Санкт-Петербургский политехнический
университет Петра Великого

Кафедра Теоретической механики

Дисциплина: Компьютерные технологии в механике

Реализация метода динамики частиц с использованием языка программирования C#

Работу выполнил:
студент кафедры
«Теор. механика»
группы 53604/1
Марков Николай

Преподаватель:
Ле-Захаров А.А.

2015

1 Постановка задачи

a) Реализовать программу моделирования методом динамики частиц на C#.

- Использовать подход ООП
- Выводить результат в формат, пригодный для визуализации движения частиц в 3D.

b) Провести привязку констант программы, параметров потенциала и т. д. к физическим параметрам моделируемого материала.

- Реализовать потенциал Леннард-Джонса или потенциал Ми либо другой потенциал на выбор.
- Реализовать создание решетки FCC для задания твердых тел

c) Провести расчет какой-либо реальной физической задачи с использованием написанной программы

2 Математическая модель

2.1 Потенциал взаимодействия

В данной работе в качестве потенциала взаимодействия между частицами взят потенциал Леннард-Джонса Π_{lj}, имеющий вид:

$$\Pi_{lj} = D\left[\left(\frac{a_0}{r}\right)^{12} - 2\left(\frac{a_0}{r}\right)^{6}\right] \tag{1}$$

где D - энергия взаимодействия; a_0 - длина связи; r - расстояние между частицами. Сила взаимодействия, соответствующая потенциалу Леннарда-Джонса, вычисляется по формуле:

$$F_{lj} = -\Pi'_{lj} = \frac{12D}{a_0}\left[\left(\frac{a_0}{r}\right)^{13} - \left(\frac{a_0}{r}\right)^{7}\right] \tag{2}$$

2.2 Список взаимодействующих частиц

Для увеличения скорости расчетов в программе используется список взаимодействующих частиц.

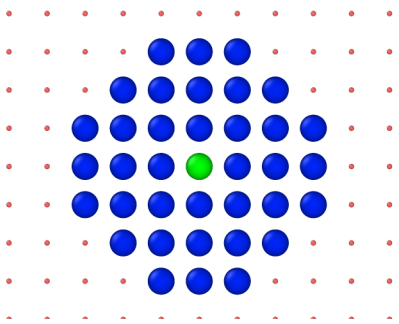

Рис. 1: Взаимодействие частиц

На Рис. 1 синим цветом маркированы все частицы, взаимодействующие с рассматриваемой (зелёной) частицей. Для задачи, рассматриваемой в данной работе,

достаточно составить список один раз. Если необходимо - список можно обновлять в течении работы программы. Критерием попадания частицы в список является выполнение условия (3):

$$r_{c,p} = \sqrt{\mathbf{r_c}^2 - \mathbf{r_p}^2 - 2r} \leqslant 4a_0 \tag{3}$$

где r_c - радиус-вектор рассматриваемой(зеленой) частицы; r_p - радиус-вектор любой другой частицы, r_p - радиус частицы. Если данное условие выполняется, то частица попадает в список.

2.3 Модифицированный алгоритм Верле (Leapfrog)

Для численного интегрирования уравнений движения в данной работе используется модифицированный алгоритм Верле

$$\ddot{\mathbf{r}}_i = \frac{d^2\mathbf{r}_i}{dt^2} = \mathbf{F}_i \tag{4}$$

или в равносильной форме

$$\dot{\mathbf{v}}_i = \frac{d\mathbf{v}_i}{dt} = \mathbf{F}_i, \;\; \dot{\mathbf{r}}_i = \frac{d\mathbf{r}_i}{dt} = \mathbf{v}_i \tag{5}$$

Плюс данного метода в высокой точности и стабильности, в сравнении с другими методами второго порядка, а так же то, что при интегрировании уравнений движения сохраняется энергия системы. В данной работе $dt = 0.004$.

3 Программная реализация

3.1 Начальная конфигурация

В данной работе рассматривается пробитие мишени пулей. Мишень является твердым телом с FCC решеткой, изображенной на Рис. 2.

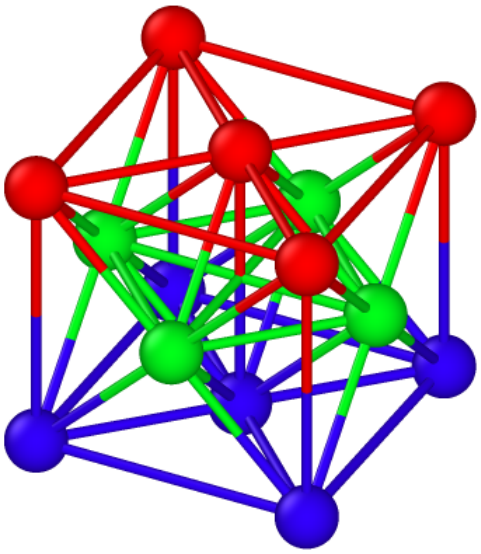

Рис. 2: FCC решётка

За безразмерные величины радиуса и массы приняты $r = 1, m = 1$ соответственно. Каждая частица рассматриваемого тела имеет радиус $r_{FCC} = r$ и массу $m_{FCC} = m$. Пуля, пробивающая мишень, имеет радиус $r_b = 16r$ и массу $m_b = 530m$. Предполагается, что в начальный момент времени пуля находится на расстоянии $R_0 = 8.6r$ от твердого тела и имеет скорость $V_0 = 3v$, где $v = 1$ - безразмерная величина. Начальная конфигурация модели представлена на Рис. 3. Для удобства визуального восприятия используется color gradient по оси OZ, который характеризует только позицию частицы по оси OZ.

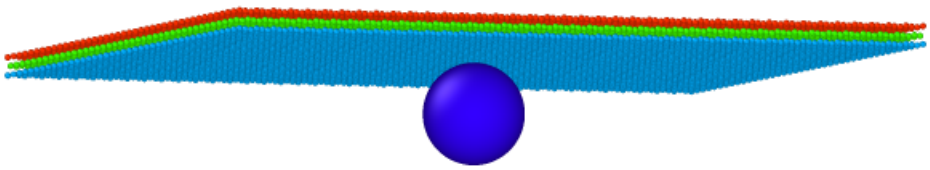

Рис. 3: Начальная конфигурация системы

3.2 Структура программы

Так как программа реализована с помощью языка программирования C#, то основой программы являются классы. В данной работе использовалось 8 различных классов:

- **Const.cs** - класс, содержащий в себе все константы, используемые в программе.

```
public static class Const
    {
        public static readonly int D = 1; //энергия связи
           в потенциале Л-Дж
        public static readonly int a0 = 1; //равновесное
           расстояние в потенциале Л-Дж
        public static readonly int a_cut = 4 * a0;
           //расстояние, отвечающее за определение
           соседних частиц
        public static readonly int n_wall_rows = 1;
           //число "строчек"  частиц в стене
        public static readonly int n_wall_col = 1; //число
           "столбцов"  частиц в стене
        public static readonly int id_standart = 0;
           //порядковый номер частицы
        public static readonly int type_standart = 1;
           //тип частицы(нужно для визуализации)
        public static readonly double MinMass = 0.001;
           //минимальная масса частицы (для отлова ошибок)
        public static readonly double MinDist = 0.001;
           //минимальное начальное расстояние между
           частицами (для отлова ошибок)
        public static readonly double DefaultMass = 1;
           //безразмерная массы
        public static readonly double DefaultDiameter = 1;
           //безразмерный диаметр частицы
        public static readonly double MaxTime = 80;
           //время интегрирования
        public static readonly double dt = 0.004; //шаг
           интегрирования
        public static readonly double BeginT = 0; //
           начальное время
    }
```

- **Vector.cs** - класс, позволяющий работать с векторами.
- **Particle.cs** - класс, позволяющий задать одну частицу.
- **Creator.cs** - класс, отвечающий за начальную конфигурацию системы.
- **WorkingModel.cs** - класс, позволяющий работать со всеми созданными частицами.
- **Integrator.cs** - класс, отвечающий за интегрирование уравнений движения частиц.
- **Optimization.cs** - класс, отвечающий за оптимизацию и ускорение вычислений.

- **OutputHelp.cs** - класс, позволяющий сохранять данные в пригодной для визуализации форме.

```
public class OutputHelp
    {
        public void SaveCurrentConfiguration(WorkingModel
           model)
        {
            string Text = "";
            Text = "ITEM: TIMESTEP" + "\r\n";
            Text += model.Time + "\r\n";
            Text += "ITEM: NUMBER OF ATOMS" + "\r\n";
            Text += model.size + "\r\n";
            Text += "ITEM: BOX BOUNDS ps ps ps" + "\r\n";
            Text += model._x + " " + model.x_ + "\r\n";
            Text += model._y + " " + model.y_ + "\r\n";
            Text += model._z + " " + model.z_ + "\r\n";
            Text += "ITEM: ATOMS id type xs ys zs" +
               "\r\n";

            foreach (var p in model.Particles)
            {
                Text += p.id + " " + p.type + " " + p.R.X
                   + " " + p.R.Y + " " + p.R.Z + "\r\n";
            }
           string NewText = Text.Replace(",",".");
           System.IO.File.AppendAllText(@"WriteLine.txt",
              NewText);
        }
    }
```

4 Результаты моделирования

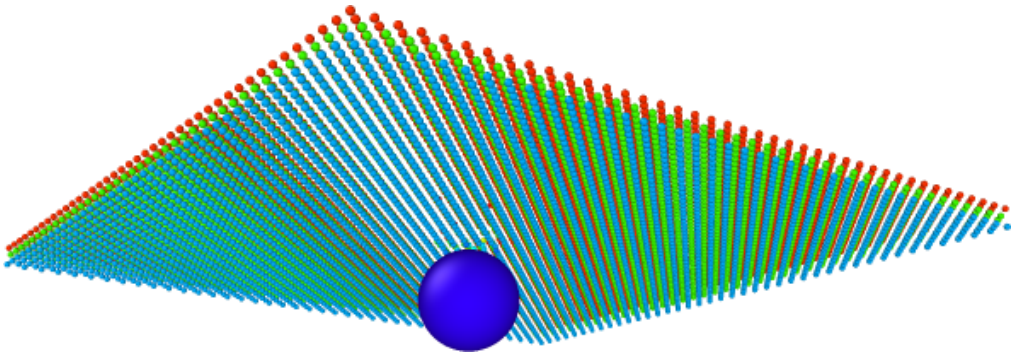

Рис. 4: кадр №0

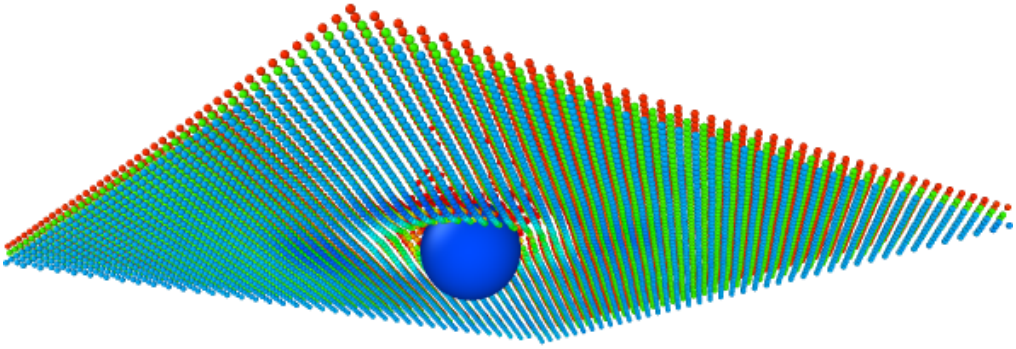

Рис. 5: кадр №1

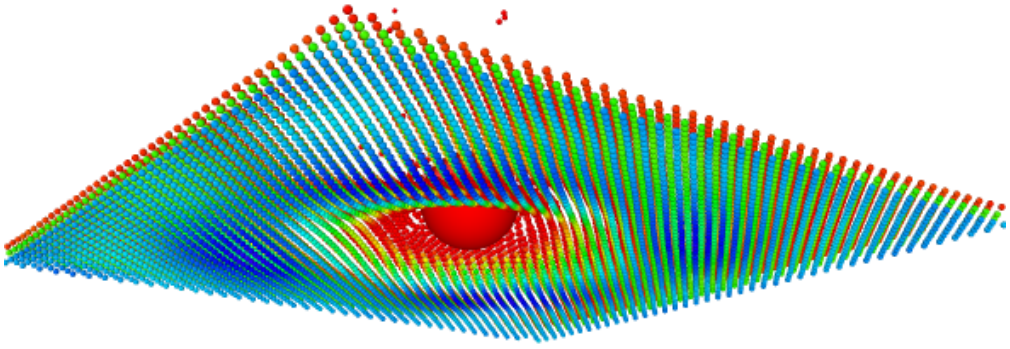

Рис. 6: кадр №2

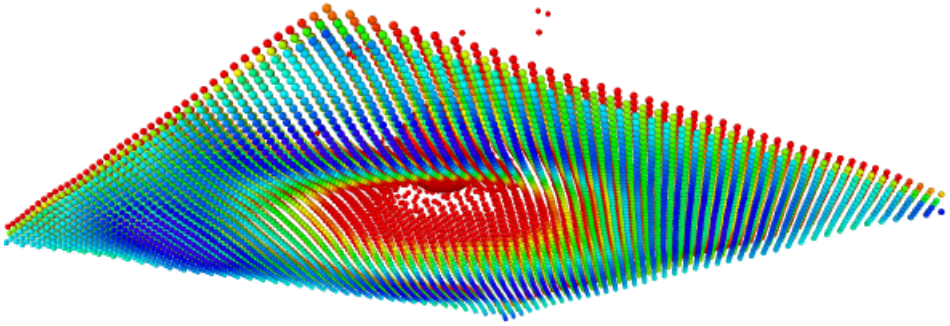

Рис. 7: кадр №3

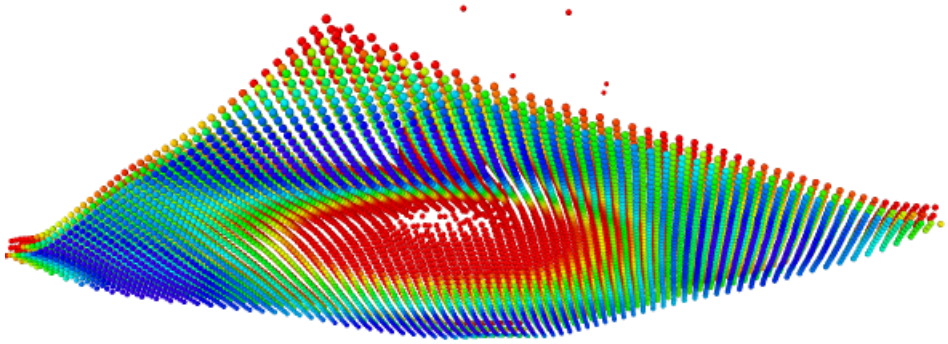

Рис. 8: кадр №4

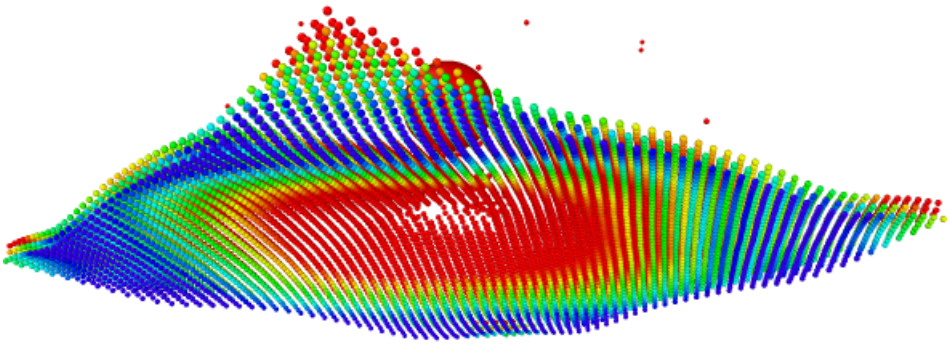

Рис. 9: кадр №5

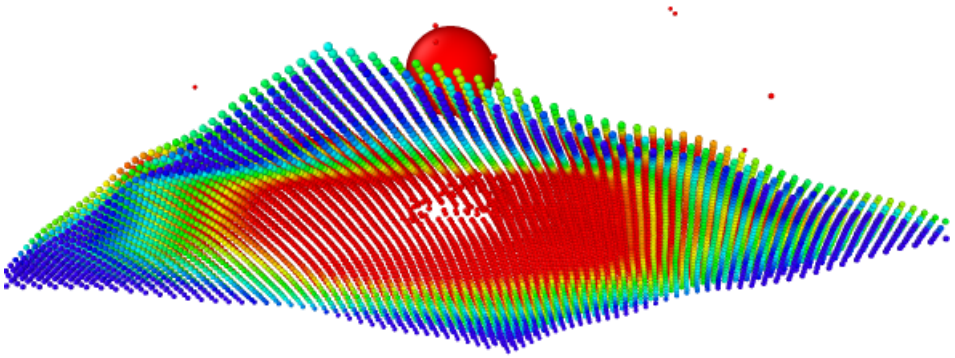

Рис. 10: кадр №6

5 Выводы

В результате работы над поставленной задачей написана программа на языке C#, в которой методом динамики частиц произведены расчеты пробития пулей твердого тела, имеющего FCC решётку. Для ускорения вычислений реализован список взаимодействующих частиц, что позволило провести расчеты для большого числа частиц. Вывод всех рассчитываемых величин производится в текстовый файл, имеющий структуру LAMMPS dump файла, что позволяет проводить визуализацию в OVITO.

На качественных результатах видно, что после прохождения пулей через тело конфигурация тела симметрична относительно оси, проходящей через его центр. Это связано с тем, что пуля попала в центр, соответственно при правильной работе программы осевая симметрия должна была наблюдаться в результатах.