САНКТ-ПЕТЕРБУРГСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра системного программирования
Группа 20.Б11-мм

Сульдин Вячеслав Романович

Влияние момента центра инерции на движение броуновских частиц

Отчет по учебной практике
в форме «Эксперимент»

Научный руководитель:
профессор кафедры ПА, д.ф.-м.н. Прозорова Э. В.

Санкт-Петербург
2022

Содержание

Введение

В настоящее время, благодаря стремительному развитию вычислительной техники, приблизиться к решению задач изучения свойств веществ позволяют методы компьютерного моделирования. Во многих случаях они оказываются единственным способом получения детальных количественных сведений о поведении молекулярных систем, как известных в природе, так и еще планируемыхк созданию. Сопоставляя результат вычислений с опытными данными, можно выявить наиболее важные факторы и закономерности, отвечающие за те или иные свойства реальных молекул.

С другой стороны, компьютерное моделирование часто выступает в качестве связующего звена между теорией и физическим экспериментом. Во многих случаях оно является мощным средством повышения информативности самих методов исследований молекул, так как позволяет целенаправленно выполнять эксперимент, ускоряя его проведение.

Одним из «прогнозирующих» компьютерных методов является метод молекулярной динамики (МД). Молекулы рассматриваются как система взаимодействующих классических частиц. Используются различные модели взаимодействия, методы расчёта движения. Распространенную модель движения броуновских частиц описывает уравнение Ланжевена. Она включает в качестве дополнительного фактора нерегулярную (статистическую) силу.

Однако есть исследования, показывающие, как движение молекул, даже в условиях равновесия, приводит к постоянному смещению центра инерции, что создает момент и приводит к появлению дополнительной силы. Нам бы хотелось проверить и исследовать вычислительным методом решение задачи о броуновском движении, используя данную модель движения. Её главное отличие заключается в том, что сила детерменирована и обусловлена изменением момента центра инерции частиц.

1 Цели и задачи

Целью данной работы является проверка корректности влияния момента центра инерции на движение броуновских частиц. Для её выполнения были поставлены следующие задачи:

1. Создание классической системы движения молекул:
    - Распределение Максвелла-Больцмана,
    - Потенциал Леннарда-Джонса,
    - Периодические граничные условия;
2. Визуализация происходящих процессов:
    - Положение частиц в пространстве,
    - Графики исследуемых величин;
3. Расчёт момента инерции и исследование его влияния на систему.

2 Обзор предметной области

В этом разделе мы детальнее рассмотрим используемые при создании проекта области физической науки. Почти все описанные ниже формулы или их аналоги появятся в реализации.

Используемые обозначения

- a - Ускорение
- x - Координаты
- v - Скорость
- Δt - Временной промежуток
- F - Сила
- k - Постоянная Больцмана
- T - Температура, K
- U - Потенциал
- r - Расстояние между молекулами
- ε - Глубина потенциальной ямы
- σ - Длина связи
- a - Центр инерции
- M_c - Момент инерции

2.1 Классическая механика

При моделировании молекулы полагаются твёрдыми сферами. Значит, они подчиняются уравненяем движения. Задача заключается в определении положения и скорости каждой частицы, и расчёте по данной траектории их новых координат. На каждом временном промежутке, который был получен после дискретизации, мы рассматриваем движение как равноускоренное. Тогда справедливы следующие формулы:

Второй закон Ньютона, позволяющий нам определять ускорение частицы

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

Зная ускорение, скорость и начальные положения молекулы, мы можем предсказать её положение в пространстве и скорость через некоторый момент времени момент

$$\vec{x} = \vec{x_0} + \vec{v_0}t + \frac{\vec{a}}{2}t^2 \tag{2}$$

$$\vec{u} = \vec{u_0} + \vec{a}t \tag{3}$$

Тем не менее, полезными оказываются другие алгоритмы вычисления движения. Уменьшая погрешность вычислений, они в большей степени обеспечивают выполнение законов сохранения. Далее будем верхними индексами обозначать временной шаг, или номер конфигурации, и опустим векторную запись. Рассмотрим наиболее известные

1. Алгоритм Верле в скоростной форме

$$x^{i+1} = x^i + v^i\Delta t + a^i\frac{\Delta t^2}{2} \tag{4}$$

$$a^i = \frac{F^i}{m} \tag{5}$$

$$v^{i+1} = v^i + \frac{(a^i + a^{i+1})\Delta t}{2} \tag{6}$$

2. Алгоритм с полушагом

$$x^{i+1} = x^i + v^i\Delta t + a^i\frac{\Delta t^2}{2} \tag{7}$$

$$a^i = \frac{F^i}{m} \tag{8}$$

$$v^{\frac{i+1}{2}} = v^i + \frac{a^i\Delta t}{2} \tag{9}$$

$$v^{i+1} = v^{\frac{i+1}{2}} + \frac{a^{i+1}\Delta t}{2} \tag{10}$$

Можно заметить, что основное их отличие - метод нахождения скорости. Действительно, наше Δt хоть и мало, но не сколь угодно мало. Поэтому и к равенству в формулах мы едва ли можем приблизиться. Алгоритмы пытаются свести погрешность к минимуму. В будущем ключевым фактором при выборе алгоритма будет его влияние на способность сохранять стабильность системы максимально длительное количество итераций.

2.2 Распределение Максвелла-Больцмана

Функция распределения Максвелла показывает, какова вероятность того, что скорость данной молекулы имеет значение, заключенное в единичном интервале скоростей, включающем данную скорость u, или каково относительное число молекул, скорости которых лежат в этом интервале.

Функция распределения значений проекции скорости v_x

$$\Psi(v_x) = \left(\frac{m}{2\pi KT}\right)^{1/2} exp\left(\frac{m{v_x}^2}{2kT}\right) \tag{11}$$

Функцию распределения Максвелла мы будем использовать, чтобы задать корректные начальные скорости частиц. Такое состояние называется равновесным, и исследуем мы систему, обладающую данным свойством.

Рис. 1: График функции Максвелла-Больцмана при различных T

2.3 Потенциал Леннарда-Джонса

Самая распространенная модель бинарного взаимодействия молекул – потенциал Леннарда-Джонса.

$$U(r) = \varepsilon \left[\left(\frac{\sigma}{r} \right)^{12} - 2 \left(\frac{\sigma}{r} \right)^{6} \right] \tag{12}$$

При больших r молекулы притягиваются, что соответствует члену $(\frac{\sigma}{r})^6$. На малых же расстояниях молекулы начинают сильно отталкиваться $(\frac{\sigma}{r})^{12}$.

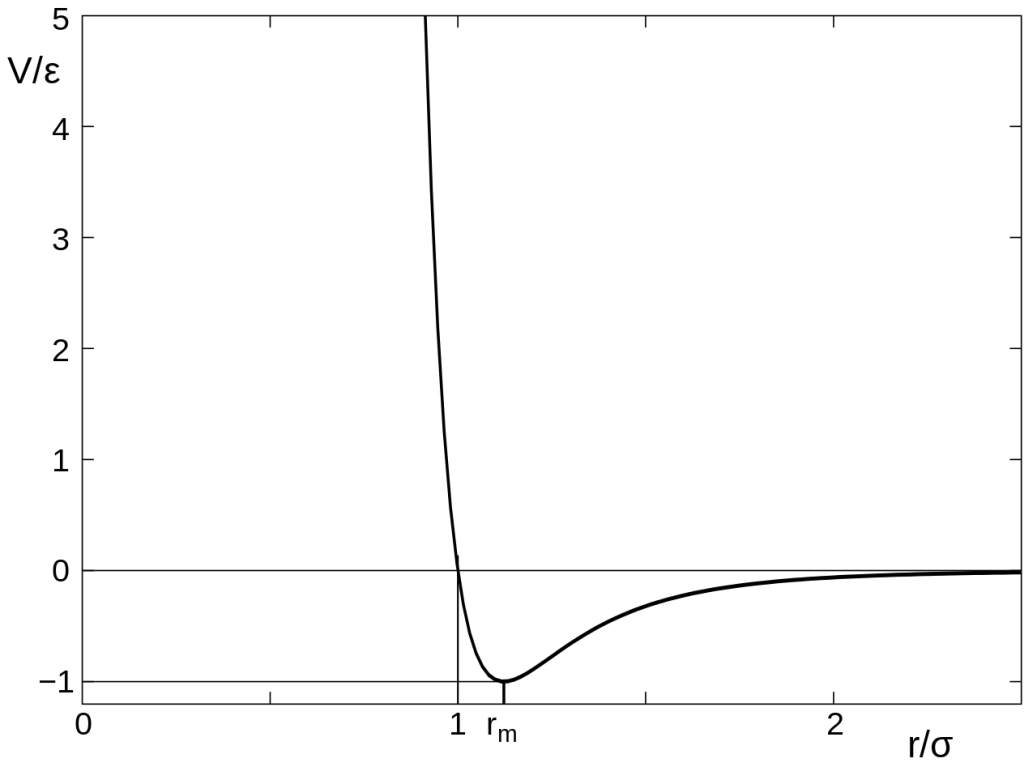

Рис. 2: График потенциала Леннарда-Джонса. Обратите внимание на масштаб и экстремумы.

Нас скорее будет интересовать сила, обусловленная этим потенциалом. Она есть ничто иное, как производная, взятая с другим знаком.

$$F(r) = -\frac{dU}{dr} \tag{13}$$

И, вычисляя производную

$$F(r) = \frac{12\varepsilon}{\sigma} \left[\left(\frac{\sigma}{r} \right)^{13} - \left(\frac{\sigma}{r} \right)^{7} \right] \tag{14}$$

Стоит отметить, что равенство достигается при $\mathrm{dr} \to 0$, чего невозможно обеспечить при компьютерном моделировании, но чем меньше dr, тем точнее будут результаты.

2.4 Момент центра инерции

Центр инерции - геометрическая точка, положение которой определяется распределением массы в теле, а перемещение характеризует движение тела или механической системы как целого. Найти её, имея конечное число частиц, довольно просто. Особенно, если все они одной массы.

$$\vec{a} = \frac{\sum_{i=1}^{n} \vec{x_i} m_i}{\sum_{i=1}^{n} m_i} \tag{15}$$

Момент центра инерции можно найти как векторное произведение плеча (вектора от центра до точки приложения) на силу

$$M_c = (x - a) \times F \tag{16}$$

После изменения центра инерции, вызванного изменением положений молекул, появиться новая сила. Эта сила - и есть влияние момента центра инерции, которое является главной исследуемой величиной в данной работе. Рассчитать её можно как производную по направлению, то есть градиент.

$$F_c = \bigtriangledown M_c \tag{17}$$

Предполагается, что суммарная сила, действующая на частицу, складывается из основных сил взаимодействия (В нашем случае потенциал Л–Дж), а также силой, вызванной изменением центра инерции.

3 Ход работы

В данном разделе я опишу, как я реализовывал изученный физико-математический аппарат программно, с какими проблемами столкнулся, и к каким решениям пришёл.

3.1 Выбор инструментов разработки

В качестве языка разработки был выбран C++, так как при должном желании он выделяется скоростью и точностью, что особенно важно при большом количестве математических вычислений.

Для наглядности, отладки и построения графиков была выбрана SFML - простая и быстрая библиотека для создания оконных приложений, которая не будет отвлекать от главной цели проекта, но поможет сформировать общую картину.

3.2 Подготовка

Для начала хотелось определить единицы, с которыми мы будем активно работать. Своё отражение это желание нашло в следующих модулях.

3.2.1 Константы

Для автоматизированной работы с программой я определил нужные константы в соответствующем файле Constants.h классу. Основной задачей было связать величины, что сделало бы постановку эксперимента более простой. Основным выбрано количество молекул на стороне куба системы. Число всех молекул есть куб количества на стороне. Определив дистанцию между молекулами, можно задать габариты всей системы. Остальные константы – физические характеристики аргона и значение растяжения, помогающие при отражении системы на визуализацию.

3.2.2 Класс Vector

В работе мы производим много вычислений нескалярных, то есть векторных величин. Для сокращения количества кода и более привычных записей я посчитал удобным добавить инструмент работы с векторами. Поэтому были добавлены классические операции: сложение, вычитание, умножение на константу и векторное, унарный минус, нормализация и нахождение длины вектора. Их реализация тривиальна

3.2.3 Класс Molecule

Заметим, что при нахождении многих величин нам требуются два значения - предыдущие и текущие. Чтобы упростить работу с ними, вве-

дем класс помощник Delta, на деле являющийся структурой с двумя векторами, логически соответвующим этим значениям.

Самым важным объектом исследования является молекула. Её обязательными характеристиками являются скорость, действующая на неё сила, положения в пространстве, масса. Также мы хотим, зная эту информацию, расчитывать и «двигать» молекулу. Для этого нам понадобятся алгоритмы из предыдущей главы, их реализация проста.

Забегая вперёд. Функция periodic() проверяет, не вылетела ли молекула из системы, в противном случае обрабатывает эту ситуацию. Важно после каждого вызова функции, обновляющей позиции молекулы, вызывать и проверку periodic(). Иначе частицы разлетятся. Её реализации выделен свой раздел 3.5.

Уже такой небольшой аппарат позволяет, добавив силы взаимодействия, моделировать движение молекул. Но пока мы имеем информацию только в числах, поэтому на этом этапе я задумался о визуальном представлении системы.

3.2.4 Визуализация. SFML

Данный раздел познакомит с самым базовым арсеналом библиотеки SFML на примере решения задачи визуализации процесса движения молекул. Далее я описал свои решения следующих задач;

- Изображение трёх-мерного пространства на дву-мерном экране.
- Организация работы с несколькими окнами
- Автоматизированое построение графиков в реальном времени

Подключение библиотеки и изучение базовых конструкций я возлагаю на производителя.
https://www.sfml-dev.org/tutorials/2.5/start-linux.php

Этого нам хватит, чтобы выполнить поставленные задачи. Запуская нашу симуляцию, нам бы хотелось видеть сразу несколько окон, (сами молекулы, график a, график b...). Но нельзя забывать об удобстве. Чтобы создать и поддерживать новое окно, нужно проделывать однотипные действия (create, isOpen, refresh, display). Занести их все в массив - тоже не вариант: с ростом их количества растёт вероятность ошибки при запросе из «Чёрного ящика». Чтобы автоматизировать этот процесс, было принято решение создать класс, содержащий в себе все окна, и обеспечивающий удобную работу с ними.

Я выбрал путь через X-Макросы. Они не сложны для понимания, но очень универсальны в использовании, и я нашел в них выход. Вот несколько полезных материалов (). Тем временем разберём этот класс.

1: Visualization.h

```
 // Список нужных окон
#define LIST_OF_WINDOW \
  X(main)              \
  X(kinetic)           \
  X(force)

class App {
 public:
  App();

  void setup(int argc, char *argv[]);
  /* Проще показать, на что заменятся,
       строки 25-27:
      sf::RenderWindow main;
      sf::RenderWindow kinetic;
      sf::RenderWindow force;
       Для каждого из списка   мы создали по
       полю, сделав работу с окнами более строгой. */
#define X(name) sf::RenderWindow name;
  LIST_OF_WINDOW
#undef X
  // Добавление окна
  void add_window(sf::RenderWindow *window, sf::String title)
      ;
  // Проверка на желание закрыть окно
  void is_close();

  void display();
  // Сравнение с аргументом и добавление окна
  void add_if(const char *name, char *cmd, sf::RenderWindow *
      window);
};
```

Осталось еще организовать своеобразный дескриптор комманд, распознающий нужный список. Рассмотрим функции по отдельности. Функция add_window инициализирует окно, add_if добавляет в случае совпадения параметров запуска с окном из списка.

2: Visualization.cpp

```
/* Создавая объект, инициализируем требуемые окна
   на основе нашего списка. Для каждого слова в
   аргументах мы запустим add_if.              */
App::App(int argc, char *argv[])
{
    for (int i = 0; i < argc; i++)
    {
#define X(name) add_if(#name, argv[i], &name);
        LIST_OF_WINDOW
#undef X
    }
}
```

3: Visualization.cpp

```
// Обновляем все окна
void App::display()
{
#define X(name) name.display();
    LIST_OF_WINDOW
#undef X
}
```

Подводя итог, чтобы добавить новое окно нужно:

- добавить его в LIST_OF_WINDOW
- упомянуть при запуске приложения в аргументах

Всю остальную работу с окном берёт на себя класс.

Построим графики, используя новоприбывающие значения. Основная из преследуемых целей: сравнивать эти графики. Это говорит о необходимости размещать несколько на одном окне. Поэтому будем разделять их от окон. У класса Graph есть свой цвет, множество его значений, граница, и функция обновления графика по новому значению.

Наибольший интерес представляет функция update_graph. Рисуем на окне X на Y. За ось ординат принято Y. Рисуем по пикселям, поэтому значений столько, сколько X. Сформируем массив значений, добавляя новое в конец, выбрасываем старейшее. Отмасштабируем значения относительно размеров окна при помощи максимального значения, и изобразим.

Иногда такой график будет растягиваться по вертикали, так он будет всегда в поле зрения для анализа.

Наконец, нарисуем молекулы. Конечно, нам хочется изобразить трёхмерность среды. Есть много способов. Например использовать вместо SFML OpenGL. Но, на мой взгляд, это слишком мощный инструмент, который может съесть львиную долю производительности, делая акцент на трехмерную графику. Нам же хватит двумерной иллюзии. Важен вызов функции отрисовки каждой молекулы.

4: Main.cpp

```
draw(&visualization.main,
     // Размер отрисовываемой фигуры.
     10 + ((X * ARGON_RADIUS / (2 * MOL_SIDE * SIGMA)) /
        SIDE_OF_SYSTEM) * molecules[i].coordinates.cur.z,
     // Цвет
     sf::Color::Cyan,
     // X и Y берём исходные.
     molecules[i].coordinates.cur.x,
     molecules[i].coordinates.cur.y,
     // Поскольку передаём значения не в пикселях, нужно отма
        сштабировать.
     SCALE);
```

Подробнее про второй аргумент. Достигать объёмности будем за счёт изменения размера. Чем ближе молекула - тем больше, и наоборот. За это отвечает z координата. Но хотим, чтобы размер, обращался в 0, когда $z = 0$ (далеко), и пропорционально увеличивался до максимального значения при $z \to$ SIDE_OF_SYSTEM. Максимальное значение есть

$$\frac{X * ARGON\_RADIUS}{2 * MOL\_SIDE * SIGMA}$$

Тут скрыта еще одна идея - добиться реальных пропорций (выполнялось

$$\frac{SIDE\_OF\_SYSTEM}{X} = \frac{ARGON\_RADIUS}{2ndArg}$$

Добавим еще draw, но уже без 10 во втором аргументе и с другим цветом. Получим меньшие круги поверх больших, это позволит различать молекулы при наложении.

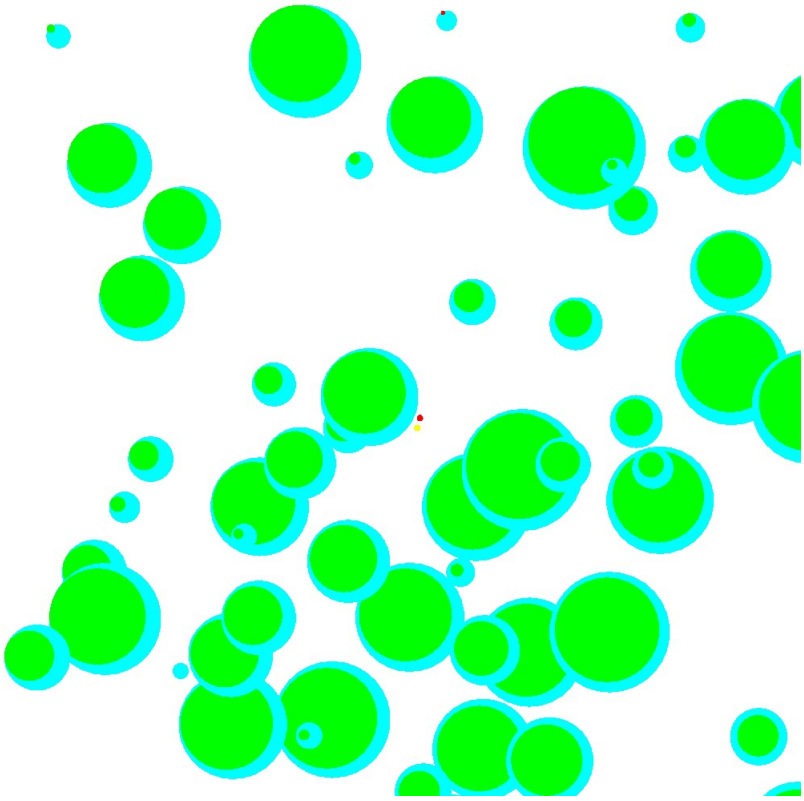

Рис. 3: Полученная визуализация молекул

Остается одна проблема - поскольку отрисовка происходит в порядке положения в массиве - далекие молекулы могут казаться ближе, отрисовываясь первее и, соответсвенно, «на» больших. Можно добавить сортировку по z, чтобы это исправить, но, на мой взгяд, это избыточно.

3.3 Начальное распределение молекул

Рассмотрим начальный момент времени. Для кооректности необходимо задать те условия, в которых могла оказаться исследуемая система. Взяв нереальное расположение молекул мы не получим полезных результатов.

3.3.1 Координаты

Про координаты было упомянуто в разделе про константы. Выбранно расположение в виде куба, поскольку мы рассматриваем лишь часть - ячейку крупной системы. Это можно сравнить с приближением площади маленькими квадратами. А еще это удобно упаковать в цикл. Чтобы задать начальное расположение, необходимо заполнить трехмерную таблицу, используя индексы как положения молекул в пространстве, то есть домножив на DIST из Constants.

3.3.2 Скорости

Подробнее про начальные скорости. Функция get_maxwell_vector() по запросу выдает вектор скорости. В нашем случае используется функция распределения Максвела. См. 2.2. Для реализации нужно построить гистограмму и создать группы скоростей. Чем больше в группе экземпляров, тем чаще встречается молекула. Устремляя количество групп к бесконечности, получаем все более точную функцию распределения. Функция maxwell() в точности повторяет функцию из раздела. Следующая функция строит распределение.

5: Physics.cpp

```
double get_maxwell_dist() {
  static unsigned int seed = time(NULL);
  double sum = 0;
  std::vector<double> vel;
  /*  Для каждого значения скорости, которому соответсвует
      индекс массива, вычисляем вероятность, и записываем
      в этот индекс. Максимальное значение скорости 1000 */
  for (int i = 0; i < 1000; i++) {
    double prob = maxwell(i);
    vel.push_back(prob);
    sum += prob;
  }
  //  Берём "случайный" процент
  double b = rand_r(&seed) % 100;
  int i = 0;
  double cur = 0;
  /*  Ищем, чему этот процент соответствует. Чем больше
      было вероятность i-ого значения скорости, тем с
      большей вероятностью мы перешагнём и остановимся
      на нём. Либо массив кончится.                    */
```

```
21    for (i = 0; i < static_cast<int>(vel.size()) && b > cur *
          100; i++) {
22      cur += vel[i] / sum;
23    }
24    return (rand_r(&seed) % 2 == 0 ? -1 : 1) * i;
25  }
```

Функция get_maxwell_vector() возвращает вектор из трех значений get_maxwell_dist(). Взятая с такими скоростями система удолетворяет распределению скоростей максвела по проекциям.

3.4 Итерация симуляции

Имея начальные значения, мы можем перейти к последующей конфигурации. Продолжать этот процесс можно до тех пор, пока накопленная погрешность не сделает вычисления бессмысленными. Не взаимодействующие молекулы ведут себя довольно скучно - двигаются по заданой начальной скоростью траектории. В действительности между молекулами возникают силы. Или, в нашем, случае вылететь за пределы системы. Эти ситуации нужно смоделировать.

3.4.1 Расчёт сил

Мы будем использовать наиболее расрпостраненный метод нахождения сил взаимодействия молекул – потенциал Леннарда-Джонса 2.3. Константы для неё вычислялись экспериментальным методом, и уникальны для каждого вида молекул. Переписываем формулу в функцию lennard_jones.

Несколько больше интересно её применение. Наивно - необходимо для каждой молекулы найти её пару, посчитать расстояние между ними, прибавить силу, забыть её, попасть на вторую молекулу, найдя ей в пару первую - снова посчитать расстояние и силу. А рассчёт расстояния выполняется не за $O(1)$, поэтому это может сыграть свою роль. Куда более успешным решением будет рассмотреть все пары, без повторов. Их уже не n^2, а $\binom{n}{2}$, что всяко лучше. Вспоминая сортировку пузырьком:

6: Physics.cpp

```
1  for (int i = 0; i < molecules.size(); i++)
2      {
3          Vector force_sum = Vector(0, 0, 0);
4        // начинаем не с 0, а с i+1. Так рассмотрим все пары и
             только их
5          for (int k = i + 1; k < molecules.size(); k++)
6          {
7                // находим численное значение расстояние, и напр
                     авление взамодействия
```

```
8               pair<double, Vector> dist_dir =
                    calc_periodic_dist(molecules[i], molecules[k])
                    ;
9
10              // Функция возвращает dist = 0, если молекулы сли
                    шком далеко друг от друга.
11              if (dist_dir.first != 0)
12              {
13                   // Высчитываем силу как направление *
14                   Vector force = dist_dir.second *
                         lennard_jones(dist_dir.first);
15
16                   // Собираем для i-ой
17                   force_sum += force;
18                   // Для каждой k-ой собираем по отдельности
19                   molecules[k].force.cur += -force;
20              }
21          }
22          // То, что собрали с последующих прибавляем.
23          molecules[i].force.cur += force_sum;
24      }
```

Таким образом для i-ой молекулы мы $(size() - i)$ раз прибавляем силы с другими (23 строчка) и еще i сил она получает, потому что до этого ровно i молекул отдали ей свою. Вот, например, для 4-ех молекул. A$\rightarrow$ B означает: сила B $+=$ сила от молекулы A. Сверху вниз - по i. Стрелки снизу означают 23 строку кода. Итого каждая молекула «получила» по 3 силы, и «отдала» равно 3.

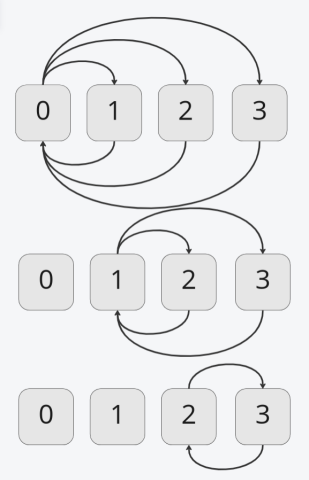

Рис. 4: Расчет сил для 4-ёх молекул

3.4.2 Получение новых положений молекул

Заполнив массив свежими значениями сил, мы готовы совершить шаг. Выполняем его одновременно для всех. Итого основной алгоритм выглядит следующим образом

1. Найти силу
2. Найти скорость
3. Сдвинуть молекулы

Далее осталось лишь продолжать данные вычисления. Причиной остановки может служить как закрытие главного окна, так и наступление определенного момента времени. В таком случае понадобиться счётчик. Я использую терминацию main.

3.5 Периодические граничные условия

Наша система ограничена. Поэтому ситуация, при которой молекула достигнет границу, неизбежна. Известны две модели разрешения задачи границ.

- Жесткие граничные условия
  Молекулы просто отскакивают от стенок. Угол отражения равен углу падения. Скорость не меняется по модулю. Простая модель.
- Периодические граничные условия

  Наша система окружена своими копиями. Молекула, вылетая из неё, заменяется своей копией, влетающей с другой. Двумерный случай изображен ниже.

Я выбрал второую модель, поскольку она, в некотором смысле, позволяет смоделировать бесконечно много частиц.

Обработать случай вылета за границу системы довольно просто. Необходимо зафиксировать значение координат молекулы либо большим, чем сторона системы, либо меньшим, чем 0. В обоих случаях перемещаем молекулу изменением этой координаты на значение длины стороны системы.

Теперь молекулы в бесконечном отеле. Но есть некоторое противоречие. Предположим, мы рассматриваем две молекулы. Они находятся на расстоянии больше действия сил, но так, что фантом одной из них близок ко второй. Логично было бы считать, что они взаимодействуют. Причем в противоположном направлении, нежели оригиналы. Я посчитал нужным учитывать данный случай. То есть, для того, чтобы найти

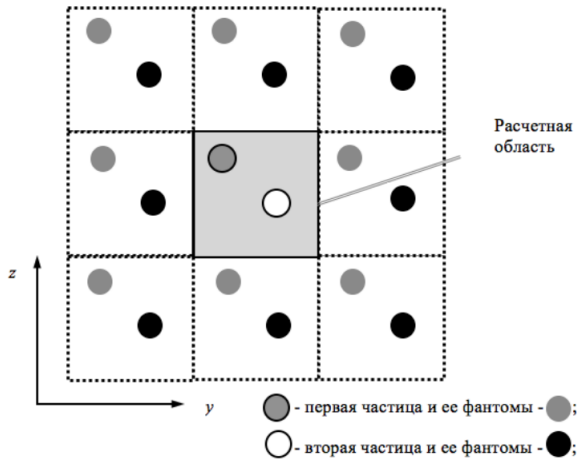

Рис. 5: Иллюстрация периодических граничных условий

расстояние между частицами, нам необходимо еще и высчитать расстояние до их копий.

Довольно просто показать, что, при данном выборе констант, если мы нашли достаточно малое расстояние между молекулой и какой-то копией второй, то дальше мы не найдем не только более влиятельного фантома, но и вовсе в радиусе действия силы. (Все копии находятся на равном расстоянии друг от друга, равном стороне системы. Поэтому уже в случае 8 молекул (0 и 1 очевидно) мы не сможем предъявить конфигурацию, где у молекулы в радиусе действия силы лежит два и более фантомов другой частицы). Реализовать можно следующим образом.

7: Physics.cpp

```
1  //Для удобства возьмём точки молекул
2  Vector target = tarMol.coordinates.cur;
3  Vector to_copy = copMol.coordinates.cur;
4      /* Заведем пустышку, которую будем изменять, получая из ц
           ентральной
5         молекулы её копии (включая центральную) */
6  Vector dummy = to_copy;
7  for (int n = -1; n <= 1; n++) {
8      for (int m = -1; m <= 1; m++) {
9        for (int k = -1; k <= 1; k++) {
10         // Смещаемся, получая из исходной копию в другой
11         dummy.x += m * SIDE_OF_SYSTEM;
12         dummy.y += n * SIDE_OF_SYSTEM;
13         dummy.z += k * SIDE_OF_SYSTEM;
14         // Находим расстояние между ними в привычном
15         double d = distance(dummy, target);
16
17         if (d < FORCE_RADIUS) {
18           return std::pair<double, Vector>(
19               d,
20               Vector(target.x - dummy.x,
```

```
21                      target.y - dummy.y,
22                      target.z - dummy.z).normalize());
23          }
24
25          dummy = to_copy;
26        }
27      }
28    }
29    return std::pair<double, Vector>(0, Vector(0, 0, 0));
```

Это помогает нам не только детектировать сложные ситуации, но правильно расчитывать силу, считая в таких случаях взаимодействия с фантомами, а не с оригиналами, что придавало бы силе неправильное направление. Вот пример такой ситуации. Вокруг молекул изображено поле действия силы. Стрелой показано её направление.

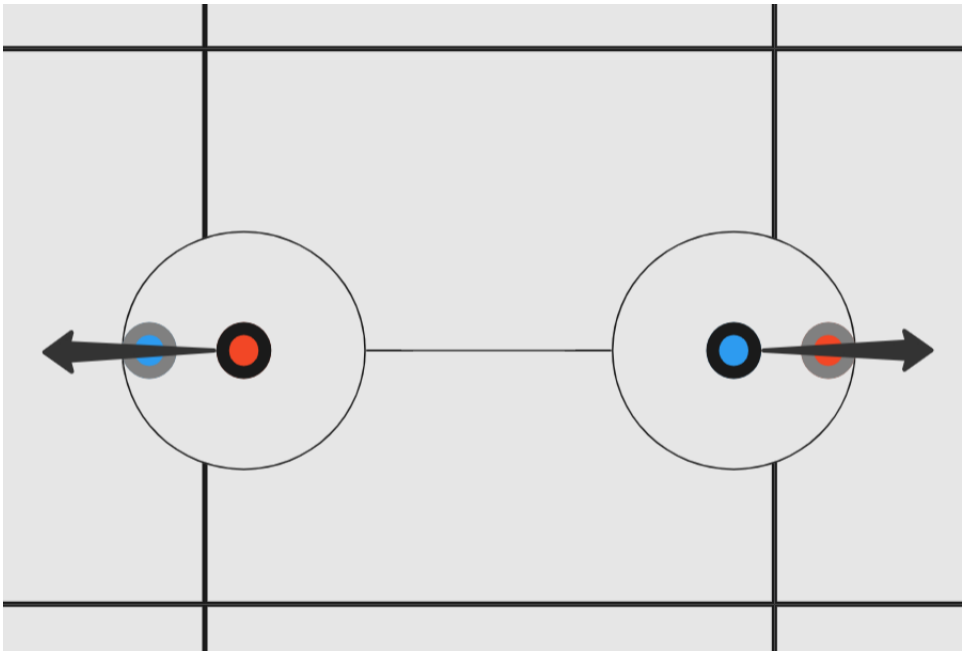

Рис. 6: Взаимодействие на границе

3.6 Проверка стабильности системы

Проверим, насколько хорошо выполняются законы сохранения. Для этого добавим график кинетической энергии. Для начала возьмём две молекулы, а скорости зададим только по одной координате - навстречу друг другу.

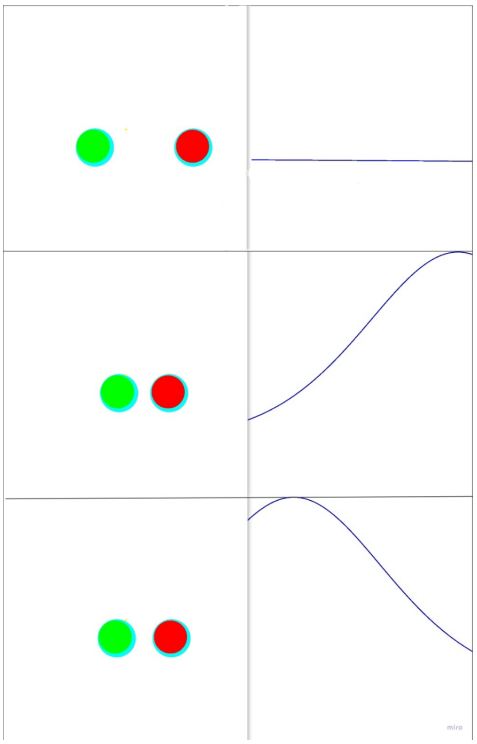

Рис. 7: Две молекулы $\Delta t = 10^{15}$

Проанализируем его. Очень хорошо видно, что побуждает изменение кинетической энергии. Это приближения молекул друг к другу, и, как следствие, влияние на них сил, ускоряющих их.

- Сначала они ускоряются - график под наклоном вниз (Влияние мало, но присутствует)
- Потом они начинают отталкиваться, вплоть до скоростей, равных 0 (Вершина графика)
- График возвращается на прежние значения до столкновения.

Полезным может оказаться рассмотреть более низкую точность, увеличив Δt.

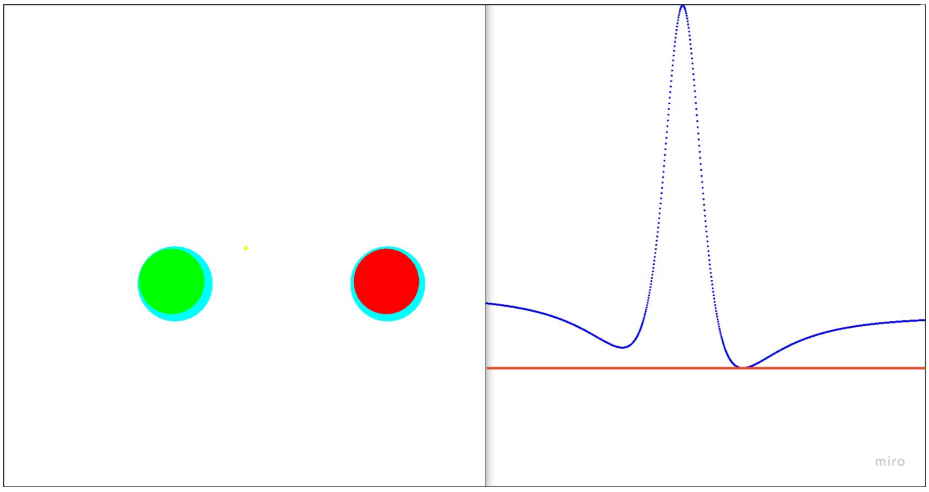

Рис. 8: Две молекулы $\Delta t = 10^{13}$

Можно видеть, что с каждым взаимодействием кинетическая энергия на очень малую величину увеличивается. При множественном взаимодействии этот эффект будет накапливаться и результаты эксперимента будут недостоверными. Поэтому с целью свести расхождение к минимуму шаг по времени берется максимально малый.

Конечно, добавив молекул, нам будет еще сложнее проследить за каждым столкновением, но важно то, что кинетическая энергия после колебаний возвращается на прежний уровень.

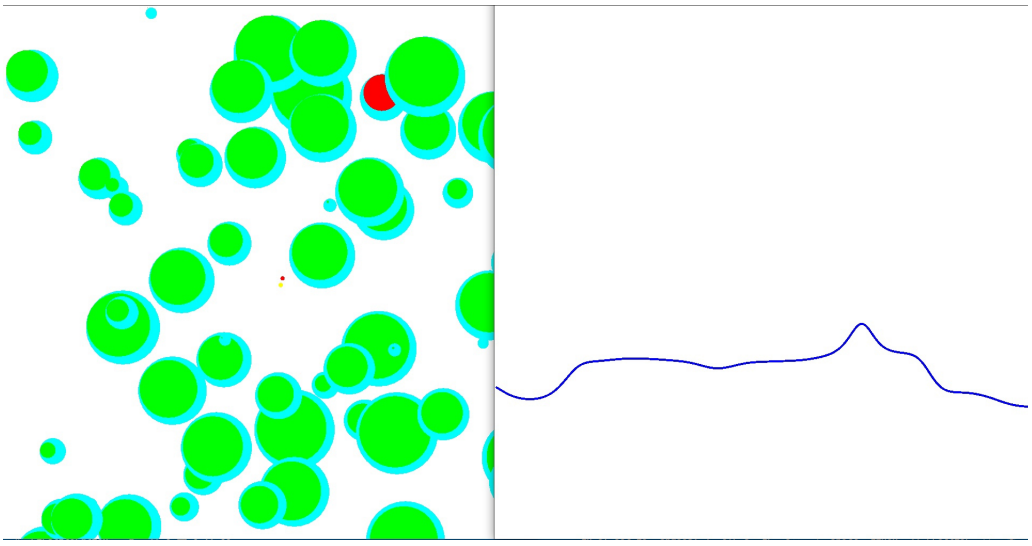

Рис. 9: 100 молекул $\Delta t = 10^{15}$

3.7 Момент центра инерции

Сформировав базу, и убедившись в её корректности, проверив важные законы, можно приступить к проверке новой модели. А именно рассчитать центр инерции, и найти силу, которую вызывает изменение его момента.

Добавим силу центра инерции как новое поле в класс Molecule. Нам понадобиться и центр инерции - характеристика скорее системы, чем отдельных молекул. И необходима простая функция расчёта этого центра - calc_iner_center, она похожа на нахождение среднего арифметического. Далее - функция расчёта силы.

8: Physics.cpp

```
for (int i = 0; i < molecules->size(); i++) {
    Delta dist
      = Delta((*molecules)[i].coordinates.cur - iner.cur,
              (*molecules)[i].coordinates.prev - iner.prev);

    Vector dM = (dist.cur | (*molecules)[i].force.cur) -
                (dist.prev | (*molecules)[i].force.prev);

    Delta coor = (*molecules)[i].coordinates;

    double dr = (coor.cur - coor.prev).length();

    (*molecules)[i].iner_force.cur = dM / dr;
}
```

Нам будет интересно рассмотреть момент приближения двух частиц, измерив действубщие силы только на одну. В остальнх случаях сила не меняется, поэтому и момент центра инерции меняется незначительно, вызывая малые значения силы.

Построим график полученных значений. Сразу Изобразим рядом с графиком основных сил.

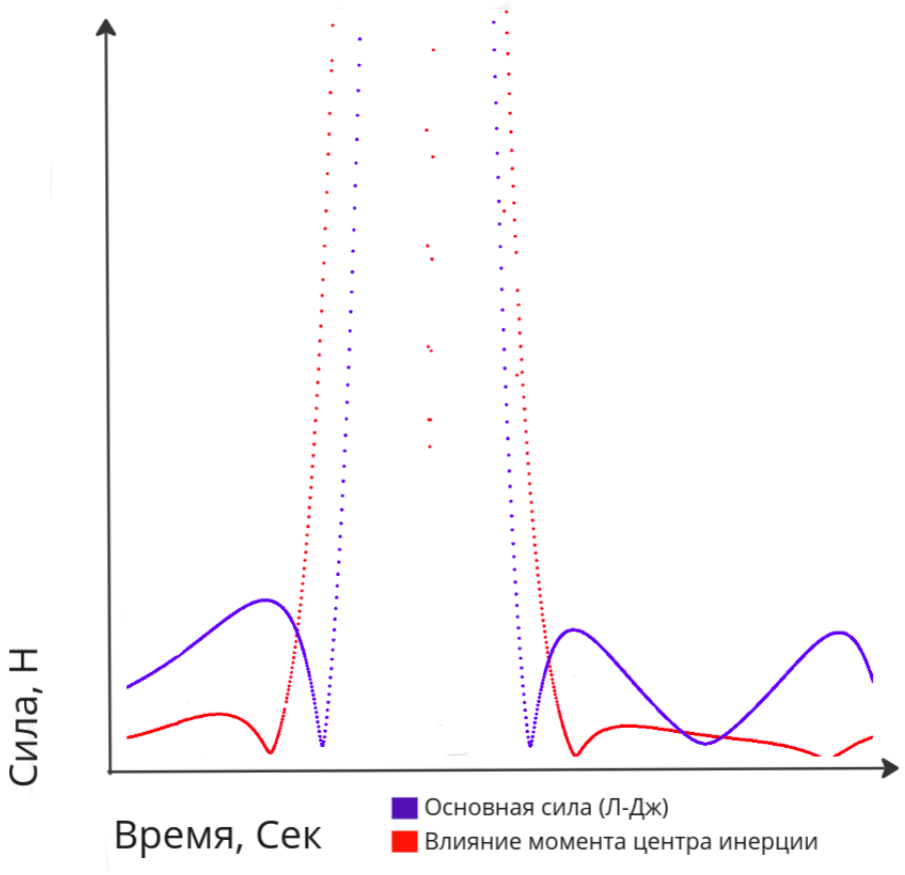

Рис. 10: Графики сил

Для упрощения сравнения все значения были приведены к одному порядку, поскольку значения силы влияния момента центра инерции превосходили основные силы. Важно заметить общее поведение графиков. На графике подтверждается природа сил инерции - в случае экстремумов синего графика у красного нули. И при наискорейшем росте основной силы возникают большие значения силы инерции.

Впоследствии мы многократно проверяли и формулу, и сам алгоритм нахождения влияния на различных тестовых данных, причём вычисление результатов совпадали. Это говорит о необходимости еще более детального изучения как теоретической составляющей, так и практического применения в методах молекулярной динамики.

4 Заключение

В результате работы над учебной практикой в течение осеннего семестра были выполнены следующие задачи:

1. Смоделировано движение броуновских частиц,
2. Добавлена визуальная составлящая проекта:
    - Движение частиц в системе,
    - График кинетической энергии,
    - График действующих сил;
3. Расчитано и исследовано влияние центра инерции.

Список литературы

[1] Гордеев Исай. Моделирование частиц в потенциале Леннарда-Джонса. https://thesaurus.rusnano.com/wiki/article1565

[2] Распределение Максвелла по проекциям скорости. https://physics.spbstu.ru/userfiles/files/molec2-03.pdf

[3] Звонарев С. В., Кортов В. С., Штанг Т. В. Моделирование структуры и свойств наносистем

[4] Майер Р. В. Компьютерное моделирование. http://komp-model.narod.ru/Komp-mod9.pdf

[5] Рудяк В. Я., Краснолуцкий С.Л. Диффузия наночастиц в разреженном газе.

[6] Физические методы исследования атомных и наномасштабных объектов для физиков

[7] Evelina Prozorova. Mechanism of Formation for Fluctuation Phenomena.

[8] Evelina Prozorova. Influence of the Moment in Mathematical Models for Open Systems. WSEAS TRANSACTIONS on APPLIED and THEORETICAL MECHANICS DOI: 10.37394/232011.2021.16.28

[9] Evelina Prozorova. The Role of the Angular Momentum in Shaping Collective Effects

[10] Evelina Prozorova. The Law of Conservation of Momentum and the Contribution of No Potential Forces to the Equations for Continuum Mechanics and Kinetics. https://www.scirp.org/journal/jamp

[11] Evelina Prozorova. On the question of the no symmetry of the stress tensor for open systems https://www.matec-conferences.org/articles/matecconf/abs/2022/09/contents/contents.html

[12] Неелов И. М. Введение в молекулярное моделирование биополимеров. https://books.ifmo.ru/file/pdf/2363.pdf

[13] University of Cambridge. The art of molecular dynamics simulation. https://www.cambridge.org/core/books/art-of-molecular-dynamics-simulation/57D40C5ECE9B7EA17C0E77E7754F5874

[14] Волков К. Н. Формулировка граничных условий на стенке в расчетах турбулентных течений на неструктурированных сетках. https://www.mathnet.ru/links/7676b4ca8f2c25d29232cfd6b9583876/zvmmf9995.pdf

[15] Шлик Т. Молекулярное моделирование и моделирование: междисциплинарное руководство.

[16] Rapaport DC. Искусство моделирования молекулярной динамики. 2-е изд. Издательство Кембриджского университета.

[17] Чистяков В. В. О динамике вращения твердого тела вокруг неподвижной оси, проходящей через центр масс, при сухом трении в опорах.

[18] Туркин В. В. Влияние смещения центра тяжести тела на точность определения момента инерции методом бифилярного подвеса.

[19] Е.В. Богатиков, Л.А. Битюцкая, А.Н. Шебанов. Моделирование нанокластеров методом молекулярной динамики. https://phys.vsu.ru/me/downloads/m13-178.pdf

[20] E. Pestryaev. Молекулярная динамика на персональном компьютере: учебное пособие.