САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО

(СПбПУ Петра Великого)

Институт прикладной механики и математики

Кафедра теоретической механики

Отчёт по дисциплине «Компьютерные технологии»

Моделирование опыта Резерфорда

работа студента группы 53604/1:

Антонова Ильи Денисовича

руководитель:

Ле-Захаров А.А.

Санкт-Петербург

2015

Содержание

Глава 1

Постановка задачи

1.1. Опыт Резерфорда

Опыт Резерфорда — эксперимент, который позволил получить новые данные о структуре вещества и выдвинуть предположения о строении атомов. Очень тонкая золотая фольга бомбардировалась α-частицами и изучалось распределение по углам рассеянных частиц. Эрнест Резерфорд получил формулу дифференциального сечения:

$$\frac{d\sigma}{d\Omega} = \frac{\alpha}{sin^4(\theta/2)}$$

α зависит от кинетической энергии налетающей частицы и зарядов взаимодействующих частиц.

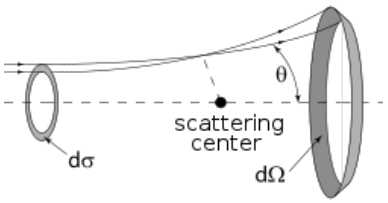

Рис. 1.1. Все частицы проходящие через кольцо слева попадают в кольцо справа

Согласно этой формуле, полученной по результатам эксперимента и основанной на новой модели атома, существуют частицы (примерно 1 из 8000 налетающих), которые будут рассеиваться под углами, большими 90 градусов. Популярная в то время предыдущая томпсоновская модель атома — большое положительно заряженное ядро с электронами внутри (пудинг с изюмом) — противоречила данным результатам, поскольку на больших расстояниях (так как ядра больше, максимальное сближение $R_1 + R_2$) действие кулоновских сил не могло быть настолько сильным, чтоб отклонять частицы на такие большие углы. А существование таких сил приводит к тому факту, что размер ядер частиц должен быть много меньше размера атома, вся его масса и заряд содержится внутри очень малой частицы. Таким образом, атом должен преимущественно состоять из пустоты.

1.2. Цель работы

Цель работы — смоделировать опыт Резерфорда используя кулоновский потенциал взаимодействия; качественно обнаружить частицы, отклоняющиеся на большие углы.

Глава 2

Реализация

2.1. Структура программы

Для реализации опыта была разработана библиотека классов ParticlesSimulation на языке C# с использованием основ ООП. Библиотека позволяет создавать модель пространства, заполненного частицами, после чего может определять динамическое поведение созданной системы для выбранного потенциала парного взаимодействия. Были разработаны следующие классы:

1. *Constants.cs* — класс, содержащий основные константы для всех вычислений — массы, заряды и диаметры частиц; время и шаг интегрирования; константы, задающие потенциал;

2. *Creator.cs* — класс, предназначенный для создания системы *Space*: заполнение её частицами *Particle* с заданными массами, зарядами, координатами и скоростями;

3. *IntegratorBase.cs* — класс, позволяющий рассчитывать поведение системы *Space*, используя потенциал гармонического осциллятора;

4. *IntegratorLJ.cs* — производный класс *IntegratorBase.cs*, позволяющий рассчитывать поведение системы *Space*, используя потенциал Леннард-Джонса;

5. *IntegratorQ.cs* — производный класс *IntegratorBase.cs*, позволяющий рассчитывать поведение системы *Space*, используя потенциал Кулона;

6. *OutputHelper.cs* — класс, позволяющий выводить данные системы *Space* в текстовый файл;

7. *OutputHelperA3R.cs* — производный класс *OutputHelper.cs*, позволяющий выводить данные системы *Space* в формате, пригодном для визуализации в программе A3R.

8. *Particle.cs* — класс, описывающий частицу;

9. *Space.cs* — класс, описывающий систему;

10. *Vector3D.cs* — класс, описывающий трехмерные вектора;

2.2. Расчёт и моделирование

Для данного эксперимента был выбран *IntegratorQ.cs* для расчёта. Потенциал оборван на расстоянии в четыре диаметра частицы. Код класса можно посмотреть в приложении А.

В классе *Creator.cs* был смоделирован кубик 5x5x5 золотой фольги, на который налетают случайные частицы. Для частиц кубика была установлена очень высокая масса, для того чтобы налетающие частицы не разрушали кристаллическую структуру. Заряд у них выбран на порядок выше, чем у налетающих альфа-частиц, что соответствует действительности. В реальном эксперименте — структура имеет очень много слоёв, от этого растёт вероятность наблюдать отраженные под большими углами частицы. В компенсацию этого, сокращаем расстояние между частицами в решётке до семи диаметров (всё ещё достаточно большое расстояние, масса и заряд сосредоточены в $\frac{1}{50}$ объема атома). Теперь вероятность отразиться с большим углом выше, достаточная для качественной регистрации интересующего эффекта. Код класса, задающего систему, можно посмотреть в приложении Б.1.

Константы вычислений из класса *Constants.cs* также приведены в приложении Б.2.

Глава 3

Результаты

В кубик было запущено 100 частиц, 2 из которых отразились под большим углом. Ниже приведена реализация, полученная в программе A3R. На рисунках зелёным цветом отмечены налетающие частицы, красным — отраженные под большим углом. Можно сказать, что ожидаемый результат успешно получен.

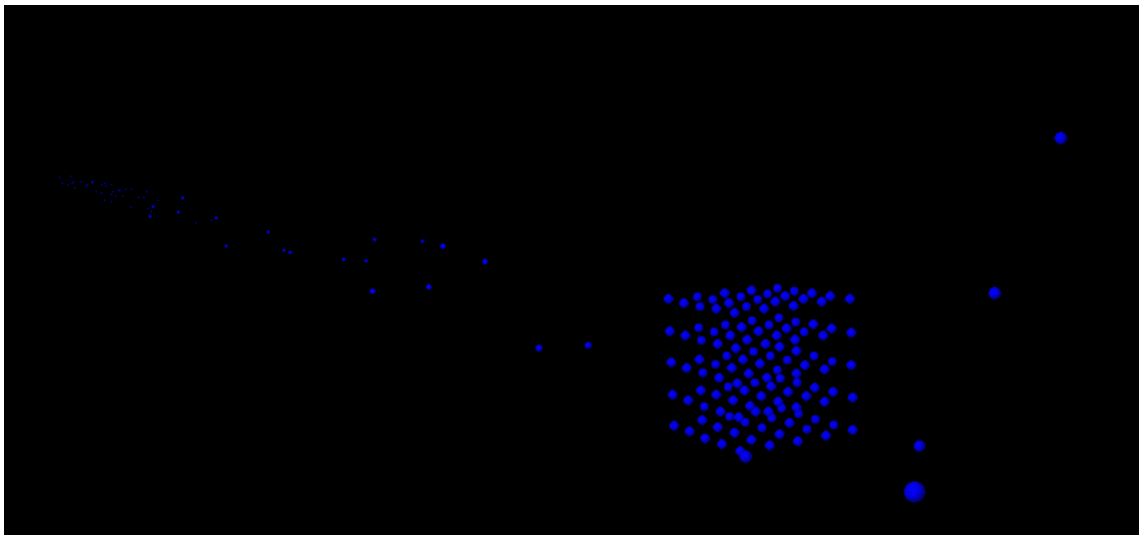

Рис. 3.1. Модель эксперимента

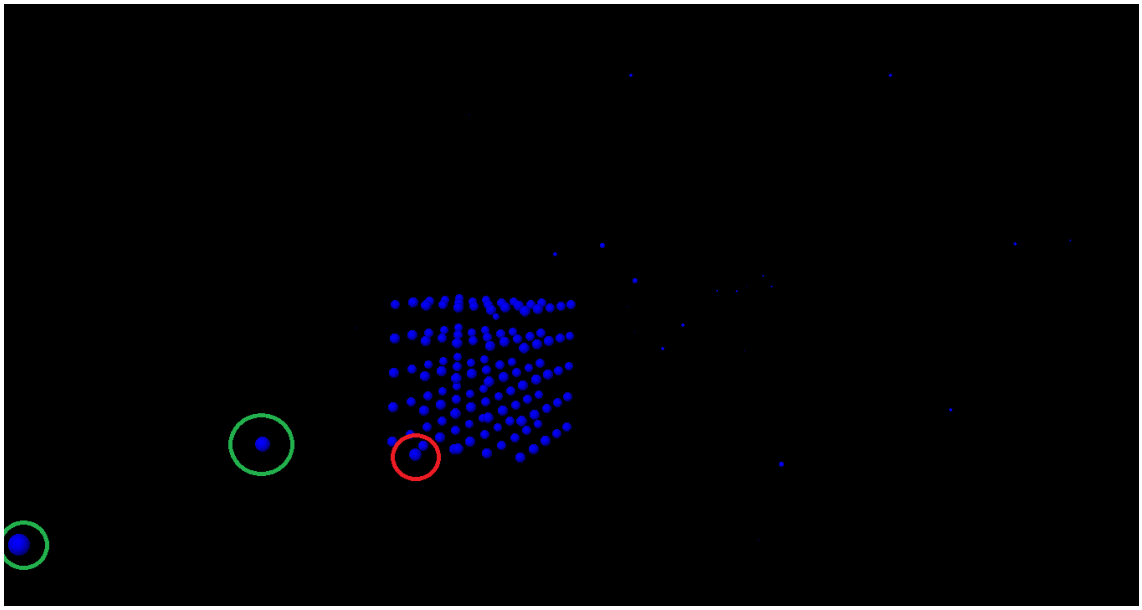

Рис. 3.2. Частица, отраженная под большим углом. Кадр 1

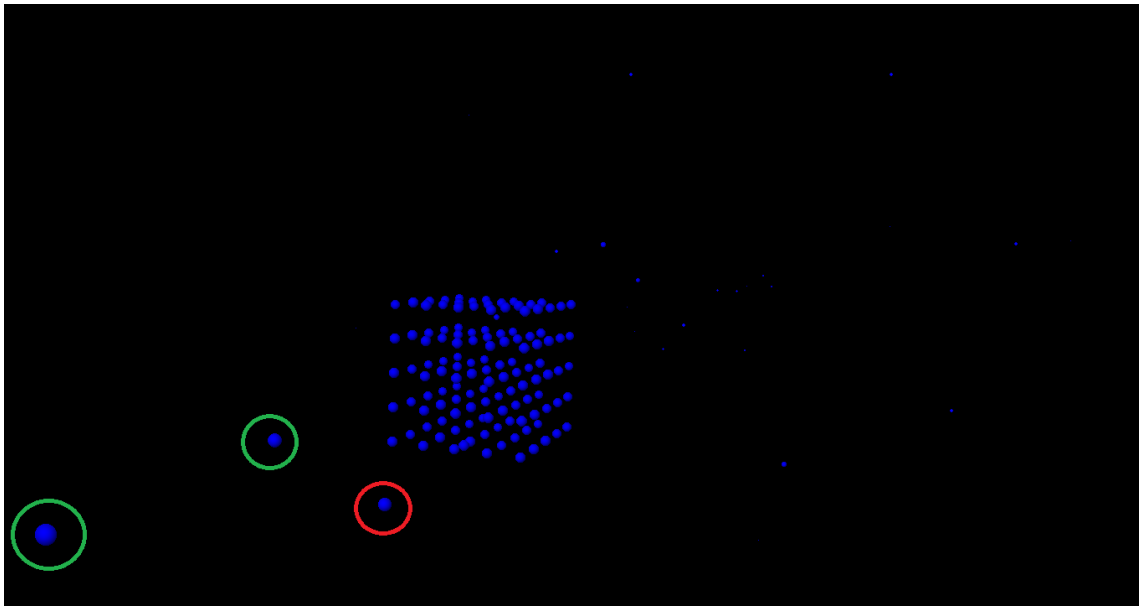

Рис. 3.3. Частица, отраженная под большим углом. Кадр 2

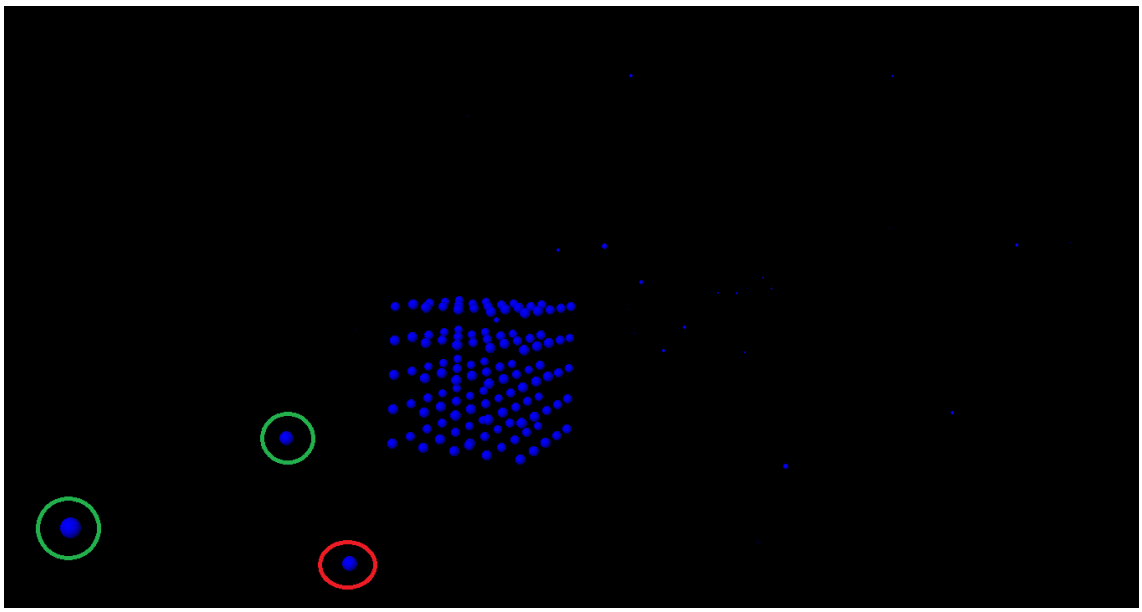

Рис. 3.4. Частица, отраженная под большим углом. Кадр 3

Выводы

Был успешно смоделирован и реализован опыт Резерфорда при помощи средств ООП, получен ожидаемый результат. Также получен опыт реализации численных вычислений на языке C#, изучены основы и преимущества объектно-ориентированного подхода к написанию решения задач.

Приложение А

Creator.cs

```
п»їusing System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ParticlesSimulation
{
    public class Creator
    {
        public Space Create()
        {
            Space model = new Space();
            /*
            Particle p1 = new Particle(0,0,1);
            Particle p2 = new Particle(0,0,9);
            p2.V.z = -1.0;
            model.Add(p1);
            model.Add(p2);
            */
            for (int i = 0; i < 5; ++i)
            {
                for (int j = 0; j < 5; ++j)
                {
                    for (int k = 0; k < 5; ++k)
                    {
                        Particle p = new Particle(i * 7.0, j * 7.0, k * 7.0);
                        p.Q = Constants.DefaultQ * 90;
                        p.M = Constants.DefaultMass * 40000;
                        model.Add(p);
                    }
                }
            }
            Random timer = new Random();
            for (int i = 0; i < 100; ++i)
            {
                double xCoord = timer.NextDouble() * 28.0;
                double yCoord = timer.NextDouble() * 28.0;
                double zCoord = timer.NextDouble() * 5000.0 + 60.0;
                Particle p = new Particle(xCoord, yCoord, zCoord);
                p.V.z = -20.0;
                p.V.x = 0;
                p.V.y = 0;
                p.Q = Constants.DefaultQ;
                model.Add(p);
            }
            return model;
        }
    }
}
```

Приложение Б

IntegratorQ.cs и Constants.cs

Б.1. IntagratorQ.cs

```
п»їusing System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ParticlesSimulation
{
    public class IntegratorQ : IntegratorBase
    {
        public IntegratorQ(Space s) : base(s) { }

        public override void RecalcParticleForce(Particle pi, Particle pj)
        {
            double ModFij = 0;
            if ((pi.R - pj.R).Norm() < pi.D * 2.0 + pj.D * 2.0){
                ModFij = Constants.K * pi.Q * pj.Q / Math.Pow((pi.R - pj.R).Norm(), 2.0) / pi.M;
            }
            pi.F = pi.F + (ModFij / (pi.R - pj.R).Norm()) * (pj.R - pi.R);
        }
    }
}
```

Б.2. Constants.cs

```
п»їusing System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ParticlesSimulation
{
   public static class Constants
    {
        public static readonly double
            MinMass = 0.001, DefaultMass = 1, DefaultD = 1, MinDistance = 0.0001, dt = 0.04,
                MaxTime = 400.0, K=1.0, DefaultQ = 1.0;

    }
}
```