Формулировка задачи

Цель работы

Создание интернет-сайта, позволяющего пользователю моделировать многоточечную систему онлайн.

Решаемые задачи:

- решение уравнения движения;
- визуализация.

Общие сведения по теме

Уравнение движения

Пусть мы наблюдаем тело в момент времени t.
Хотим знать, где окажется тело через малое изменение времени - Δt.

Рассмотрим базовый метод интегрирования Верле:
$x(t + \Delta t) = 2\,x(t) - x(t - \Delta t) + R(t)\Delta t^2 / m$, где
x - позиция точки,
R - равнодействующая всех сил, действующих на тело,
m - масса тела,
t - текущий момент времени,
Δt - малое изменение времени.

Метод Верле позволяет вычислять траекторию по упрощённой схеме: зная предыдущее и текущее положения ($x(t - \Delta t)$ и $x(t)$ соответственно) и мгновенное значение равнодействующей приложенных сил в текущем положении $R(t)$.

Достоинства метода: самокоррекция и бóльшая точность по сравнению с численным методом Эйлера.

Язык реализации: JavaScript.

Визуализация

Язык рализации: pure SCSS.

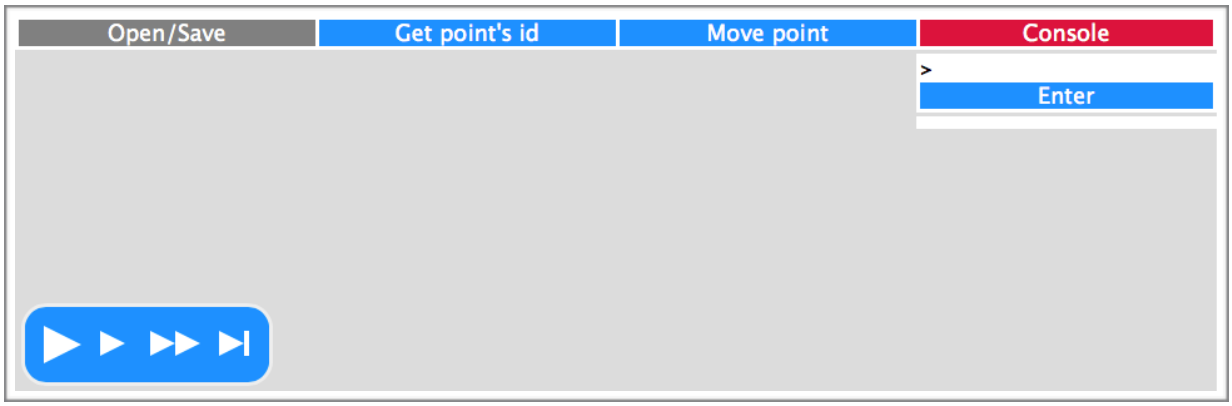
Обработка событий: JavaScript.

Манипуляции с DOM: jQuery (безболезненно заменяется на Zepto).

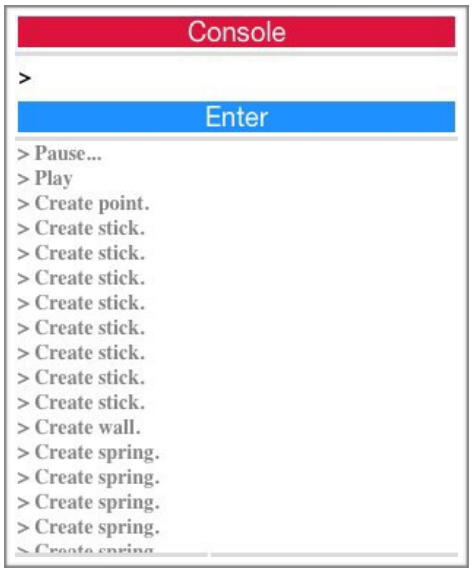
Отказ от WebGL продиктован выбором методов оптимизации для возможности работы с тысячами частиц.

Решение

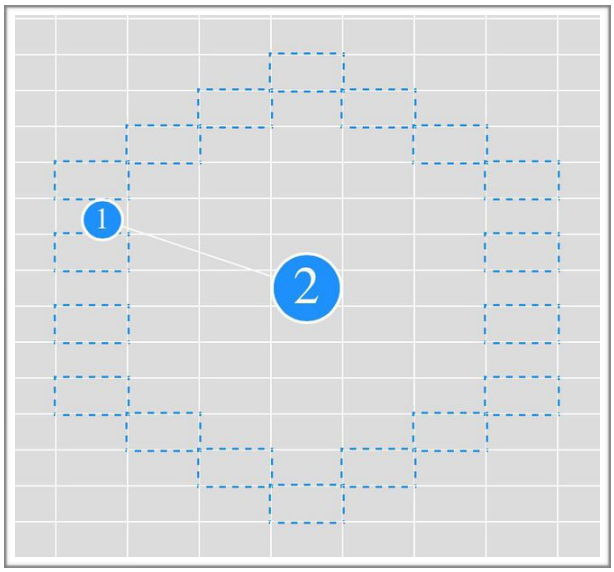
Скриншоты программы



Рабочее окно



Пример вывода консоли



Нахождение периода в простом движении

Элементы системы:

- Частицы;
- Стержни и пружины[2];
- Стенки;
- Поле сил;
- Рабочее окно;
- Сетки разметки;
- Консоль;
- Плеер.

Возможности консоли

- Конфигурация начальной системы тел;
- Изменение системы в процессе работы ("на лету");
- Запуск алгоритмов анализа системы;
- Распознавание и вывод ошибок в пользовательских запросах и в исходном коде;
- Распознавание и вывод предупреждений в пользовательских запросах и в исходном коде;
- Подключение/отключение сеток разметки, в том числе с пользовательскими размерами ячейки;
- Тетрис.

Команды консоли

Координаты пишутся в декартовой системе (x,y), единица измерения - пиксели, орт x направлен от левого края к правому, орт y от верхнего края к нижнему. Пример: (0,100) - координаты точки, лежащей на левом краю экрана на 100 пикселей ниже верхней границы.

Консоль выводит каждое сообщение как одну строку. Если доступной длины строки не хватает, сообщение обрезается. Для отображения полной версии сообщения необходимо кликнуть по нему мышью.

Примеры основных запросов

Очистить поле консоли

~ clrscr

Отобразить статистику элементов системы и число тиков:

~ getInfo

Создать частицу

~ addPoint (100,100) (0,10) 80 5[3][4], где

(100,100) - текущие координаты

(0,10) - вектор скорости относительно начала координат

80 - радиус частицы в пикселях

5 - масса частицы в у. е.

Задать вектор скорости (относительно начала координат)

~ setVelocity #0 (10,10), где

0 - id частицы[5]

(10,10) - новый вектор скорости относительно начала координат

Переместить частицу (относительно начала координат)

~ movePoint #3 (100,100) saveV[3][6], где

3 - id частицы[5]

(100,100) - новые координаты

saveV - флаг сохранения скорости. Если указан, частица после перемещения сохранит вектор своей скорости.

Задать массу

~ setProp #0 mass 10, где

0 - id частицы[5]

10 - новая масса в у. е.

~ setProp #0 invmass 0.1, где

0 - id частицы[5]

0.1 - обратное значение новой массы

Задать радиус

~ setProp #0 radius 100, где

0 - id частицы[5]
100 - радиус частицы в пикселях

Создать пружину между частицами
~ addSpring #0 #1 50[3][6][7], где
0, 1 - id частиц[5]
50 - жёсткость пружины в у. е.

Изменить жёсткость пружины
~ changeSpring #5 10, где
5 - id пружины[5]
10 - новая жёсткость в у. е.

Изменить жёсткость одинаковых пружин
~ changeSprings 10->15, где
10 - текущая жёсткость в у. е.
15 - новая жёсткость в у. е.

Изменить жёсткость всех пружин
~ changeSprings all->20, где
20 - новая жёсткость в у. е.

Создать стержень между частицами
~ addStick #2 #1[6], где
2, 1 - id частиц[5]

Отключение гравитации
~ gravity disable

Задать вектор ускорения свободного падения (относительно начала координат)
~ gravity (0,10)

Отобразить сетку[8]
~ showGrid type 0
Сетка 100x50 пикселей.

~ showGrid type 1
    Сетка 50x25 пикселей.
~ showGrid type 2
    Сетка 20x10 пикселей.
~ showGrid 100x75, где
    100 - ширина ячеек в пикселях
    50 - высота ячеек в пикселях

    Спрятать сетку
~ hideGrid

    Запустить симуляцию
~ play

    Остановить симуляцию
~ stop

    "Промотать" симуляцию
~ step(100), где
    100 - число пропускаемых отрисовкой тиков

    Исполнить внутренний метод
~ execute nameMethod(params), где
    nameMethod - название метода
    params - сообщаемые параметры

    Посмотреть историю запросов
~ getHistory [9]

    Объединить команды в одном запросе
~ commandOne---commandTwo, где
    commandOne, commandTwo - команды консоли, могут также
    состоять из объединённых команд

Возможности плеера

~ Воспроизведение/пауза симуляции с заданным Δt;

~ Скачок вперёд на кратное Δt время;

~ "Замедление времени"[10].

Кнопки интерфейса

~ Кнопка Get point's id

После нажатия на кнопку, а затем на частицу выводит id последней в консоль.

~ Кнопка Move point[6]

Эквивалентна команде консоли movePoint без флага saveV.
После нажатия на кнопку, а затем на частицу закрепляет управление положением последней за курсором мыши.
Дальнейшие нажатия на свободные участки поля переносят эту частицу в точку нажатия, при этом её скорость считается нулевой.
Для прекращения управления следует вновь нажать исходную кнопку.

~ Клавиша клавиатуры ~ (также `, ё, Ё)

Эквивалентна командам консоли play/stop и кнопкам ▶ / | | плеера.

~ Кнопки плеера

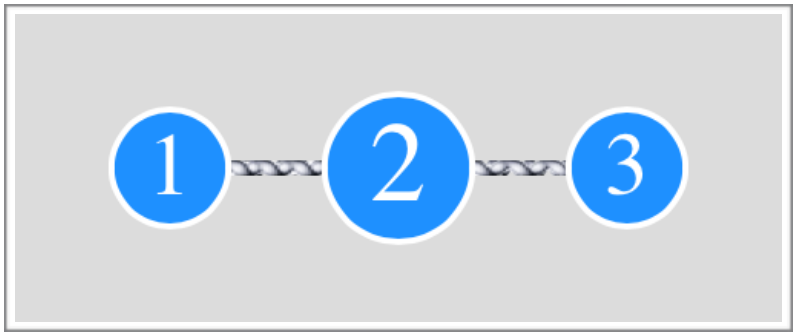
▶ / | | - воспроизведение/остановка симуляции (эквивалентна командам консоли play/stop).

▶ - переход на тик вперёд (эквивалентна команде консоли step(1)).

▶▶ - переход на 50 тиков вперёд (эквивалентна команде консоли step(50)).

▶| - переход на 100 тиков вперёд (эквивалентна команде консоли step(100)).

Пример системы



Смоделируем молекулу из трёх атомов (подобную молекуле углекислого газа CO_2).

Для создания такой системы необходимо последовательно выполнить следующие команды:

Создаём атомы кислорода, углерода и кислорода соответственно.

~ addPoint (100,100) (0,0) 40 5

~ addPoint (250,100) (0,0) 50 10

~ addPoint (400,100) (0,0) 40 5

Соединяем атомы пружинами для моделирования связей. Первая пружина обеспечивает устойчивость молекулы, не создавая внутреннего напряжения.

~ addSpring #0 #2 10

~ addSpring #0 #1 20

~ addSpring #1 #2 20

Дополнительно можем задать скорости у атомов кислорода для создания колебаний.

~ setVelocity #0 (1,0)

~ setVelocity #2 (-1,0)

Итоговый код (одна строка):

~ addPoint (100,100) (0,0) 40 5---addPoint (250,100) (0,0) 50 10---addPoint (400,100) (0,0) 40 5---addSpring #0 #2 10---addSpring #0 #1 20---addSpring #1 #2 20---setVelocity #0 (1,0)---setVelocity #2 (-1,0)

Выдержки кода решения

Суммарно код текущей версии (v2.0 b от 2 июня 2015 года) движка занимает
2 500 строк без учёта библиотеки.

Отличие от семейства версий 1.* в полностью переписанной логике проекта для оптимизации вычислений и соблюдения принципов OOP JS.

Файл **“point.js" - работаем с одной частицей**

```
1   function Point(mass, radius, coorXY, oldCoorXY, isExpToPot)
2   {
3       this.mass = mass;
4       this.invmass = 1 / mass;
5       this.radius = radius;
6       this.coor = coorXY;
7       this.oldCoor = oldCoorXY || coorXY;
8       this.id = getId();
9       this.type = 'point';
10      this.isExpToPot = isExpToPot || false;
11
12  // log
13      console.log('Create: '+this.toString());
14  }
15
16
17  Point.prototype.checkCoor = function(){
18      if (this.coor.isNaN())
19      {
20          console.error('Coor is NaN: ' + this);
21          player.stop();
22      }
23      else
24      {
25          if (this.oldCoor.isNaN())
26          {
27              console.warning('OldCoor is NaN: ' + this);
```

```
28              this.oldCoor = new Coor(this.coor);
29          }
30          else
31          {
32              return ;
33          }
34      }
35  }
36
37  Point.prototype.toString = function(type){
38      if (type === 'full')
39      {
40          return JSON.stringify(this, [
41                                      'type',
42                                      'id',
43                                      'coor',
44                                      'mass_',
45                                      'radius_',
46                                      'x',
47                                      'y',
48                                      'begin',
49                                      'end'
50                              ], 4)
51      }
52      else
53      {
54          return JSON.stringify(this, [
55                                      'type',
56                                      'id',
57                                      'coor',
58                                      'x',
59                                      'y',
60                                      'begin',
61                                      'end'
62                              ], 4)
63      }
```

```
64      }
65
66      Point.prototype.move = function(coorXY){
67              var delta = minusCoor(this.coor, this.oldCoor);
68          this.coor = coorXY;
69          this.oldCoor = minusCoor(coorXY, delta);
70      // log
71          console.log('Move: '+this.toString());
72      }
73
74      Point.prototype.moveEase = function(coorXY){
75          this.coor = coorXY;
76          this.oldCoor = coorXY;
77      // log
78          console.log('Move: '+this.toString());
79      }
```

~ `this.mass = mass;`

Сообщаем нашей частице массу, равную mass.

~ `this.invmass = 1 / mass;`

Вводим обратную массу, равную 1/mass. В дальнейшем работаем именно с обратным значением массы, так как это позволит вводить “бесконечно тяжёлые” частицы (значение invmass можно задать отдельно впоследствие).

~ `this.radius = radius;`

Задаём радиус, равный radius. Измеряется в пикселях.

~ `this.coor = coorXY;`

Задаём текущие координаты значениями coorXY.x и coorXY.y.

~ `this.oldCoor = oldCoorXY || coorXY;`

Если были заданы предыдущие координаты (oldCoor.x и oldCoor.y), то записываем их. Иначе записываем текущие координаты, тогда частица будет обладать нулевым вектором скорости

~ **this**.id = getId();

Генерируем уникальный номер для частицы.

~ **this**.type = 'point';

Указываем тип элемента как “point”, чтобы обработчик отличал частицу от других объектов.

~ **this**.isExpToPot = isExpToPot || **false**;

Сообщаем, должна ли участвовать в потенциальном парном взаимодействии. Да - если isExpToPot равняется true, нет - во всех прочих случаях.

~ console.log('Create: ‘+**this**.toString());

Выводим в консоль подробную информацию о созданной частице.

~ Point.**prototype**.checkCoor

Метод проверки значений координат положений частицы. Если текущее положение частицы не может быть определено (не верный запрос, статически неопределимая система и пр.), останавливаем симуляцию и выводим в консоль ошибку. Если не удаётся определить значения координат предыдущего положения (симуляция запущена пользователем после получения ошибки в текущих координатах частицы и пр.), выдаём предупреждение, используем координаты текущего положения также и как координаты предыдущего.

~ Point.**prototype**.toString

Метод, собирающий информацию о частице и выдающий её в удобном виде.

~ Point.**prototype**.move

Метод перемещения частицы в указанные координаты. Вектор скорости сохраняется, в консоль записывается информация о перемещении.

~ Point.**prototype**.moveEase

Другой метод перемещения частицы в указанные координаты. Вектор скорости зануляется, в консоль записывается информация о перемещении.

Использованная литература

~ T. Jakobsen. “Advanced Character Physics”, 2003;

~ Л. Ландау, Е. Лифшиц. “Теоретическая физика”, том первый, “Механика”, 1988;

~ D. Baraff. “Dynamic Simulation of Non-Penetrating Rigid Bodies”, 1992;

~ D. Stewart, J. Trinkle. “An Implicit Time-Stepping Scheme for Rigid Body Dynamics with Inelastic Collisions and Coulomb Friction”;

~ L. Verlet. “Computer experiments on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules”, 1967;

~ A. Witkin, D. Baraff. “Physically Based Modeling: Principles and Practice”, 1997.

Примечания

1.

Строго говоря, мы рассматриваем проекцию трёхмерной системы на двухмерное пространство. Так, на изображении в заголовке страницы демонстрируется следующий эксперимент: есть тело в форме параллелепипеда (набор частиц со связями: пружинами), на тело сверху падает стержень. Проекцией этого тела является его слой: треугольная решётка, закреплённая на концах. Проекция стержня - частица (на изображении отсутствует).

2.

Стержни рассчитываются на расстяжение/сжатие методом коррекции координат.

Действие пружин учитывается как действие сил упругости.

3. [3.0, 3.1, 3.2]

Без выделения жирным написаны необязательные параметры. При желании указать необязательный параметр все значения слева от него следует считать обязательными (во избежание путаницы при парсинге безразмерных величин в команде).

4.

Значения по умолчанию: вектор скорости нулевой, радиус равен 50 пикселям, масса равна 5 у. е.

5. [5.0, 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7]

Идентификационный номер элемента в системе. Генерируется последовательно, начиная с нуля, для стенок, стержней, пружин и частиц при их добавлении. Для частиц значение id можно найти нажатием сначала на кнопку "Get point's id", а затем на частицу: тогда Id отобразится в консоли.

6 [6.0, 6.1, 6.2, 6.3]

Важно! Визуально действие этой команды применится только при перерисовке кадра.

7.

Жёсткость пружины по умолчанию равна 5 у. е.

8.

Важно! Отображается максимум одна таблица за раз.

9.

Запросы выводятся в всплывающем окне одной цельной командой, для воспроизведения цепочки запросов достаточно ввести эту команду в консоль.

Важно! Учитываются только запросы, успешно введённые в консоль. Воздействия на систему при помощи кнопки "Move point" будут потеряны.

10.

При малой производительности клиента уменьшаем число отрисовок в единицу времени для сохранения гладкости анимации. Управлятся через консоль.