

#09, сентябрь 2015

УДК 004.94

Система для моделирования и рендеринга движения частиц

Макаров Д. В., студент

*Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Программное обеспечение ЭВМ и информационные технологии»*

Научный руководитель: Степанов В. П., к.т.н., доцент

*Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Программное обеспечение ЭВМ и информационные технологии»*

bauman@bmstu.ru

Введение. В процессе разработки в проектах различной сложности часто возникает необходимость в моделировании того или иного процесса. Это позволяет определить то, как поведет себя система до того, как будет готова ее реализация. Результат моделирования несжимаемой жидкости представлен в работе [1]. Целью данной работы является рендеринг и моделирование движения элементарных частиц с точки зрения молекулярной динамики.

Постановка задачи. Реализовать графический движок, физический движок, который будет моделировать процесс согласно модели молекулярной динамики реализовать пользовательский интерфейс, распараллелить приложение.

Для обоих движков выберем объектно-ориентированную парадигму разработки, так как это обеспечит гибкость в их разработке. Физический движок должен быть легко масштабируемым для того, чтобы при необходимости в него можно было добавить новый функционал для моделирования явлений, отличных от тех, которые изначально предполагались при проектировании. Графический движок должен поддерживать освещение и камеру от первого лица FPS-камеру. FPS-камера обеспечит привычный для пользователя метод перемещения по трехмерному пространству. Оба движка должны представлять собой самостоятельные библиотеки, который при необходимости можно подключить к другим проектам.

При реализации пользовательского интерфейса необходимо учитывать тот факт, что процесс вывода изображения и моделирования может занять много времени, поэтому под

эти процессы необходимо выделить отдельные потоки, чтобы обеспечить безотказный отклик интерфейса.

Математическая модель движения частиц. Пусть дано множество частиц P , каждый элемент которого имеет следующие характеристики m (масса), r (радиус), ch (заряд), $\vec{v}$ (скорость). Тогда будем учитывать в модели действие следующих сил:

1. Сила тяготения $\vec{F} = G \frac{m_1 m_2}{r^3} \vec{r}$, где m_i – масса частицы, $\vec{r}$ – вектор, соединяющий центры частиц, G – гравитационная постоянная;
2. Закон Кулона $\vec{F} = k \frac{q_1 q_2}{r^3} \vec{r}$, где q_i – заряд частицы, $\vec{r}$ – вектор, соединяющий центры частиц, k – электрическая постоянная;
3. Сила Лоренца для заряда, движущегося в магнитном поле, созданном другим движущимся зарядом $\vec{F} = q[\vec{v}, \vec{B}]$, где q – заряд частицы, $\vec{v}$ – скорость частицы, $\vec{B} = \frac{\mu_0}{4\pi} \frac{Q}{r^3} [\vec{V}, \vec{r}]$, где Q – заряд частицы, создающей поле; $\vec{V}$ – скорость движения частицы, создающей поле; $\vec{r}$ – вектор, соединяющий центры частиц.

Для расчета движения частиц используются уравнения Ньютоновской механики: второй закон Ньютона, закон сохранения энергии и импульса. Δt выбирается согласно особенностям поставленной задачи и необходимой точности вычислений. В программе будет использоваться итерационный метод моделирования.

Программная реализация. В процессе работы программа совершает различные трансформации над объектами, используя аффинные преобразования [2]. Для трехмерного пространства любое аффинное преобразование может быть представлено последовательностью простейших операций.

Ниже приводятся уравнения и матрицы преобразований:

- сдвиг точки вдоль координатных осей на dx , dy , dz :

$$\begin{cases} X = x + dx \\ Y = y + dy \\ Z = z + dz \end{cases} \quad \begin{pmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{pmatrix};$$

- масштабирование относительно начала координат с коэффициентами k_x , k_y , k_z :

$$\begin{cases} X = \frac{x}{k_x} \\ Y = \frac{y}{k_y} \\ Z = \frac{z}{k_z} \end{cases} \quad \begin{pmatrix} \frac{1}{k_x} & 0 & 0 & 0 \\ 0 & \frac{1}{k_y} & 0 & 0 \\ 0 & 0 & \frac{1}{k_z} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- поворот относительно осей x, y, z на угол φ :

О ось x:

$$\begin{cases} X = x \\ Y = y \cos \varphi + z \sin \varphi \\ Z = -y \sin \varphi + z \cos \varphi \end{cases} \quad \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \varphi & \sin \varphi & 0 \\ 0 & -\sin \varphi & \cos \varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

О ось y:

$$\begin{cases} X = x \cos \varphi + z \sin \varphi \\ Y = y \\ Z = -x \sin \varphi + z \cos \varphi \end{cases} \quad \begin{pmatrix} \cos \varphi & 0 & \sin \varphi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \varphi & 0 & \cos \varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

О ось z:

$$\begin{cases} X = x \cos \varphi + y \sin \varphi \\ Y = -x \sin \varphi + y \cos \varphi \\ Z = z \end{cases} \quad \begin{pmatrix} \cos \varphi & \sin \varphi & 0 & 0 \\ -\sin \varphi & \cos \varphi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- поворот вокруг произвольной оси

$\vec{v} = (x, y, z)$ – ось, θ – угол поворота

$$\begin{pmatrix} \cos \theta + (1 - \cos \theta)x^2 & (1 - \cos \theta)xy - (\sin \theta)z & (1 - \cos \theta)xz + (\sin \theta)y \\ (1 - \cos \theta)yx + (\sin \theta)z & \cos \theta + (1 - \cos \theta)y^2 & (1 - \cos \theta)yz - (\sin \theta)x \\ (1 - \cos \theta)zx - (\sin \theta)y & (1 - \cos \theta)zy + (\sin \theta)x & \cos \theta + (1 - \cos \theta)z^2 \end{pmatrix}$$

В реализации программы используются преобразования, которые указаны в системах уравнений, так как при этом нужно совершить меньшее количество вычислений, чем при перемножении матриц с матрицами/векторами. Исключение составляет последнее преобразование: для него количество необходимых вычислений одинаково для матричного представления и для скалярного.

Проецирование. При использовании графических устройств обычно используют проекции. Проекция задает способ отображения объекта на графическом устройстве. При отображении пространственных объектов на экране необходимо знать координаты

объектов. Рассмотрим трехмерную декартову систему координат. Для синтеза изображения на плоскости достаточно, казалось бы, двумерной экранной системы координат, однако из-за использования алгоритма Z-буфера, они должны быть трехмерными [3].

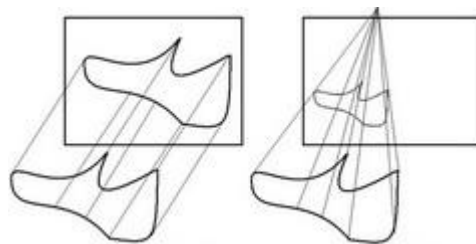


Рис. 1. Параллельная и центральная проекции

Для центральной проекции лучи проецирования исходят из одной точки, размещенной на конечном расстоянии от объектов и плоскости проецирования. Для параллельной проекции лучи проецирования параллельны (Рис. 1).

В данной работе для визуализации сцены использовалась центральная проекция.

Центральная проекция обычно определяется таким образом:

$$\begin{bmatrix} \frac{f}{\text{aspect}} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{z_{\text{Near}} + z_{\text{Far}}}{z_{\text{Near}} - z_{\text{Far}}} & \frac{2 \cdot z_{\text{Near}} \cdot z_{\text{Far}}}{z_{\text{Near}} - z_{\text{Far}}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

где $f = \text{ctg}(\frac{fov}{2})$, aspect = ширина окна / высота окна, zNear, zFar – ближняя и дальняя плоскости, fov – угол обзора в градусах.

Такое матричное представление данной проекции потребует некоторого количества дополнительных вычислений. Чтобы этого избежать определяется операция проецирования следующим образом. Пусть даны width - ширина, height - высота окна, fov - угол обзора. Тогда расстояние до картинной плоскости рассчитаем следующим образом:

$$d = \frac{\text{width}}{2} \text{tg}(\frac{fov}{2}).$$

Если вершина в мировой системе координат задается радиус-вектором $\vec{r}(x, y, z)$, тогда радиус-вектор этой вершины на картинной плоскости определяется так: $\vec{r'} = \frac{d}{z} \vec{r}$.

Построение модели сферы. В работе для построения модели сферы используется геодезический купол, сетка, построенная из множества граней, максимально близкая к форме сферы. За основу берется икосаэдр. Все вершины икосаэдра лежат на сфере радиуса R . Далее для каждой грани икосаэдра применяется алгоритм из трех шагов:

1. Найти середину треугольник (точка пересечения медиан) (Рис. 2).
2. Найти середину каждого ребра треугольника (Рис. 3).
3. Заменить исходную грань 6 гранями, предварительно нормализовав и помножив на R радиус-вектор полученных в пунктах 1 и 2 вершин (Рис. 4).

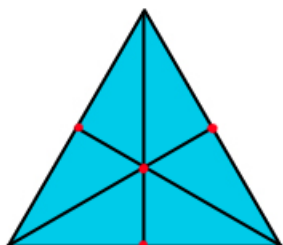


Рис. 2. Шаг 1, 2

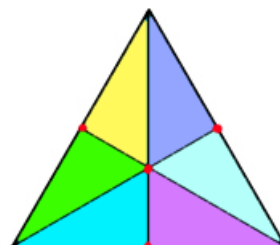


Рис. 3. Шаг 3

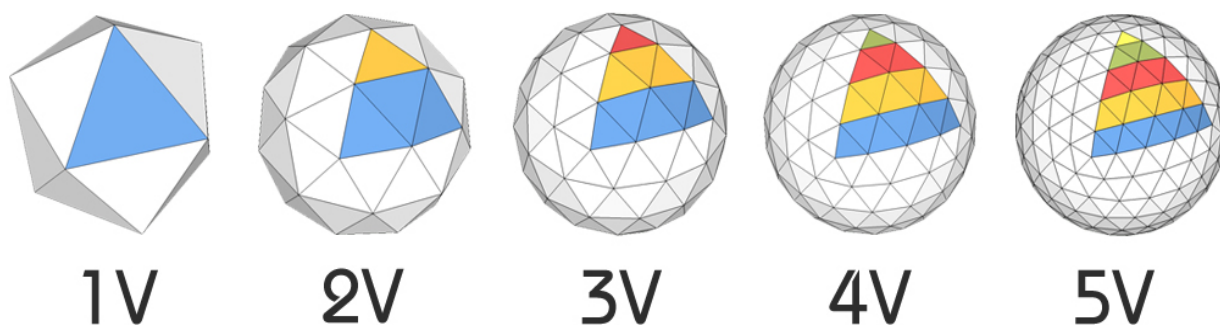


Рис. 4. Результат работы алгоритма

Программная реализация. Для реализации данного программного продукта выбран язык native C++, так как он поддерживает объектно-ориентированную парадигму программирования и компилируется в машинный код. Для разработки интерфейса пользователя используется библиотека Qt5.

Реализация физического движка продумана таким образом, чтобы движок можно было в дальнейшем развивать. В движке предусмотрена гибкая структура классов для различных объектов сцены. В движок не составит большого труда добавить новый вид силы, тела или поля. Также в движке присутствует такое понятие, как вычислитель. Процесс вычисления можно организовать не только на центральном процессоре, но, например, и на графическом процессоре. В данной реализации присутствует алгоритм расчета на CPU. Создав производный класс от PhysEstimator, можно реализовать вычисления, например, на CUDA [4] (технология от NVidia для реализации параллельных

вычислений на базе графического процессора). При реализации вычислений на CUDA есть ряд сложностей, одна из которых перенос данных на GPU. Программный интерфейс движка разработан так, что при необходимости переноса вычислений на GPU не возникнет сложностей.

Графический движок разработан по объектно-ориентированной парадигме и построена соответствующая иерархия классов. При проектировании минимизированы затраты, связанные с передачей данных в программе. В движок введено понятие модели-ссылки. Модель-ссылка – это модель, которые ссылается на информацию о точках и примитивах другой модели, но при рендеринге использует свои преобразования для данной модели. Это позволило реализовать в программе создание базовой модели для какого-то конкретного типа частиц, а потом размещать на сцене произвольное количество таких моделей, каждая из которых считается за самостоятельный объект. В движке нет обычных указателей. В нем используется самостоятельная реализация паттерна «умный указатель». Благодаря этому не происходит утечка памяти. Так как процесс рендеринга занимает достаточно большое время, то интерфейс пользователя разработан так, чтобы он обрабатывал ввод пользователя независимо от сложности сцены. Для достижения данной цели для каждого окна вывода создается отдельный поток с очередью запросов. Такой подход позволяет приложению постоянно взаимодействовать с пользователем.

Описание пользовательского интерфейса. На рисунке 5 изображено главное окно программы.

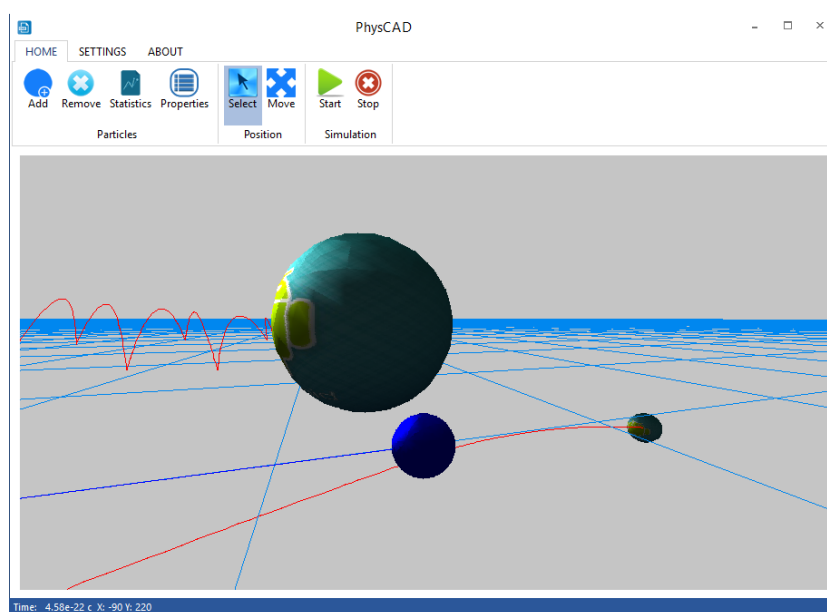


Рис. 5. Основное окно приложения

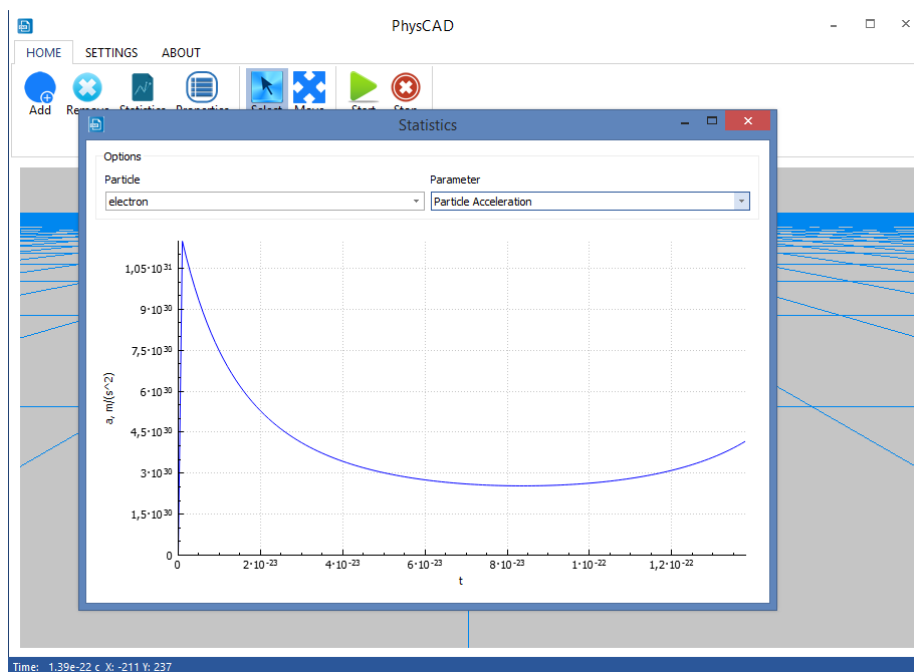


Рис. 6. Окно графиков зависимостей

В окне статистики рис.6 можно выбрать, для какой частицы мы хотим ее посмотреть и график какой величины, мы хотим посмотреть. В окне типов частиц можно добавить частицу, удалить частицу, задать ее свойства и условный внешний вид. В настройках можно указать степень детализации, шаг камеры и шаг перемещения объектов. Объекты расположены в метрическом пространстве, которое проецируется на экран. Движение в пространстве осуществляется с помощью клавиш W движение вперед, S движение назад, либо с помощью стрелок на клавиатуре для вращения камеры, либо с помощью мыши.

Экспериментально-исследовательская часть. Проведены эксперименты с целью проверки работоспособности программы. Произведены замеры производительности графического движка в зависимости от количества полигонов на сцене. Полученные результаты, имеющие вид экспоненциальной зависимости, приведены на рис.7. Это связано с тем, что все вычисления для рендеринга выполняются на CPU.

Для демонстрации возможностей системы рассмотрим расстановку частиц, приведенную на рис. 8. Синий – это протон, сфера с текстурой – это электрон. Пользователю доступно задание положения частиц как в момент их создания, так и после. Для того чтобы сменить положение частицы, пользователю нужно выделить ее мышью, а затем с помощью стрелок на клавиатуре передвинуть частицу вдоль координатных осей. Шаг смещения задается в настройках. Это позволит пользователю выбирать те значения смещения, которые ему необходимы в данной конкретной задаче.

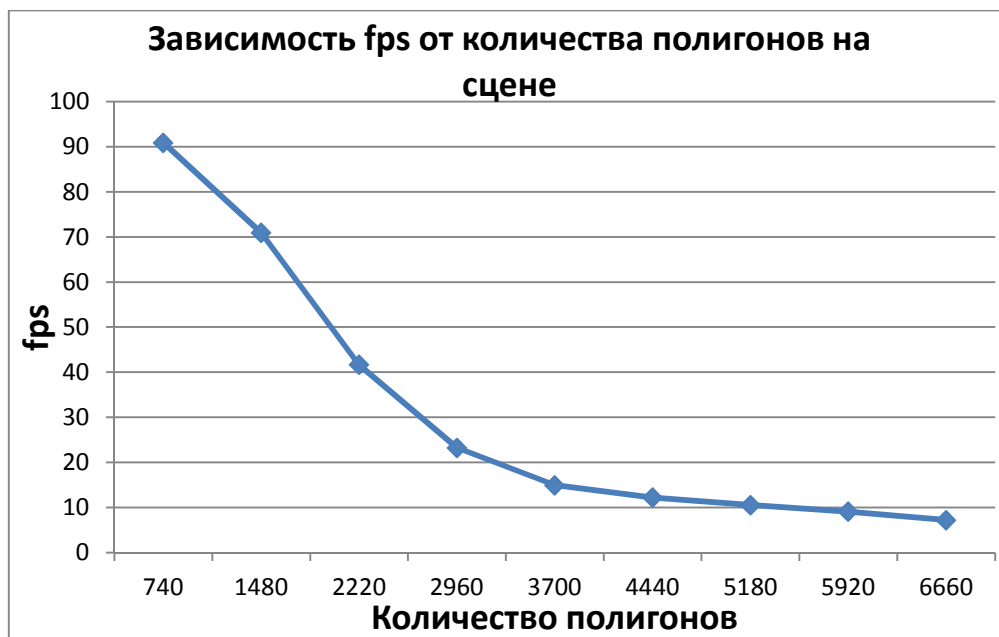


Рис. 7. График зависимости количества кадров в секунду от числа полигонов

Пользователь может остановить процесс моделирование в необходимый ему момент времени и посмотреть параметры каждой частицы. Запускается процесс итерационного моделирования. После $1.64 \cdot 10^{-22}$ секунд получается картина, приведенная на рис. 9.

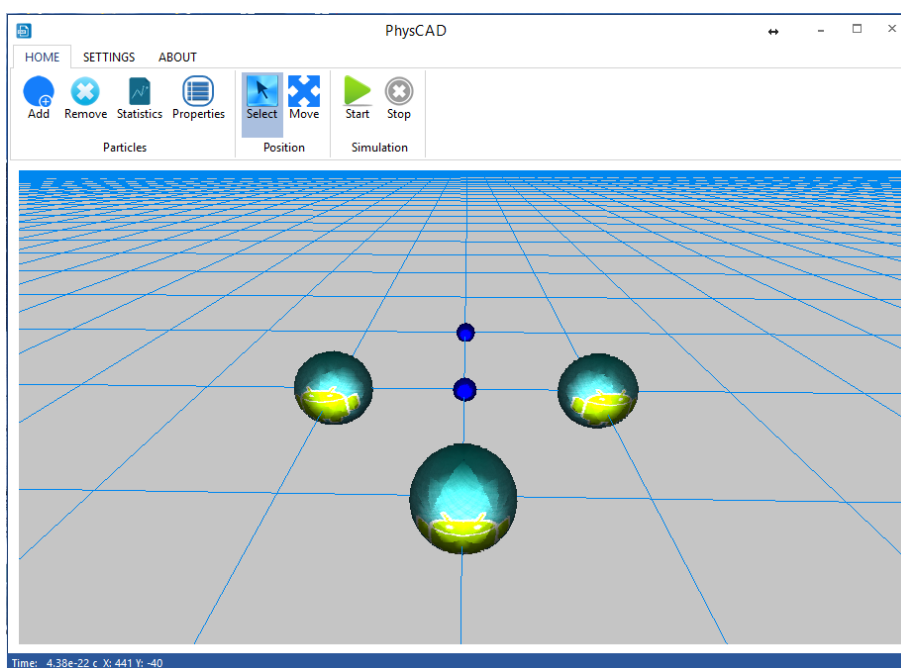


Рис. 8. Начальная расстановка частиц

В процессе моделирования сохраняется информация о положении частицы, начиная с начального момента времени. Благодаря этому пользователю также выводится траектория движения частиц (рис. 9). График, приведенный на рис. 10 показывает, как изменяется модуль ускорения частицы со временем. Так как частицы двигались, то модули сил,

действующих на частицы, менялись, следовательно, и влияние частиц друг на друга тоже изменялось. Разработанная программа позволяет собрать достаточно большую статистику о движении частиц, что может использоваться в исследованиях в области молекулярной физики и термодинамики.

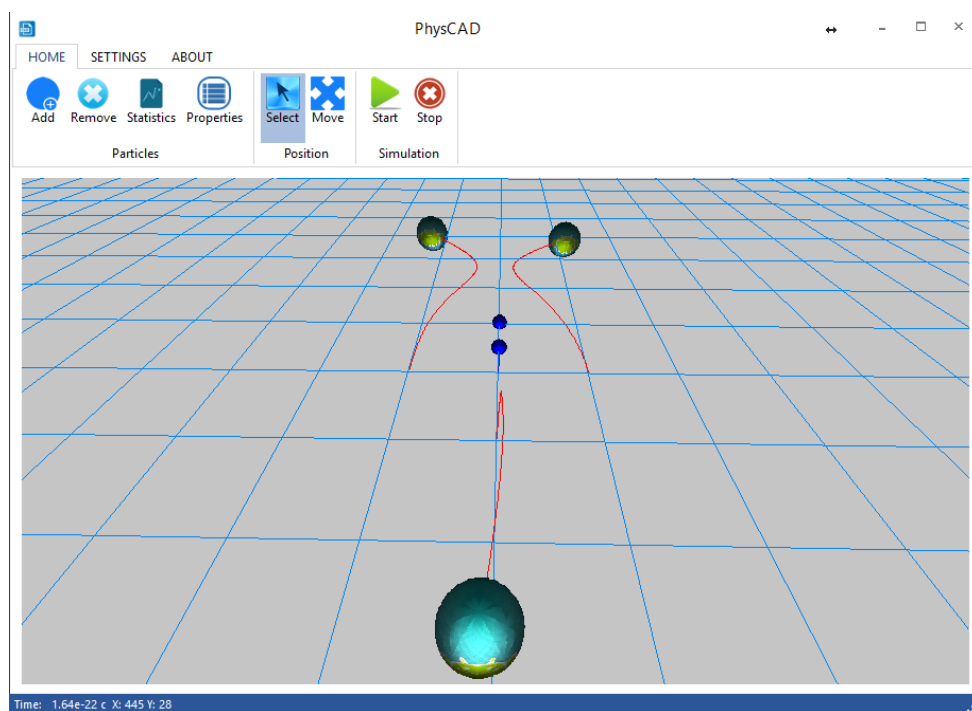


Рис. 9. Результат моделирования

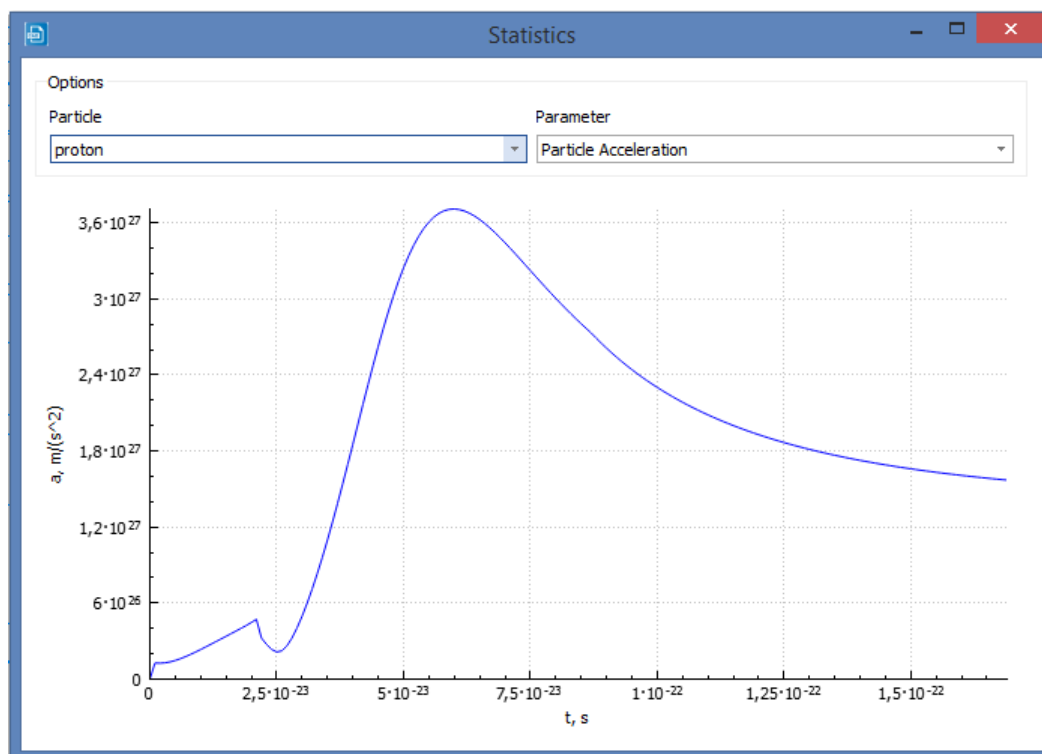


Рис. 10. График зависимости ускорения второго протона от времени

Заключение

1. Разработан физический движок, который моделирует движение частиц согласно математической модели молекулярной динамики.
2. Разработан графический движок, который выполняет рендеринг на CPU. Движок поддерживает: различные примитивы, освещение, монотонную заливку полигона, перспективно корректное текстурирование полигона, Z-буффер, FPS – камеру камера от первого лица, что позволяет перемещаться по сцене привычным для человека способом.
3. Разработан пользовательский интерфейс, который способен обработать ввод пользователя, не зависимо от того, насколько сложная сцена в данный момент рендерится. Это достигается за счет того, что процесс рендеринга, моделирования и обработки запросов пользователя выполняются в разных потоках. Для обеспечения стабильности данной системы была проделана работа, связанная с синхронизацией потоков с помощью системных средств синхронизации.
4. Разработанный графический движок рекомендуется использовать в задачах, где объекты, которые необходимо вывести на экран, можно аппроксимировать с помощью полигонов.

Список литературы

1. Суравикин А. Ю. Реализация метода SPH на CUDA // Наука и образование. МГТУ им. Н.Э. Баумана. Электрон. журн. 2012. № 7. Режим доступа: <http://technomag.bmstu.ru/doc/423582.html> (дата обращения 2.04.2015).
2. Дональд Херн, М. Паулин Бейкер Компьютерная графика и стандарт OpenGL. М.: Издательский дом «Вильямс», 2005. 1168 с.
3. Ричард С. Райт-мл., Бенджамин Липчак OpenGL. Суперкнига. М.: Издательский дом «Вильямс», 2006. 1039 с.
4. Боресков А., Харламов А. Основы работы с технологией CUDA. М.: ДМК Пресс, 2010. 232 с.