СОДЕРЖАНИЕ

ВВЕДЕНИЕ

С развитием компьютерных технологий компьютерная графика приобрела совершенно новый статус, поэтому сегодня она является основной технологией в цифровой фотографии, кино, видеоиграх, а также во многих специализированных приложениях. Было разработано большое количество алгоритмов отображения. Главными критериями, которые к ним предъявляются, являются реалистичность изображения и скорость отрисовки. Однако зачастую чем выше реалистичность, тем больше времени и памяти требуется для работы алгоритма.

Одним из направлений моделирования является моделирование движения частиц. Имеется огромная потребность в качественной и эффективной отрисовке распространения частиц вируса. Особенно эта тема стала актуальной после начала пандемии короновируса. Пандемия COVID-19 повлияла на жизнь миллионов людей по всему миру. Помимо серьезных последствий для здоровья, пандемия также изменила нашу повседневную жизнь, перевернула рынок вакансий и подорвала экономическую стабильность. В данном курсовом проекте речь пойдет о моделировании распространения частиц вирусной инфекции.

Цель данной курсовой работы — разработать программу с пользовательским интерфейсом, которая предоставит функционал для моделирования броуновского движения частиц короновирусной инфекции в помещении с учетом скорости их распространения и времени жизни на разных поверхностях.

Задачи, которые необходимо выполнить для достижения поставленной цели:

— изучить алгоритмы удаления невидимых линий и поверхностей и методы закраски;

— проанализировать алгоритмы моделирования броуновского движения;

— выбрать подходящие для решения поставленной задачи алгоритмы и реализовать их;

— формализовать модель и описать выбранные типы и структуры данных;

— выявить зависимость времени отрисовки кадра от количества частиц вируса, находящихся на сцене.

1 Аналитический раздел

В данном разделе представлено описание объектов сцены, а также обоснован выбор алгоритмов, которые будут использованы для ее визуализации.

1.1 Описание и формализация объектов сцены

1.1.1 Объекты сцены

Объекты сцены:

— абстрактная фигура человека;

— стены и пол;

— частицы вируса.

Стены и пол представляют собой параллелепипеды. Частицы вируса представлены в форме шаров.

1.1.2 Выбор формы представления трехмерных объектов

Обычно используются три формы задания моделей [1]:

— каркасная;

— поверхностная;

— объемная.

Каркасная модель — одна из простейших форм задания модели, так как заключается в хранении информации только о вершинах и ребрах объекта.

Поверхностная модель объекта — это оболочка объекта, пустая внутри. Такая информационная модель содержит данные только о внешних геометрических параметрах объекта. Данный тип модели часто используется в компьютерной графике. При этом могут использоваться различные типы поверхностей, ограничивающих объект, такие как полигональные модели, поверхности второго порядка и другие.

При объемном моделировании учитывается материал, из которого изготовлен объект.

Для решения поставленной задачи будет использована поверхностная модель, так как каркасные модели могут привести к неправильному восприятию формы объекта, а реализация объемной модели потребует большего количества ресурсов на отображение деталей, не влияющих на качество решения задачи в ее заданной формулировке.

В свою очередь поверхностная модель может задаваться параметрическим представлением или полигональной сеткой.

В случае полигональной сетки форма объекта задаётся некоторой совокупностью вершин, ребер и граней. Наиболее подходящим представлением сцены в условиях поставленной задачи будет представление в виде списка граней, так как оно позволяет проводить явный поиск вершин грани и самих граней, которые окружают вершину.

1.2 Выбор и анализ алгоритмов удаления невидимых ребер и поверхностей

1.2.1 Алгоритм обратной трассировки

Алгоритм обратной трассировки лучей отслеживает лучи в обратном направлении (от наблюдателя к объекту). Считается, что наблюдатель расположен на положительной полуоси z в бесконечности, поэтому все световые лучи параллельны оси z. В ходе работы испускаются лучи от наблюдателя и ищутся пересечения луча и всех объектов сцены. В результате, пересечение с максимальным значением z является видимой частью поверхности, и атрибуты данного объекта используются для определения характеристик пикселя, через центр которого проходит данный световой луч.

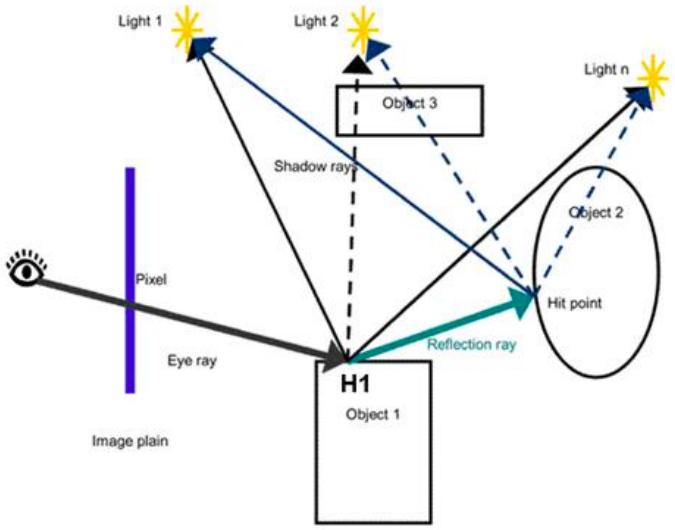

Рисунок 1.1 — Алгоритм обратной трассировки

Эффективность процедуры определения пересечений луча с поверхностью объекта оказывает самое большое влияние на эффективность всего алгоритма. Чтобы избавиться от ненужного поиска пересечений было придумано искать пересечение луча с объемной оболочкой рассматриваемого объекта. Под оболочкой понимается некоторый простой объект, внутрь которого можно поместить рассматриваемый объект, к примеру параллелепипед или сферу.

В дальнейшем при рассмотрении пересечения луча и объемной оболочкой рассматриваемого объекта, если такого пересечения нет, то и соответственно пересечения луча и

самого рассматриваемого объекта нет, и наоборот, пересечение найдено, то возможно, есть пересечение луча и рассматриваемого объекта.

Преимущества алгоритма:

— возможность использования алгоритма в параллельных вычислительных системах.

Недостаттки алгоритма:

— требуется большое количество вычислений;

— производительность алгоритма.

1.2.2 Алгоритм Робертса

Работа данного алгоритма проходит в 2 этапа:

— определение нелицевых граней для каждого тела отдельно;

— определение и удаление невидимых ребер [2].

Для определения, лежит ли точка в положительном подпространстве, используют проверку знака скалярного произведения (l, n), где l — вектор, направленный к наблюдателю, фактически определяет точку наблюдения; n — вектор внешней нормали грани. Если $(l, n) > 0$, т. е. угол между векторами острый, то грань является лицевой. Если $(l, n) < 0$, т. е. угол между векторами тупой, то грань является нелицевой. В алгоритме Робертса требуется, чтобы все изображаемые тела или объекты были выпуклыми. Невыпуклые тела должны быть разбиты на выпуклые части. В этом алгоритме выпуклое многогранное тело с плоскими гранями должно представиться набором пересекающихся плоскостей. Уравнение произвольной плоскости в трехмерном пространстве имеет вид 1.1.

$$ax + by + cz + d = 0 \tag{1.1}$$

В матричной форме 1.1 выглядит как 1.2.

$$[x\ y\ z\ 1][P]^T = 0 \tag{1.2}$$

В формуле 1.2 выражение $[P]^T = [a\ b\ c\ d]$ представляет собой плоскость. Поэтому любое выпуклое твердое тело можно выразить матрицей тела, состоящей из коэффициентов уравнений плоскостей, т. е.

$$M = \begin{bmatrix} a_1 & a_2 & ... & a_n \\ b_1 & b_2 & ... & b_n \\ c_1 & c_2 & ... & c_n \\ d_1 & d_2 & ... & d_n \end{bmatrix}, \tag{1.3}$$

где каждый столбец содержит коэффициенты одной плоскости.

Любая точка пространства может быть представлена в однородных координатах вектором $[S] = [x\ y\ z\ 1]$. Более того, если точка $[S]$ лежит на плоскости, то $[S] * [P]^T = 0$. Если же $[S]$ не лежит на плоскости, то знак этого скалярного произведения показывает, по какую сторону от плоскости расположена точка. В алгоритме Робертса предполагается, что точки, лежащие внутри тела, дают отрицательное скалярное произведение.

Преимущества алгоритма:

— высокая точность вычислений.

Недостатки алгоритма:

— рост числа трудоемкости алгоритма, как квадрата числа объектов;

— работа только с выпуклыми телами [2].

1.2.3 Алгоритм, использующий Z-буфер

В данном алгоритме рассматриваются два буфера. Буфер кадра используется для запоминания атрибутов (интенсивности) каждого пикселя в пространстве изображения, z-буфер — это отдельный буфер глубины, используемый для запоминания координаты z или глубины каждого видимого пикселя в пространстве изображения. В процессе работы глубина или значение z каждого нового пикселя, который нужно занести в буфер кадра, сравнивается с глубиной того пикселя, который уже занесен в z-буфер. Если это сравнение показывает, что новый пиксель расположен впереди пикселя, находящегося в буфере кадра, то новый пиксель заносится в этот буфер и, кроме того, производится корректировка z-буфера новым значением z. Если же сравнение дает противоположный результат, то никаких действий не производится. По сути, алгоритм является поиском по x и y наибольшего значения функции $z(x, y)$ [2].

Преимущества алгоритма:

— элементы сцены заносятся в буфер кадра в произвольном порядке, поэтому в данном алгоритме не тратится время на выполнение сортировок;

— произвольная сложность сцены;

— поскольку размеры изображения ограничены размером экрана дисплея, трудоемкость алгоритма зависит линейно от числа рассматриваемых поверхностей.

Недостатки алгоритма:

— трудоемкость устранения лестничного эффекта;

— большой объем требуемой памяти.

1.3 Анализ методов закрашивания

Методы закрашивания используются для затенения полигонов (или поверхностей, аппроксимированных полигонами) в условиях некоторой сцены, имеющей источники освещения.

Существует несколько основных методов закраски:

— простая закраска;

— закраска по Гуро, основанная на интерполяции значений интенсивности освещенности поверхности;

— закраска по Фонгу, основанная на интерполяции векторов нормалей к граням многогранника.

1.3.1 Простая закраска

При однотонной закраске вычисляют один уровень интенсивности, который используется для закраски всего многоугольника. При этом предполагается, что:

— источник света расположен в бесконечности;

— наблюдатель находится в бесконечности;

— многоугольник представляет реальную моделируемую поверхность, а не является аппроксимацией криволинейной поверхности.

Большим недостатком данной модели является то, что все точки грани будут иметь одинаковую интенсивность.

Рисунок 1.2 — Пример простой закраскии

1.3.2 Закраска по Гуро

Данный алгоритм предполагает следующие шаги:

— вычисляются нормали ко всем полигонам;

— определяются нормали в вершинах путем усреднения нормалей по всем полигональным граням, которым принадлежит вершина;

— используя нормали в вершинах и применяя произвольный метод закраски, вычисляются значения интенсивности в вершинах;

— каждый многоугольник закрашивается путем линейной интерполяции значений интенсивностей в вершинах сначала вдоль каждого ребра, а затем и между ребрами вдоль каждой сканирующей строки.

Достоинства:

— хорошо сочетается с диффузным отражением;

— изображение получается более реалистичным, чем при простой закраске.

Недостатки:

— данный метод интерполяции обеспечивает лишь непрерывность значений интенсивности вдоль границ многоугольников, но не обеспечивает непрерывность изменения интенсивности.

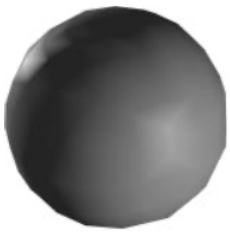

Рисунок 1.3 — Пример закраски по Гуро

1.3.3 Закраска по Фонгу

При такой закраске, в отличие от закраски по Гуро, вдоль сканирующей строки интерполируется значение вектора нормали, а не интенсивности.

Шаги алгоритма:

— вычисление векторов нормалей в каждой грани и к каждой вершине грани;

— интерполяция векторов нормалей вдоль ребер грани;

— линейная интерполяция векторов нормалей вдоль сканирующей строки;

— вычисление интенсивности в очередной точке сканирующей строки.

Достоинства:

— можно достичь лучшей локальной аппроксимации кривизны поверхности.

Недостатки:

— ресурсоемкость;

— вычислительная сложность.

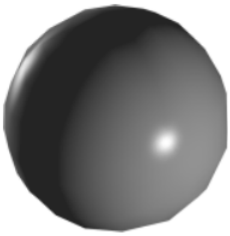

Рисунок 1.4 — Пример закраски по Фонгу

1.4 Алгоритмы моделирования броуновского движения

Броуновское движение — беспорядочное движение малых частиц, взвешенных в жидкости или газе, происходящее под действием молекул окружающей среды.

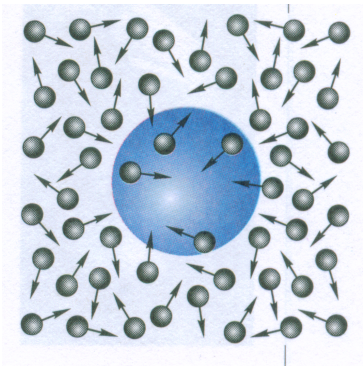

Рисунок 1.5 — Броуновское движение

Рассмотрим случайную величину $X(t)$, заданную на отрезке $[0,T]$. Случайный процесс $X(t)$ называется одномерным броуновским движением на интервале $[0,T]$, если он обладает следущими свойствами:

— $X(0) = 0$ почти наверное и $X(t)$ - почти наверное непрерывная функция на $[0,T]$;

— $X(t)$ - процесс с независимыми приращениями;

— $X(t)$ - процесс с приращениями, распределёнными нормально.

Для моделирования броуновского движения можно воспользоваться различными алгоритмами.

1.4.1 Классическое броуновское движение

Проще всего реализовать дискретную реализацию броуновского движения, рассмотрев последовательность $x_0 = 0$, $x_{n+1} = x_n + g_n$, где g_n - случайная величина, имеющая нормальное распределение (например, $N(0,1)$) [3].

1: $array[N]$
2: $array[0] \leftarrow 0$
3: **for** i = 1,..., N **do**
4: $\quad array[i+1] \leftarrow array[i] + randomNormal(0,1)$
5: **end for**

1.4.2 Алгоритм срединных смещений

Метод случайного срединного смещения более сложен, чем классический метод, однако используется для конструктивного доказательства существования броуновского движения, а также для построения фрактальной интерполяции (когда необходимо, чтобы кривая проходила через заданные точки интерполяции). Метод также может быть обобщен на случай n-мерных броуновских движений [3].

Алгоритм случайного срединного смещения вычисляет значения $X(t)$ в диадических рациональных точках вида $\frac{k}{2^n} \in [0,1]$. Последовательно вычисляются значения в середине отрезка [0,1], а затем в серединах отрезков $[0, \frac{1}{2}]$ и $[\frac{1}{2}, 1]$ и т.д. На каждом шаге итерации должен выполняться закон дисперсии для приращения в вычисленных точках. Параметр σ определяет масщтаб по вертикальной оси, не влияя на фрактальную размерность графика.

Вход: N, σ (N - число шагов алгоритма, при этом всего $2^N + 1$ точек интерполяции, σ - параметр вертикального масштаба, коэффициент дисперсии).

Выход: массив значений $\left\{X(\frac{k}{2^N})\right\}_{k=0}^{2^N}$ (реализация броуновского движения $X(t)$ на дискретном множестве точек вида $t_k = \frac{k}{2^N}$, k $\in [0, 2^N]$).

1: $X(0) \leftarrow 0$
2: $X(1) \leftarrow \sigma g$ // g - случайная величина, распределенная нормально с параметрами $N(0,1)$
3: **for** j = 1,..., N **do**
4: $\quad$ **for** i = 1,..., 2^{N-1} **do**
5: $\quad\quad X((2i-1)2^{N-j}) \leftarrow X((i-1)2^{N-j+1}) + X(i2^{N-j+1}) + \frac{1}{2^{(j+1)/2}}\sigma$ g
6: $\quad$ **end for**
7: **end for**

1.4.3 Фрактальное броуновское движение

Фрактальное броуновское движение (ФБД) уже не является марковским процессом, а обладает некоторой "памятью". Для апроксимации ФБД нет простого метода, вроде суммирования нормальных случайных величин, как в случае классического броуновского движения. Для апроксимации ФБД наиболее удобно использовать преобразования Фурье.

Рассмотрим случайную величину $X(t)$, заданную на отрезке $[0, T]$. *Случайный процесс* $X(t)$ называется одномерным фрактальным броуновским движением на интервале $[0, T]$, если он обладает следущими свойствами:

— $X(0) = 0$ почти наверное и $X(t)$ - почти наверное непрерывная функция на $[0, T]$;

— $X(t)$ - процесс с приращениями, распределенными нормально.

Теорема 1 *Если $X(t)$ — ФБД с параметром H, то его спектральная плотность*

$$S(f) \propto \frac{1}{f^{2H+1}} \tag{1.4}$$

Идея метода состоит в следующем. Строится преобразование Фурье для искомого ФБД в частной области, задавая случайные фазы и подбирая амплитуды, удовлетворяющие свойству из Теоремы 1. Затем получается ФБД во временной области с помощью обратного преобразования Фурье.

Необходимо смоделировать дискретный аналог ФБД, то есть цель — получить величины $\{X_n\}_{n=0}^{N-1}$, апроксимирующие ФБД в точках n. Для этого необходимо воспользоваться формулой дискретного преобразования Фурье

$$\hat{X_n} = \sum_{k=0}^{N-1} X_k e^{-2\pi kn/N} \tag{1.5}$$

и обратного дискретного преобразования Фурье

$$X_n = \sum_{k=0}^{N-1} \hat{X_k} e^{2\pi kn/N} \tag{1.6}$$

Далее будут рассмотрены только четные значения N, а для применения метода быстрого дискретного преобразования Фурье нужно, чтобы $N = 2^M$, $M \in \mathbb{N}$.

Вход: $H \in (0{,}1)$, $N = 2^M$, $M \in \mathbb{N}$ (H - параметр ФБД, размерность графика равна $d = 2 - H$, N - параметр, определяющий количество точек дискретизации ФБД).

Выход: массив значений $\{X_n\}_{n=0}^{N-1}$ (дискретная апроксимация ФБД в последовательные моменты времени n).

1: $\hat{X}_0 \leftarrow g$

2: **for** j = 1,..., N/2-1 **do**

3: $\hat{X}_j \leftarrow \frac{ge^{2\pi iu}}{j^{H+0.5}}$

4: **end for**

5: $\hat{X}_{N/2} \leftarrow \frac{g\cos(2\pi iu)}{(N/2)^{H+0.5}}$ // Здесь cos — вещественная часть комплексной экспоненты e

6: **for** j = N/2+1,..., N-1 **do**

7: $\hat{X}_j \leftarrow \overline{\hat{X}_{N-j}}$

8: **end for**

9: $X \leftarrow convert(\hat{X})$ // Вектор $X = \{X_0,...,X_{N-1}\}$ получается обратным дискретным преобразованием Фурье из вектора $\hat{X} = \left\{\hat{X}_0,...,\hat{X}_{N-1}\right\}$.

Вывод

В данном разделе было представлено описание объектов сцены, а также обоснован выбор алгоритмов, которые будут использованы для ее визуализации.

Для удаления невидимых линий и поверхностей выбран алгоритм Z-буфера, так как он позволяет отображать сцены произвольной сложности. Также была выбрана закраска по Гуро с локальной моделью освещения Ламберта, так как алгоритм достаточно быстр и хорошо сочетается с выбранным ранее z-буфером.

Для моделирования броуновского движения был выбран алгоритм срединных смещений, так как он позволяет реалистично изобразить броуновское движение, а также обобщается для случая n-мерных движений.

2 Конструкторский раздел

В данном разделе будут представлены схемы алгоритмов, выбранных для решения задачи, и диаграмма классов.

2.1 Алгоритм срединных смещений

Схема алгоритма срединных смещений изображена на рисунке 2.1.

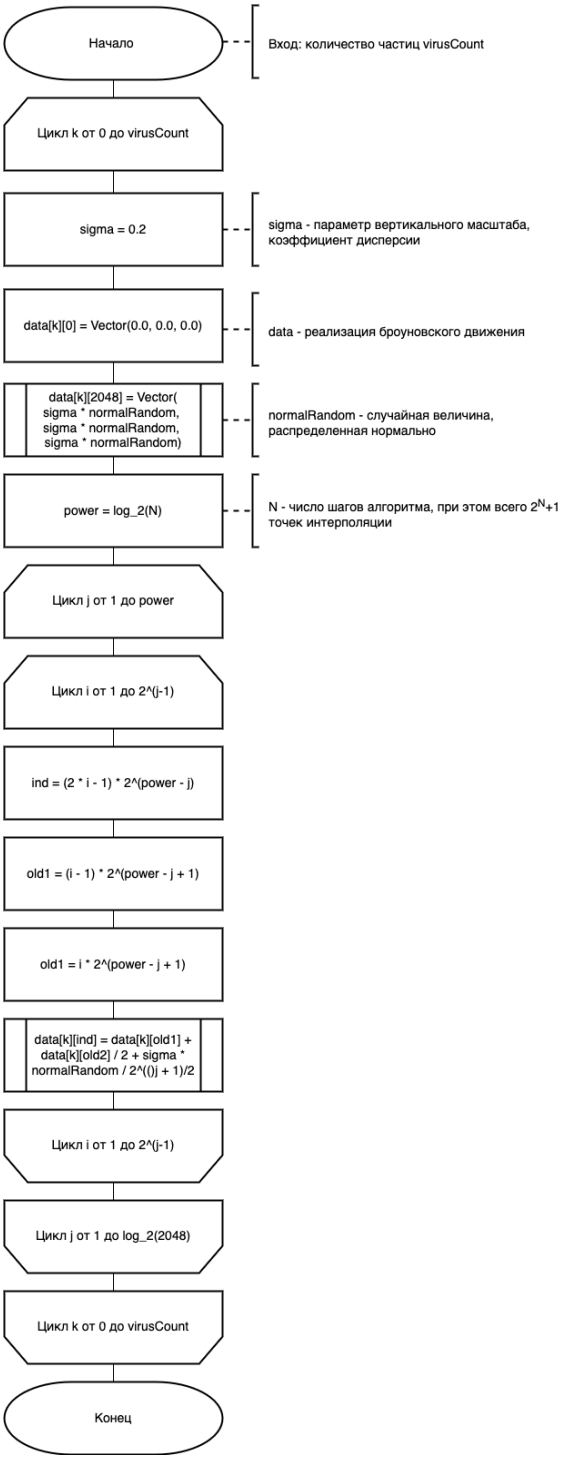

Рисунок 2.1 — Схема алгоритма срединных смещений

2.2 Алгоритмы отрисовки

Схема алгоритма Z-буфера изображена на рисунке 2.2.

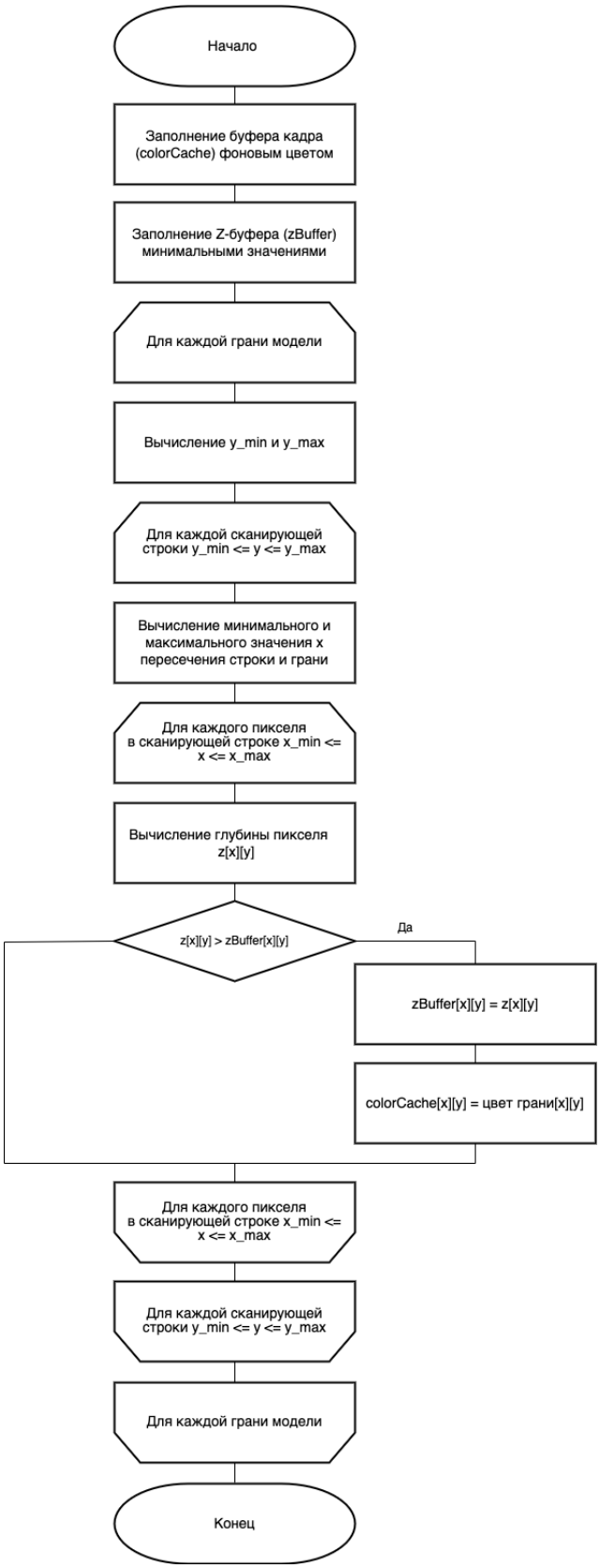

Рисунок 2.2 — Схема алгоритма Z-буфера

Схема алгоритма закраски по Гуро изображена на рисунке 2.3.

Рисунок 2.3 — Схема алгоритма закраски по Гуро

2.3 Диаграмма классов

На рисунке 2.4 представлена диаграмма классов.

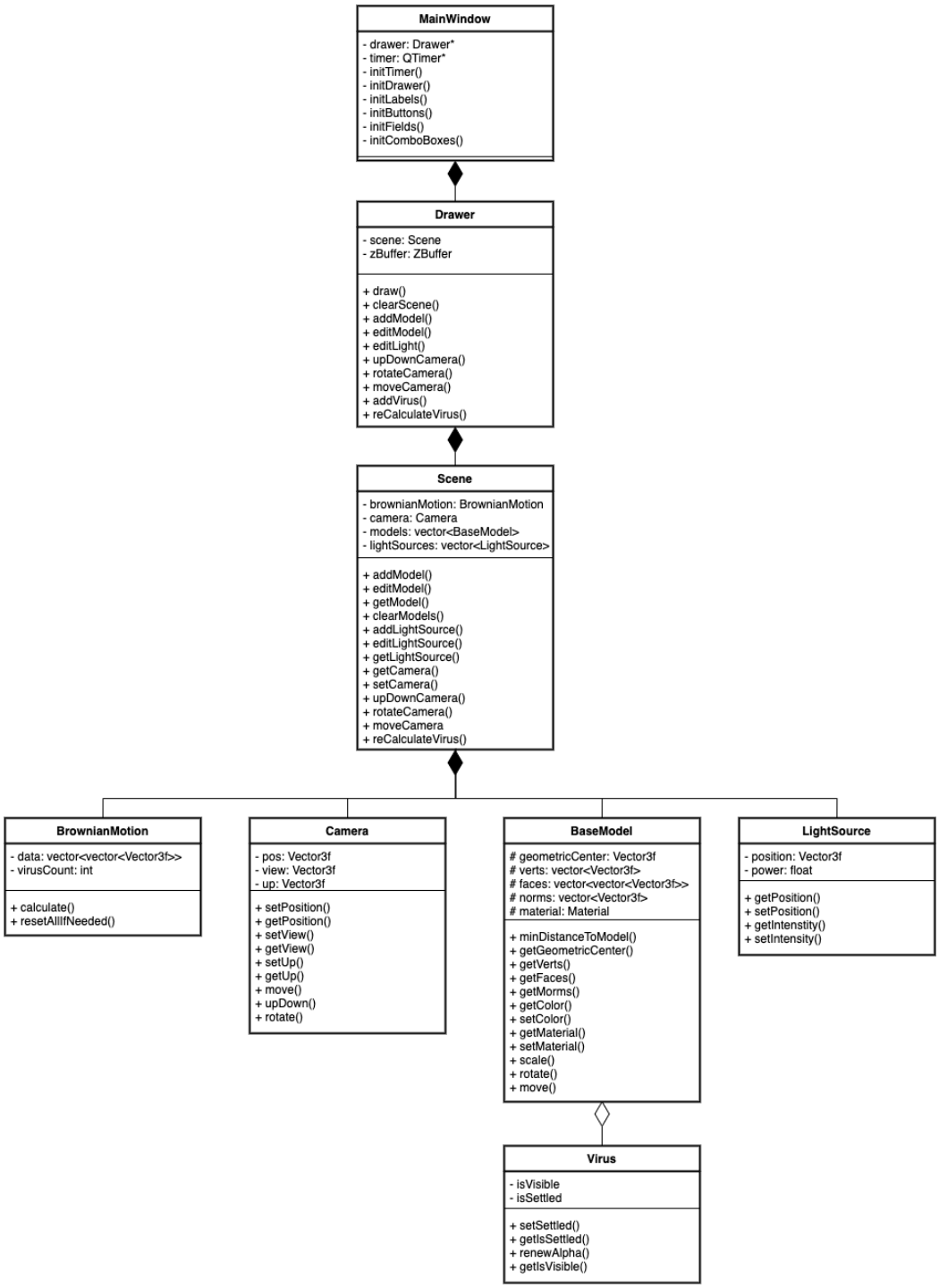

Рисунок 2.4 — Диаграмма классов

Вывод

В данном разделе были рассмотрены схемы алгоритмов, использованных при отрисовке сцены, а также диаграмма классов.

3 Технологический раздел

В данном разделе будут описаны средства реализации программного обеспечения, требования к нему, представлены реализации алгоритмов и рассмотрен интерфейс программы.

3.1 Требования к программному обеспечению

Программа должна предоставлять доступ к функционалу:

— возможность выбора материала покрытия пола и стен из предложенных вариантов (дерево, бумага(обои), керамика(плитка));

— изменение скорости движения частиц;

— изменение количества частиц инфекции;

— включение и выключение работы модели распространения частиц;

— вращение, перемещение и масштабирование модели.

3.2 Средства реализации

В качестве языка программирования для реализации курсовой работы использовался язык программирования C++ [4], так как он содержит возможности для создания программ с графическим интерфейсом. В качестве среды разработки использовался Qt Creator [5].

3.3 Реализация алгоритмов

В листинге 3.1 представлена реализация алгоритма моделирования броуновского движения.

Листинг 3.1 — Реализация вычисления изменения координат центров частиц (броуновское движение)

```
1  void BrownianMotion::setVirusCountAndResetAllIfNeeded(int newVirusCount)
2  {
3      if (virusCount == newVirusCount)
4      {
5          return;
6      }
7      virusCount = newVirusCount;
8      float sigma = 0.2;
9      data = std::vector<std::vector<Vector3f>>(virusCount,
           std::vector<Vector3f>(SIZE + 1));
10     for (int k = 0; k < virusCount; k++)
11     {
```

```
12          data.at(k).at(0) = Vector3f(0.0, 0.0, 0.0);
13          data.at(k).at(SIZE) = Vector3f(sigma * getNormalRandom(), sigma *
                getNormalRandom(), sigma * getNormalRandom());
14          for (int j = 1; j <= POWER; j++)
15          {
16              for (int i = 1; i <= pow(2, (j - 1)); i++)
17              {
18                  int ind = (2 * i - 1) * pow(2, POWER - j);
19                  int old1 = (i - 1) * pow(2, POWER - j + 1);
20                  int old2 = i * pow(2, POWER - j + 1);
21                  data.at(k).at(ind).x = (data.at(k).at(old1).x +
                        data.at(k).at(old2).x) / 2 + sigma * getNormalRandom() /
                        pow(2, (j + 1) / 2);
22                  data.at(k).at(ind).y = (data.at(k).at(old1).y +
                        data.at(k).at(old2).y) / 2 + sigma * getNormalRandom() /
                        pow(2, (j + 1) / 2);
23                  data.at(k).at(ind).z = (data.at(k).at(old1).z +
                        data.at(k).at(old2).z) / 2 + sigma * getNormalRandom() /
                        pow(2, (j + 1) / 2);
24              }
25          }
26      }
27  }
```

В листинге 3.2 представлена реализация вычисления интенсивности вершины от источников света на сцене.

Листинг 3.2 — Реализация вычисления интенсивности вершины

```
1  float Drawer::processLight(const Vector3f& vert, const Vector3f& norm)
2  {
3      float wholeIntensity = 0;
4      float intensity;
5
6      size_t lights = scene.getLightSourceCount();
7
8      for (size_t i = 0; i < lights; i++)
9      {
10         intensity = 0;
11         LightSourcePoint lsp = scene.getLightSource(i);
12
13         Vector3f lightDir = vert - lsp.getPosition();
14
15         intensity += lightDir * norm / pow(lightDir.norm(), 2.0);
16         intensity *= lsp.getIntensity() * LIGHT_REFLECT;
17
18         intensity = fmax(0.0, intensity);
19         intensity = fmin(1.0, intensity);
```

```
20
21          intensity = BG_LIGHT + intensity * (1 - BG_LIGHT);
22
23          wholeIntensity += intensity;
24      }
25
26      if (wholeIntensity == 0)
27      wholeIntensity = BG_LIGHT;
28      else
29      wholeIntensity /= lights;
30
31      return wholeIntensity;
32  }
```

В листинге 3.3 представлен алгоритм вычисления глубины каждой точки полигона.

Листинг 3.3 — Реализация вычисления глубины каждой точки полигона

```
 1  void Drawer::processPoligon(Vector3i& t0, Vector3i& t1, Vector3i& t2,
 2  const QColor& color, float& i0, float& i1, float& i2,
 3  float modelAlpha)
 4  {
 5      if (t0.y == t1.y && t0.y == t2.y)
 6      return;
 7
 8      if (t0.y > t1.y)
 9      {
10          std::swap(t0, t1);
11          std::swap(i0, i1);
12      }
13      if (t0.y > t2.y)
14      {
15          std::swap(t0, t2);
16          std::swap(i0, i2);
17      }
18      if (t1.y > t2.y)
19      {
20          std::swap(t1, t2);
21          std::swap(i1, i2);
22      }
23
24      int total_height = t2.y - t0.y;
25
26      for (int i = 0; i < total_height; i++)
27      {
28          bool second_half = i > t1.y - t0.y || t1.y == t0.y;
29          int segment_height = second_half ? t2.y - t1.y : t1.y - t0.y;
30
```

```
31          float alpha = (float)i / total_height;
32          float betta = (float)(i - (second_half ? t1.y - t0.y : 0)) /
                segment_height;
33
34          Vector3i A = t0 + Vector3f(t2 - t0) * alpha;
35          Vector3i B = second_half ? t1 + Vector3f(t2 - t1) * betta : t0 +
                Vector3f(t1 - t0) * betta;
36
37          float iA = i0 + (i2 - i0) * alpha;
38          float iB = second_half ? i1 + (i2 - i1) * betta : i0 + (i1 - i0) *
                betta;
39
40          if (A.x > B.x)
41          {
42              std::swap(A, B);
43              std::swap(iA, iB);
44          }
45
46          A.x = std::max(A.x, 0);
47          B.x = std::min(B.x, w);
48
49          for (int j = A.x; j <= B.x; j++)
50          {
51              float phi = B.x == A.x ? 1. : (float)(j - A.x) / (float)(B.x -
                    A.x);
52
53              Vector3i P = Vector3f(A) + Vector3f(B - A) * phi;
54              float iP = iA + (iB - iA) * phi;
55              if (P.x >= w || P.y >= h || P.x < 0 || P.y < 0) continue;
56
57              if (zBuffer.getDepth(P.x, P.y) < P.z)
58              {
59                  zBuffer.setDepth(P.x, P.y, P.z);
60                  QColor newColor = QColor(iColor(color.rgba(), iP));
61                  if (fabs(modelAlpha - 1.0) > EPS)
62                  {
63                      newColor = calculateNewColor(newColor,
                            colorCache[P.x][P.y], modelAlpha);
64                  }
65                  colorCache[P.x][P.y] = newColor;
66              }
67          }
68      }
69  }
```

3.4 Интерфейс работы программного обеспечения

На рисунке 3.1 изображена сцена без частиц вируса.

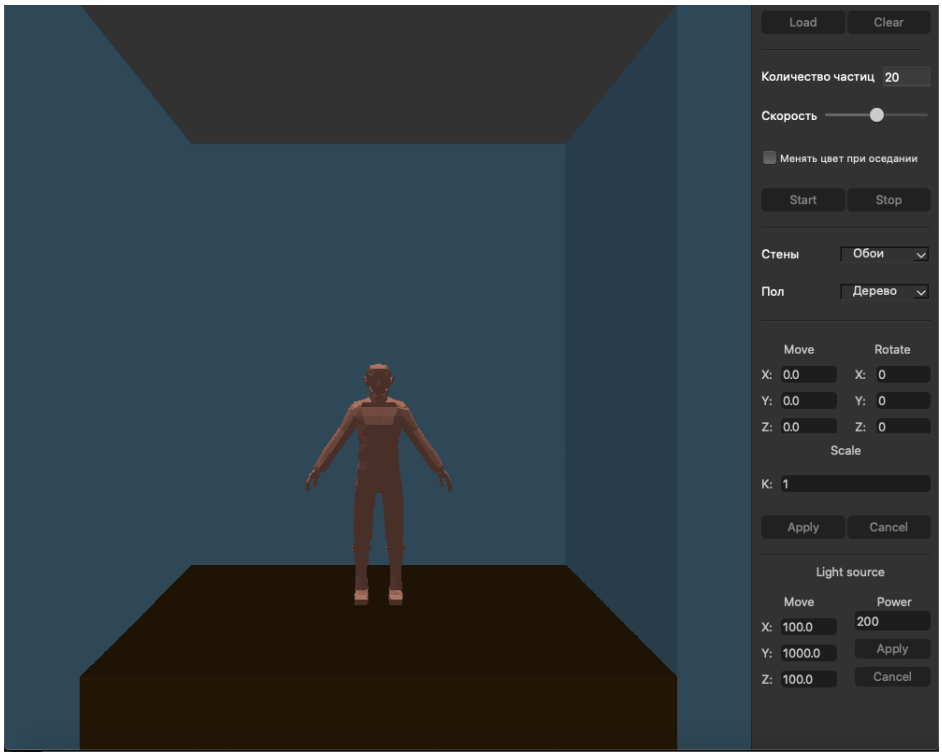

Рисунок 3.1 — Отображение сцены без частиц вируса

На рисунке 3.2 изображено движение частиц.

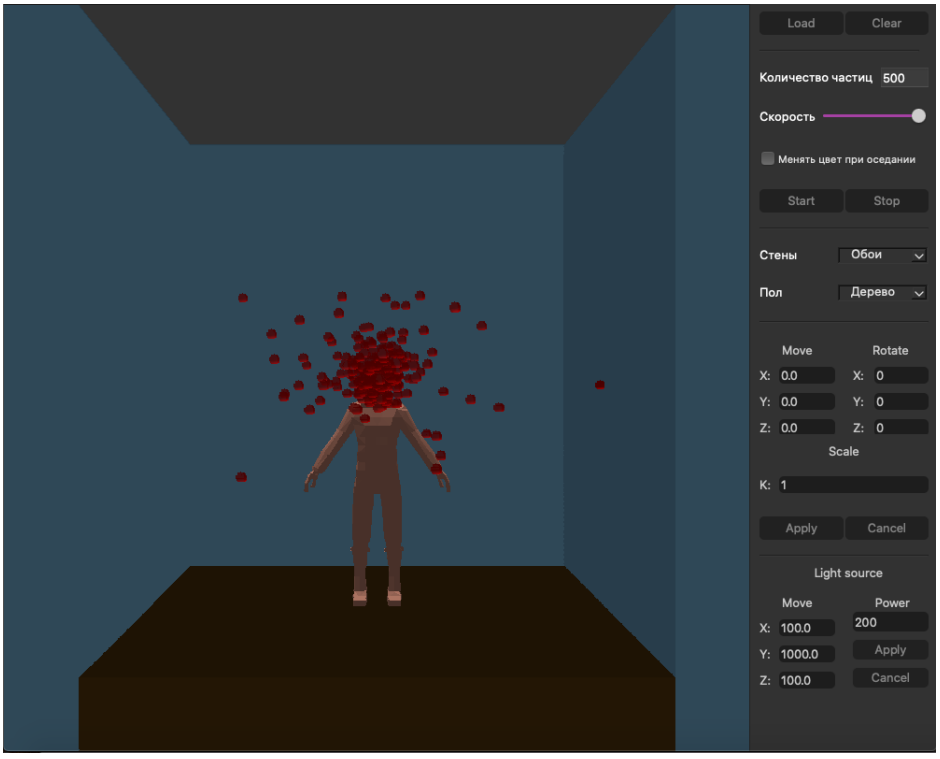

Рисунок 3.2 — Пример движения частиц

На рисунке 3.3 видно, как частицы осели на теле человека и стене и начали тускнеть.

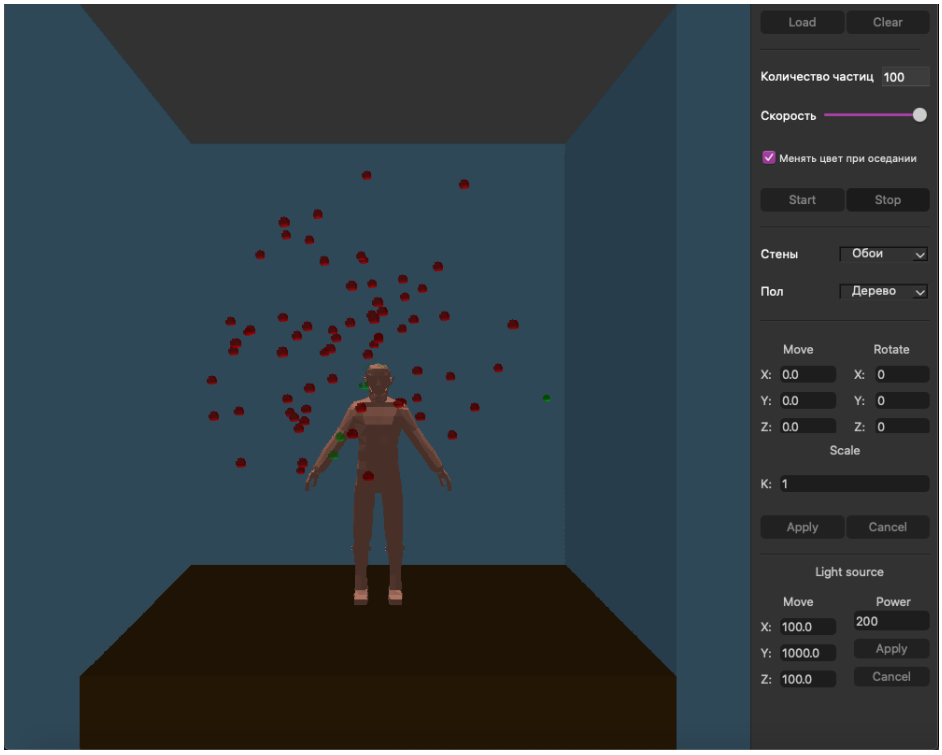

Рисунок 3.3 — Пример оседания и тускнения частиц

Вывод

В данном разделе были перечислены требования к программному обеспечению, средства разработки, с помощью которых была реализована программа, приведены реализации алгоритмов вычисления необходимых параметров для отрисовки сцены, а также реализация алгоритма моделирования броуновского движения.

4 Исследовательский раздел

В данном разделе будут приведены замеры времени работы реализации алгоритма отрисовки сцены. Все параметры замерялись на устройстве со следующими техническими характеристиками:

— операционная система macOS Catalina 10.15.46 [6];

— оперативная память 8 Гб;

— 1,8 ГГц 2-ядерный Intel Core i5 7 поколения [7].

Во время замеров ноутбук был подключен к сети питания и нагружен только приложениями, использующимися при замерах.

4.1 Цель эксперимента

Цель эксперимента — выявить зависимость времени отрисовки кадра от количества частиц вируса, находящихся на сцене.

4.2 План эксперимента

Измерения проводятся при одной скорости движения частиц. Каждый замер производится 30 раз, а затем время усредняется.

При замерах камера не будет передвигаться и поворачиваться, не будет изменяться положение объектов на сцене.

4.3 Результаты эксперимента

Процессорное время замерялось при помощи функции std::chrono::system_clock::now() из заголовочного файла chrono [4]. Результаты сформированы в виде графиков при помощи библиотеки matplotlib [8].

Результаты замеров представлены в таблице 4.1 и на рисунке 4.1.

На рисунке 4.1 на оси X указано количество частиц. Время на оси Y указано в миллисекундах.

Таблица 4.1 — Результаты замеров

Количество частиц вируса	Время отрисовки сцены (в миллисекундах)
1	235.0
10	255.0
20	265.0
40	295.0
60	330.0
100	390.0
150	460.0
200	550.0
250	620.0
300	680.0
400	830.0
500	1000.0
600	1130.0

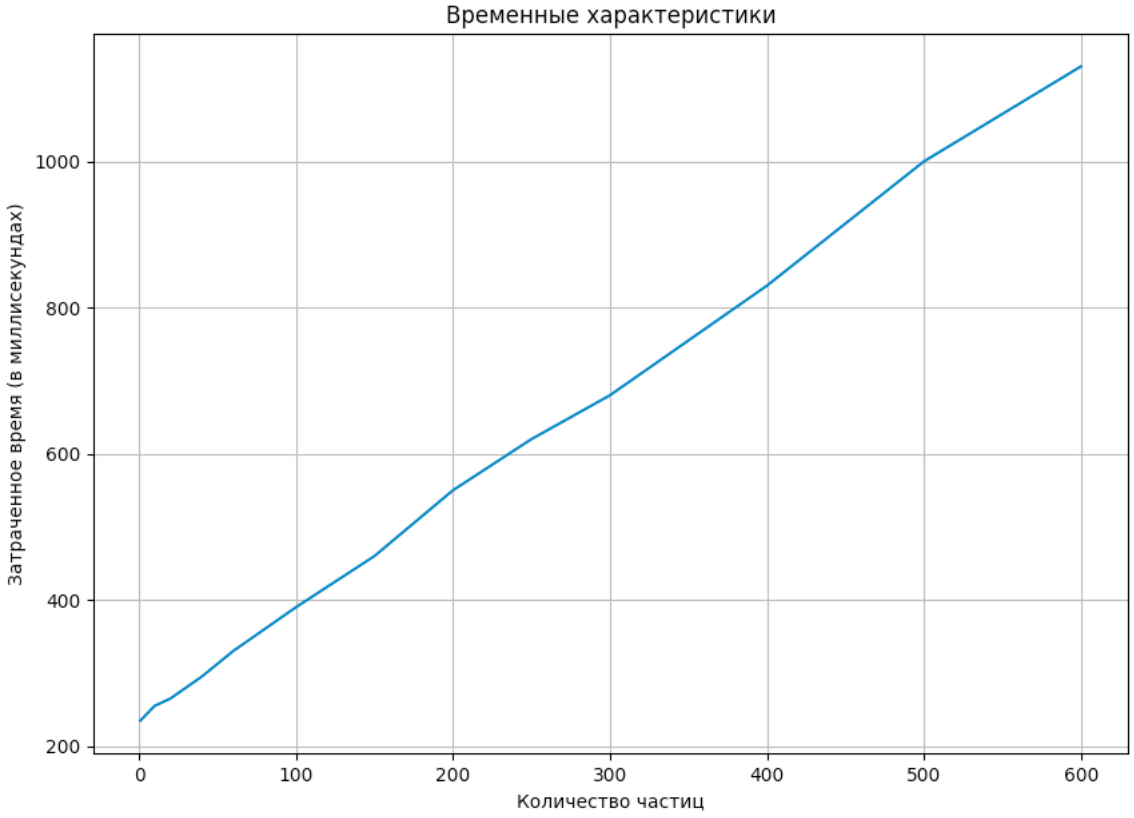

Рисунок 4.1 — Результаты замеров времени отрисовки кадра

Вывод

В результате эксперимента можно сделать вывод, что увеличение количества частиц вируса значительно влияет на скорость отрисовки изображения. Время, затрачиваемое на отрисовку сцены, линейно зависит от количества частиц.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была разработана программа с пользовательским интерфейсом, которая предоставляет функционал для моделирования броуновского движения частиц короновирусной инфекции в помещении с учетом скорости их распространения и времени жизни на разных поверхностях. Были решены следующие задачи:

— рассмотрены алгоритмы удаления невидимых линий и поверхностей и методы закраски;

— проанализированы алгоритмы моделирования броуновского движения;

— выбраны и реализованы подходящие для решения поставленной задачи алгоритмы;

— формализована модель, представлена диаграмма классов;

— выявлена зависимость времени отрисовки кадра от количества частиц вируса, находящихся на сцене.

В ходе замеров было выявлено, что скорость отрисовки сцены линейно зависит от количества частиц вируса.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Авдеева С.М. Куров А.В. Алгоритмы трехмерной машинной графики: учебное пособие. — Москва: Издательство МГТУ им. Н.Э. Баумана, 1996. — 60 с.

2. А.Ю. Демин. Основы компьютерной графики: учебное пособие. — Томск: Изд-во Томского политехнического университета, 2011. — 191 с.

3. П.И. Трошин. Моделирование фракталов в среде Maxima. — Казань: Изд-во Казанского федерального университета, 2012. — 48 с.

4. Документация языка C++ [Электронный ресурс]. — Режим доступа: `https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3690.pdf` (дата обращения: 13.11.2022).

5. Qt Creator [Электронный ресурс]. — Режим доступа: `https://www.qt.io/product/development-tools` (дата обращения: 26.11.2022).

6. macOS Catalina [Электронный ресурс]. — Режим доступа: `https://www.apple.com/macos/catalina/` (дата обращения: 30.09.2022).

7. Процессор Intel Core i5 7 поколения [Электронный ресурс]. — Режим доступа: `https://ark.intel.com/content/www/ru/ru/ark/products/97539/intel-core-i57260u-processor-4m-cache-up-to-3-40-ghz.html` (дата обращения: 13.11.2022).

8. Документация библиотеки matplotlib [Электронный ресурс]. — Режим доступа: `https://matplotlib.org` (дата обращения: 25.09.2022).