САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО

Институт прикладной математики и механики

Кафедра теоретической механики

Отчёт по дисциплине «Компьютерные технологии»
«Моделирование столкновения пули со стеной»

Выполнил: студент группы 53604/1
Редькин Евгений Александрович

Руководитель:
Ле-Захаров А.А.

Санкт-Петербург
2015

Постановка задачи

Необходимо провести моделирования взаимодействия частиц и одна из задач является столкновения пули со стеной. Результаты моделирования данной задачи позволяют наглядно увидеть особенности взаимодействия частиц разного рода материалов, влияние скоростей, массы и диаметра частиц на эти взаимодействия.

Цель данной работы: смоделировать столкновение пули со стеной. Взаимодействие частиц между собой будет происходить через потенциал Леннард-Джонса.

Математическая модель

Потенциал и сила взаимодействия Леннарда-Джонса имеют вид

$П(r) = D\left[\left(\frac{a}{r}\right)^{12} - 2\left(\frac{a}{r}\right)^{6}\right]$, где D- энергия связи, α– равновесия расстояния

Сила взаимодействия рассчитывается по формуле:

$$F(r) = -\nabla П(r) = \frac{12D}{a^2}(\left(\frac{a}{r}\right)^{14} - \left(\frac{a}{r}\right)^{8})r$$

Реализация

Для реализации опыта была разработана библиотека классов ParticlesSimulation на языке C# с использованием основ ООП. Были разработаны следующие классы:

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

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

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

IntegratorLJ.cs — производный класс IntegratorBase.cs, позволяющий рассчитывать поведение системы Space;

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

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

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

Particle.cs — класс, описывающий частицу;

Space.cs — класс, описывающий систему;

Vector3D.cs — класс, описывающий трехмерные вектора;

Программная реализация

Для создания модели поставленной задачи будет использоваться метод динамики частиц, была взята пуля, летящая в стенку, форма была выбрана в виде параллелепипеда с заданными параметрами, указанными в Приложении, а мишенью была стена, которая представляет собой небольшой слой из частиц. Строение соответствует гранецентрированной решетке (ГЦР). Шаг интегрирования dt = 0.05

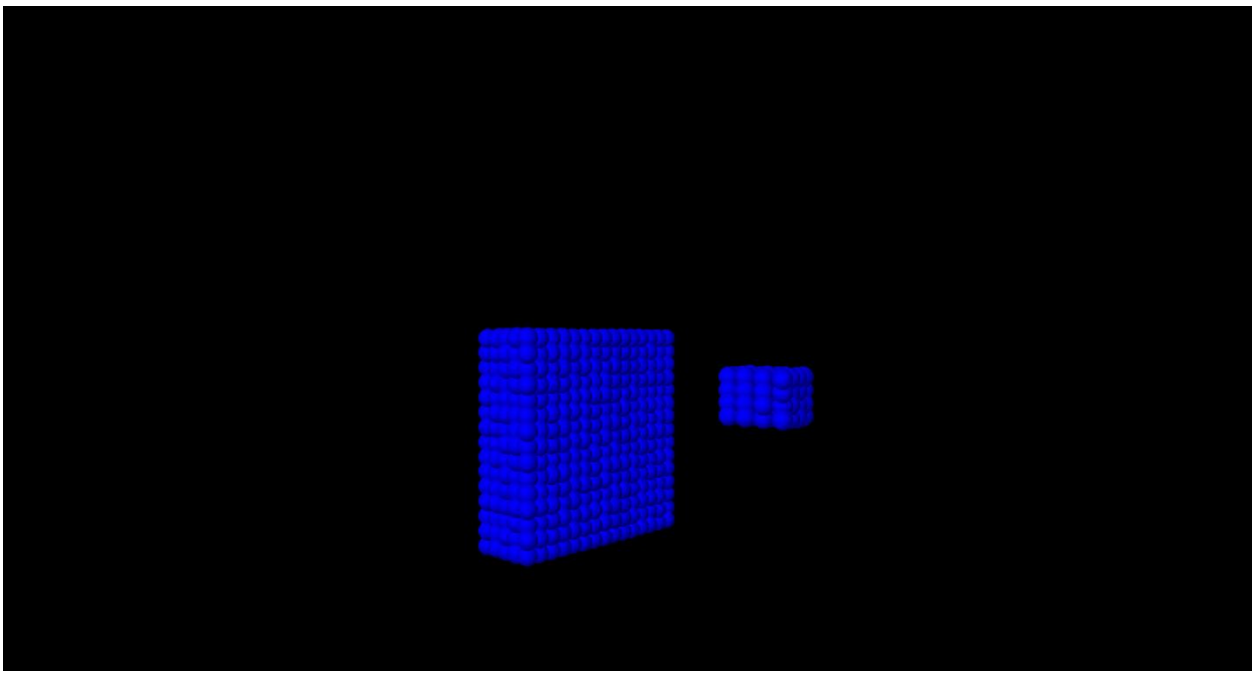

Рис 1. Летящая пуля в начальном положении

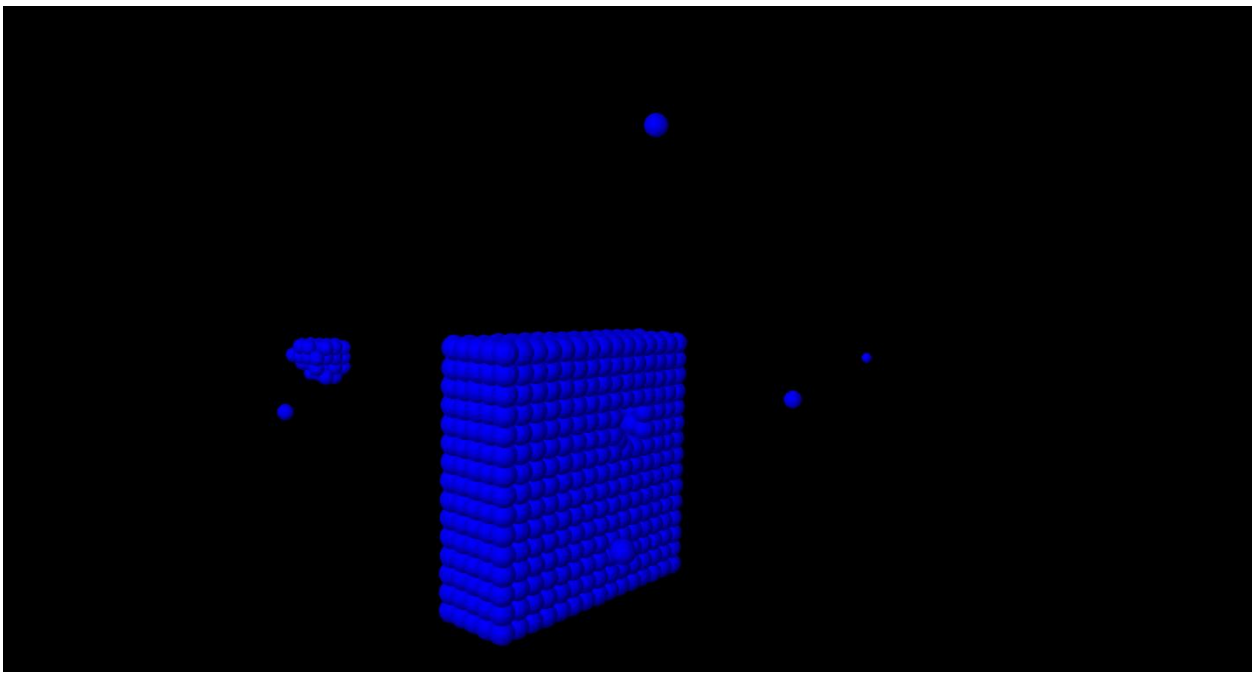

Рис 2. Пуля после столкновения со стеной

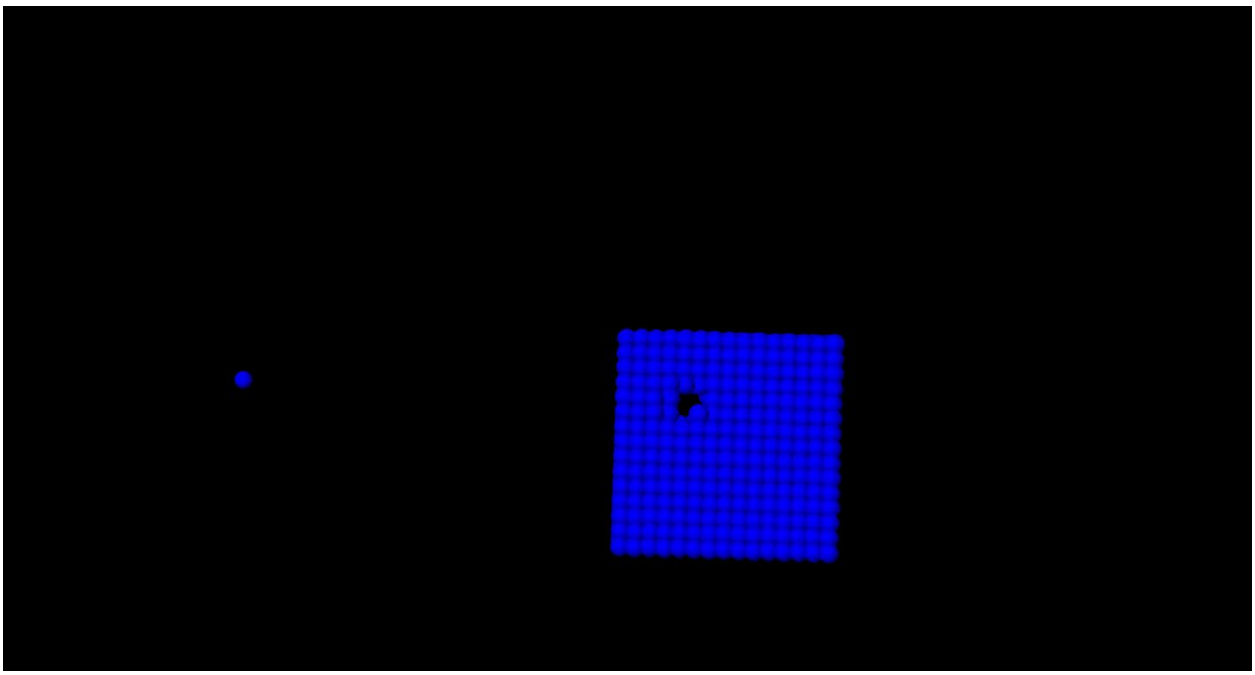

Рис 3. Отверстие после пули

В результате моделировании можно наблюдать, что пуля, проходя сквозь стенку, оставляет после себя отверстие соответствующее своим размерам. Однако на выходе за счет взаимодействия частиц между собой в стенке, данное отверстие сужается. Это можно объяснить за счет отталкивание соседних частиц по потенциалу Леннард – Джонсона из–за нарушения первоначальной структуры.

Выводы

Была создана программа моделирования взаимодействия частиц на основе потенциала Леннард-Джонса на языке C#, изучены основы и преимущества объектно-ориентированного подхода к написанию решения задач.

Приложение А.
Creator.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ParticlesSimulation
{
     public class Creator
    {
         public Space Create()
         {
             Space model = new Space();


              for (int i = 0; i < 15; ++i)              {
                  for (int j = 0; j < 15; ++j)
                  {
                      for (int k = 0; k < 5; ++k)
                      {
                          Particle p = new Particle(i * 6.0, j * 6.0, k * 6.0);
                          //p.D = Constants.DefaultD * 5;
                          p.M = Constants.DefaultMass * 2000;
                          model.Add(p);
                      }
                  }
              }

              for (int i = 0; i < 4; ++i)
              {
                  for (int j = 0; j < 4; ++j)
                  {
                      for (int k = 0; k < 4; ++k)
                      {
                          Particle p2 = new Particle(i * 4.0 + 15, j * 4.0 + 15, k * 8.0
+ 80.0);
                          p2.D = Constants.DefaultD * 5;
                          p2.M = Constants.DefaultMass * 26000;
                          p2.V.z = -50.0;
                          p2.V.x = 0.0;
                          p2.V.y = 0.0;
                          model.Add(p2);
                      }
                  }
              }
```