

Оглавление

Введение	5
1 Аналитическая часть	6
1.1 Анализ алгоритмов удаления невидимых линий и поверхностей	6
1.1.1 Алгоритм обратной трассировки	6
1.1.2 Алгоритм Робертса	7
1.1.3 Алгоритм, использующий Z-буфер	9
1.2 Анализ методов закрашивания	10
1.2.1 Простая закрашка	10
1.2.2 Закрашка по Гуро	11
1.2.3 Закрашка по Фонгу	12
2 Конструкторская часть	15
2.1 Понятие броуновского движения	15
2.2 Моделирование броуновского движения	16
2.2.1 Классическое броуновское движение	16
2.2.2 Алгоритм срединных смещений	17
2.2.3 Фрактальное броуновское движение	18
2.3 Формализация модели	22
2.4 Требования к программному обеспечению	23
3 Технологическая часть	24
3.1 Средства реализации	24
3.2 Выбор программного обеспечения	24
3.2.1 Swing	24
3.2.2 JavaFX	25
3.3 Описание используемых типов и структур данных	26
Заключение	28

Литература	29
----------------------	----

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 коронавируса. 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.

Цели данной работы - подготовить всю необходимую базу для реализации программного обеспечения, которое предоставляет возможность моделировать распространение частиц коронавирусной инфекции в помещении.

Задачи, которые необходимо выполнить для достижения поставленной цели:

- изучить алгоритмы удаления невидимых линий и поверхностей и методы закраски;
- проанализировать алгоритмы, моделирующие броуновское движение частиц;
- выбрать подходящие для решения поставленной задачи алгоритмы;
- выявить основные требования для программного обеспечения;
- формализовать модель и описать выбранные типы и структуры данных;
- выбрать язык программирования и среду разработки.

1 Аналитическая часть

1.1 Анализ алгоритмов удаления невидимых линий и поверхностей

1.1.1 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.

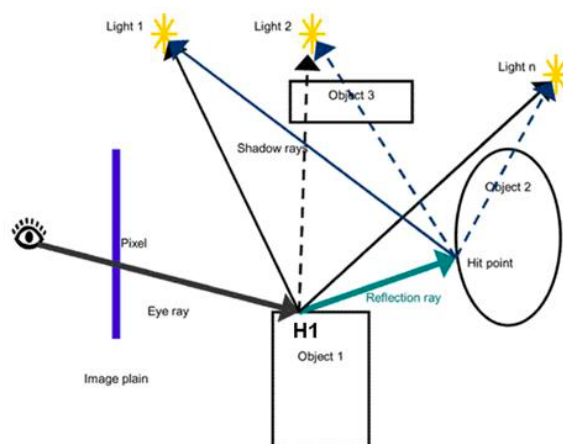


Рисунок 1.1 – Алгоритм обратной трассировки

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.

Для расчета эффектов освещения сцены проводятся вторичные лучи от точек пересечения ко всем источникам света. Если на пути этих лучей встречается непрозрачное тело, значит данная точка находится в тени, иначе он влияет на освещение данной точки. Также для получения более реалистичного изображения сцены, нужно учитывать вклады отраженных и преломленных лучей.

Плюсы:

- возможность использования алгоритма в параллельных вычислительных системах.

Минусы:

- требуется большое количество вычислений;
- производительность алгоритма.

1.1.2 Алгоритм Робертса

Алгоритм Робертса работает в объектном пространстве, кроме того работает только с выпуклыми телами. Если тело изначально является не выпуклым, то нужно его разбить на выпуклые составляющие.

Данный алгоритм состоит из следующих основных этапов:

- подготовка исходных данных;
- удаление линий, экранируемых самим телом;
- удаление линий, экранируемых другими телами.

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

$$ax + by + cz + d = 0 \quad (1.1)$$

В матричной форме этот результат выглядит так:

$$[x \ y \ z \ 1][P]^T = 0 \quad (1.2)$$

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

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

где каждый столбец содержит коэффициенты одной плоскости.

Любая точка пространства представима 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, т. е. нормали направлены наружу.

Плюсы алгоритма:

- высокая точность вычислений.

Минусы:

- рост числа трудоемкости алгоритма, как квадрата числа объектов;
- работа только с выпуклыми телами.

1.1.3 Алгоритм, использующий Z-буфер

Алгоритм Z-буфера решает задачу в пространстве изображений.

В данном алгоритме рассматривается два буфера. Буфер кадра (регенерации) используется для заполнения атрибутов (интенсивности) каждого пикселя в пространстве изображения. В Z-буфер (буфер глубины) можно помещать информацию о координате z для каждого пикселя.

Для начала необходимо подготовить буферы. Для этого в Z-буфер заносятся максимально возможные значения z , а буфер кадра заполняется значениями пикселя, который описывает фон. Также нужно каждый многоугольник преобразовать в растровую форму и записать в буфер кадра. Сам процесс работы заключается в сравнении глубины каждого нового пикселя, который нужно занести в буфер кадра, с глубиной того пикселя, который уже занесён в Z-буфер. В зависимости от сравнения принимается решение, нужно ли заносить новый пиксель в буфер кадра и, если нужно, также корректируется Z-буфер (в него нужно занести глубину нового пикселя).

Плюсы:

- 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.

Минусы:

- трудоемкость устранения лестничного эффекта;
- трудности реализации эффектов прозрачности;
- большой объем требуемой памяти.

Вывод

Для удаления невидимых линий и поверхностей выбран алгоритм Z-буфера, так как обладает важными преимуществами - высокой скоростью работы и произвольной сложностью сцены.

1.2 Анализ методов закрашивания

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

Существует несколько основных методов закрашки:

- простая закрашка;
- закрашка по Гуро, основанная на интерполяции значений интенсивности освещенности поверхности;
- закрашка по Фонгу, основанная на интерполяции векторов нормалей к граням многогранника.

1.2.1 Простая закрашка

Одной из самых простых моделей освещения является модель Ламберта. Она учитывает только идеальное диффузное отражение света от тела. Считается, что свет падающий в точку, одинаково рассеивается по всем направлениям полупространства. Таким образом, освещенность в точке определяется

только плотностью света в точке поверхности, а она линейно зависит от косинуса угла падения. При этом положение наблюдателя не имеет значения, т.к. диффузно отраженный свет рассеивается равномерно по всем направлениям.

Простая закраска используется при выполнении трех условий:

- предполагается, что источник находится в бесконечности;
- предполагается, что наблюдатель находится в бесконечности;
- закрашиваемая грань является реально существующей, а не полученной в результате аппроксимации поверхности.

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



Рисунок 1.2 – Пример простой закраски

1.2.2 Закраска по Гуро

Данный алгоритм предполагает следующие шаги:

- вычисление векторов нормалей к каждой грани;
- вычисление векторов нормали к каждой вершине грани путем усреднения нормалей к граням;
- вычисление интенсивности в вершинах грани;
- интерполяция интенсивности вдоль ребер грани;

- линейная интерполяция интенсивности вдоль сканирующей строки.

Плюсы:

- хорошо сочетается с диффузным отражением;
- изображение получается более реалистичным, чем при простой закраске.

Минусы:

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

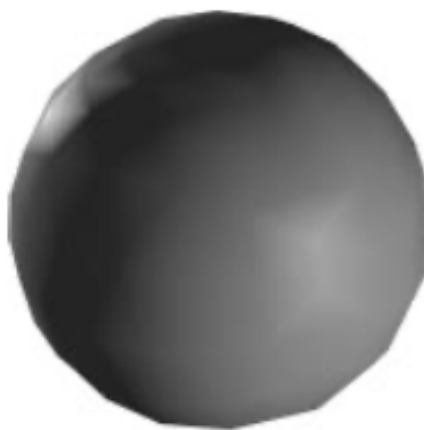


Рисунок 1.3 – Пример закраски по Гуро

1.2.3 Закраска по Фонгу

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

Шаги алгоритма:

- вычисление векторов нормалей в каждой грани.
- вычисление векторов нормали к каждой вершине грани.
- интерполяция векторов нормалей вдоль ребер грани.

- линейная интерполяция векторов нормалей вдоль сканирующей строки.
- вычисление интенсивности в очередной точке сканирующей строки.

Плюсы:

- можно достичь лучшей локальной аппроксимации кривизны поверхности.

Минусы:

- ресурсоемкость;
- вычислительная сложность.

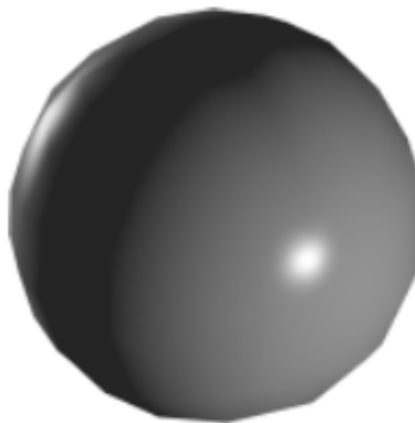


Рисунок 1.4 – Пример закраски по Фонгу

Вывод

Для закрашивания выбран алгоритм Фонга, так как данный алгоритм обладает важным преимуществом - высокой реалистичностью изображения.

Вывод

В данном разделе были формально описаны алгоритмы удаления невидимых линий и поверхностей, методы закрашивания поверхностей. В качестве

алгоритма удаления невидимых линий и поверхностей был выбран алгоритм Z-буфера, в качестве метода закрашивания был выбран алгоритм закрашки Фонга.

2 Конструкторская часть

2.1 Понятие броуновского движения

Броуновское движение (иногда называют Брауновское движение) – беспорядочное движение малых частиц, взвешенных в жидкости или газе, происходящее под действием молекул окружающей среды.

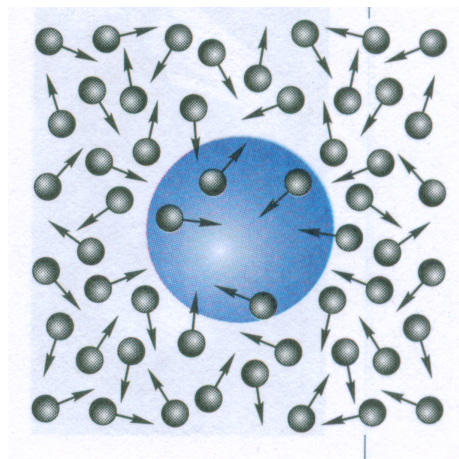


Рисунок 2.1 – Броуновское движение

Броуновское движение представляет собой пример естественного фрактала с фрактальной размерностью $d = 1.5$ (Мандельброт и Ван Несс, 1968). Впервые его наблюдал шотландский ботаник Роберт Броун в 1827 году: он заметил непрерывное беспорядочное движение взвешенных в жидкости маленьких частиц (пыльцы), но ошибочно приписал причину движения самим частицам. Только в 1905 году Альберт Эйнштейн и вслед за ним в 1906 году Мариан Смолуховский объяснили это движение хаотическими соударениями с молекулами окружающей среды. В 1908–1913 годах Жан Батист Перрен поставил ряд опытов, подтвердивших выводы Эйнштейна и Смолуховского. И, наконец, в 1923 год Норберт Винер построил первую математическую модель броуновского движения. Альтернативные подходы были предложены Андреем Николаевичем Колмогоровым в 1933 году и Полем Леви в 1948 году.

2.2 Моделирование броуновского движения

2.2.1 Классическое броуновское движение

Рассмотрим случайный процесс (случайную величину) $X(t)$, заданную на отрезке $[0, T]$.

Случайный процесс $X(t)$ называется одномерным броуновским движением (или винеровским процессом) на интервале $[0, T]$, если он обладает следующими 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.

Отметим следующие свойства броуновского движения:

- $X(t)$ почти наверное нигде не дифференцируем
- $X(t)$ - марковский процесс (не обладает памятью), т.е. если известна величина $X(t)$, то при $t_1 < t < t_2$ величины $X(t_1)$ и $X(t_2)$ независимы.
- Фрактальная размерность графика $X(t)$ равна 1.5
- Приращение $X(t)$ обладает свойством статистического самоподобия: для любого $r > 0$

$$X(t + \Delta t) = \frac{1}{\sqrt{r}}(X(t + r \Delta t) - X(t)) \quad (2.1)$$

- Стационарность приращений: дисперсия приращения зависит только от разности моментов времени

$$D(X(t_2) - X(t_1)) = \sigma^2 |t_2 - t_1| \quad (2.2)$$

- Математическое ожидание приращения равно

$$E(|X(t_2) - X(t_1)|) = \sqrt{\frac{2}{\pi}} \sigma \sqrt{|t_2 - t_1|} \quad (2.3)$$

Для моделирования броуновского движения можно воспользоваться разными алгоритмами. Рассмотрим 3 из них.

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, $N(0, 1)$).

```

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

2.2.2 Алгоритм срединных смещений

Метод случайного срединного смещения основан на работах Н. Виннера, он более сложен, чем метод из предыдущего параграфа, 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.

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 масштаб по вертикальной оси, не влияя на фрактальную размерность графика.

Броуновское движение методом срединного смещения

Вход: N, σ // N - число шагов алгоритма, при этом всего $2^N + 1$ точек интерполяции, σ - параметр вертикального масштаба, коэффициент диспер-

сии

Выход: массив значений $\{X(\frac{k}{2^N})\}_{k=0}^{2^N}$ // 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  $j = 1, \dots, N$  do
4:   for  $i = 1, \dots, 2^{N-1}$  do
5:      $X((2i-1)2^{N-j}) \leftarrow X((i-1)2^{N-j+1}) + X(i2^{N-j+1}) + \frac{1}{2^{(j+1)/2}} \sigma g$ 
6:   end for
7: end for

```

2.2.3 Фрактальное броуновское движение

Фрактальное броуновское движение (ФБД) уже не является марковским процессом, а обладает некой "памятью". Кроме того, вводя параметр $0 < H < 1$ можно получить одномерное ФБД размерности $d = 2 - H$ и двумерное ФБД размерности $d = 3 - H$. Заметим, что классическое броуновское движение получается как частный случай при $H = 0.5$. Для аппроксимации ФБД нет простого метода, вроде суммирования нормальных случайных величин, как в случае классического броуновского движения. Для аппроксимации ФБД наиболее удобно использовать преобразования Фурье.

Рассмотрим случайный процесс (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 следующими свойствами:

- $X(0) = 0$ почти наверное и $X(t)$ - почти наверное непрерывная функция на $[0, T]$
- $X(t)$ - процесс с приращениями, распределенными нормально

Отметим следующие свойства фрактального броуновского движения:

- $X(t)$ почти наверное нигде не дифференцируем

- Фрактальная размерность графика $X(t)$ равна $2 - H$
- Процесс $x(t)$ не обладает свойством независимости приращений
- Приращение $X(t)$ обладает свойством статистического самоподобия: для любого $r > 0$

$$X(t + \Delta t) = \frac{1}{\sqrt{r}}(X(t + r \Delta t) - X(t)) \quad (2.4)$$

- Стационарность приращений: дисперсия приращения зависит только от разности моментов времени

$$D(X(t_2) - X(t_1)) = \sigma^2 |t_2 - t_1|^{2H} \quad (2.5)$$

- Математическое ожидание приращения равно

$$E(|X(t_2) - X(t_1)|) = \sqrt{\frac{2}{\pi}} \sigma |t_2 - t_1|^H \quad (2.6)$$

Метод Фурье-фильтрации для построения ФБД

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

$$S(f) \propto \frac{1}{f^{2H+1}} \quad (2.7)$$

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 получаем ФБД во временной области с помощью обратного преобразования Фурье.

Будем моделировать дискретный аналог ФБД, то есть наша цель - получить величины $\{X_n\}_{n=0}^{N-1}$, аппроксимирующие ФБД в точках n . Воспользуемся формулой дискретного преобразования Фурье

$$\hat{X}_n = \sum_{k=0}^{N-1} X_k e^{-2\pi k n / N} \quad (2.8)$$

и обратного дискретного преобразования Фурье

$$X_n = \sum_{k=0}^{N-1} \hat{X}_k e^{2\pi kn/N} \quad (2.9)$$

Далее будем рассматривать только четные значения N , а для применения метода *быстрого дискретного преобразования Фурье* нужно, чтобы $N = 2^M$, $M \in \mathbb{N}$. Метод быстрого дискретного преобразования Фурье реализован во многих системах компьютерной алгебры. Он позволяет сократить вычисления в $\frac{2N}{\log_2 N}$ раз.

Для того, чтобы получающиеся величины X_n были вещественными мы наложим условие сопряженной симметрии:

$$\hat{X}_0, \hat{X}_{N/2} \in \mathbb{R}, \hat{X}_n = \hat{X}_{N-n}, n = 1, \dots, N/2 - 1 \quad (2.10)$$

Фильтрация относится к той части моделирования, когда мы заставляем коэффициенты преобразования Фурье удовлетворять степенному закону из Теоремы 1:

$$|\hat{X}_n|^2 \propto \frac{1}{n^{2H+1}}, n = 1, \dots, N/2 \quad (2.11)$$

Для этого возьмем

$$\hat{X}_n = \frac{ge^{2\pi iu}}{n^{H+0.5}} \quad (2.12)$$

где g - независимые значения нормально распределенной случайной величины с параметрами $N(0, 1)$, а u - независимые значения равномерно распределенной на отрезке $[0, 1]$ случайной величины. Оставшиеся коэффициенты вычислим из соотношений 1.15.

Для вычисления искомой аппроксимации ФБД $\{X_n\}_{n=0}^{N-1}$ применим обратное дискретное преобразование Фурье к набору $\{\hat{X}_n\}_{n=0}^{N-1}$.

Кривая ФБД методом Фурье-фильтрации

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 аппроксимация ФБД в

последовательные моменты времени n .

```

1:  $\hat{X}_0 \leftarrow g$ 
2: for  $j = 1, \dots, N/2-1$  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  $j = N/2+1, \dots, N-1$  do
7:    $\hat{X}_j \leftarrow \hat{X}_{N-j}$ 
8: end for
9:  $X \leftarrow \text{convert}(\hat{X})$  // Вектор  $X = \{X_0, \dots, X_{N-1}\}$  получается обратным дискретным преобразованием Фурье из вектора  $\hat{X} = \{\hat{X}_0, \dots, \hat{X}_{N-1}\}$ .

```

Для построения аппроксимации двумерного фрактального броуновского движения методом Фурье-фильтрации используются те же идеи, что и в одномерном случае. Вместо \hat{X}_n используется $\hat{X}_{k,j}$, $k, j = \overline{0, N-1}$, условие Теоремы 1 примет вид:

$$|\hat{X}_{k,j}|^2 \propto \frac{1}{(k^2 + j^2)^{H+1}}, n, k = 1, \dots, N/2 \quad (2.13)$$

МЫ ВОЗЬМЕМ

$$\hat{X}_{k,j} = \frac{ge^{2\pi iu}}{(k^2 + j^2)^{H/2+0.5}}, n, k = 1, \dots, N/2 \quad (2.14)$$

Запишем обратное дискретное преобразование Фурье: для $m, n = \overline{0, N-1}$

$$\begin{aligned}
\hat{X}_{m,n} &= \sum_{k=0}^{N-1} \sum_{j=0}^{N-1} \hat{X}_{k,j} e^{-2\pi i \frac{kn+jm}{N}} = \hat{X}_{0,0} + \sum_{k=1}^{N-1} \hat{X}_{k,0} e^{-2\pi i \frac{kn}{N}} + \sum_{j=1}^{N-1} \hat{X}_{0,j} e^{-2\pi i \frac{jm}{N}} + \\
&\quad \sum_{k=1}^{N/2} \sum_{j=1}^{N/2} \hat{X}_{k,j} e^{-2\pi i \frac{kn+jm}{N}} + \sum_{k=\frac{N}{2}+1}^{N-1} \sum_{j=\frac{N}{2}+1}^{N-1} (...) + \sum_{k=1}^{N/2} \sum_{j=\frac{N}{2}+1}^{N-1} (...) + \sum_{k=\frac{N}{2}+1}^{N-1} \sum_{j=1}^{N/2} (...)
\end{aligned} \quad (2.15)$$

Из формулы (1.19) следует, что для вещественности всех величин $X_{m,n}$ достаточно выполнения следующих условий сопряженной симметрии:

$$\hat{X}_{N-k,N-j} = \overline{\hat{X}_{k,j}} \quad k, j = \overline{1, N/2} \quad \hat{X}_{N/2,N/2} \in \mathbb{R} \quad (2.16)$$

$$\hat{X}_{k,N-j} = \overline{\hat{X}_{N-k,j}} \quad k, j = \overline{1, N/2 - 1} \quad \hat{X}_{0,0} \in \mathbb{R} \quad (2.17)$$

$$\hat{X}_{0,N-j} = \overline{\hat{X}_{0,j}} \quad j = \overline{1, N/2} \quad \hat{X}_{0,N/2} \in \mathbb{R} \quad (2.18)$$

$$\hat{X}_{N-k,0} = \overline{\hat{X}_{k,0}} \quad k = \overline{1, N/2} \quad \hat{X}_{N/2,0} \in \mathbb{R} \quad (2.19)$$

Условия (1.22)-(1.23) обеспечивают вещественность первых двух сумм, а условия (1.20)-(1.21) - оставшихся четырех сумм.

Вывод

Алгоритм Фурье-фильтрации содержит большое количество сложных вычислений, которые отрицательно влияют на скорость работы программы. Однако он позволяет наиболее реалистично изобразить броуновское движение, а также легко обобщается для случая n -мерных движений. Поэтому для реализации поставленной задачи был выбран именно этот алгоритм.

2.3 Формализация модели

Модель броуновского движения частиц будет задаваться такими характеристиками, как:

- скорость распространения – число типа *int*;
- количество частиц – число типа *int*.

Также на сцене будет изображено пустое помещение и абстрактная фигура человека в его центре. Пользователь должен уметь задавать материал покрытия стен:

- дерево;
- бумага (обои).

и пола:

- дерево (паркет);
- керамика (плитка).

2.4 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;
- изменение количества частиц инфекции;
- включение и выключение работы модели распространения частиц;
- вращение, перемещение и масштабирование модели.

Требования, которые предъявляются к программе:

- время отклика программы должно быть менее 1 секунды для корректной работы в интерактивном режиме;
- программа должна корректно реагировать на любое действие пользователя.

Вывод

В данном разделе были рассмотрены основные алгоритмы для реализации поставленной задачи, т.е. моделирования броуновского движения частиц. В качестве основного алгоритма был выбран метод Фурье-фильтрации. Также была формализована модель и определены требования, которые выдвигаются к программному продукту.

3 Технологическая часть

3.1 Средства реализации

При написании программного продукта был выбран язык *Java*. Это обусловлено следующими факторами:

- объектно-ориентированный язык, что позволяет использовать структуру классов;
- имеются необходимые библиотеки для реализации поставленной задачи;
- существует много учебной литературы.

В качестве среды разработки был выбран *IntelliJ IDEA*. *IntelliJ IDEA* – это интеллектуальная IDE, учитывающая контекст. Она предназначена для разработки разнообразных приложений на Java и других языках JVM, например Kotlin, Scala и Groovy. Также она поддерживает Git.

3.2 Выбор программного обеспечения

В языке Java есть несколько основных инструментов для создания пользовательских изображений. Самыми популярными из них являются JavaFX и Swing.

3.2.1 Swing

Swing – библиотека для создания графического интерфейса для программ на языке Java. Swing был разработан компанией Sun Microsystems. Он содержит ряд графических компонентов, таких как кнопки, поля ввода, таблицы и т. д.

Преимущества:

- Кроссплатформенность;

- Компоненты Swing следуют парадигме Model-View-Controller (MVC) и, таким образом, могут обеспечить гораздо более гибкий пользовательский интерфейс;
- Swing обеспечивает встроенную двойную буферизацию.

Недостатки:

- достаточно узкий спектр возможностей при работе с ui.
- считается устаревшей библиотекой.

3.2.2 JavaFX

JavaFX – платформа на основе Java для создания приложений с насыщенным графическим интерфейсом. Может использоваться как для создания настольных приложений, запускаемых непосредственно из-под операционных систем, так и для интернет-приложений, работающих в браузерах, и для приложений на мобильных устройствах.

JavaFX предназначен для предоставления приложениям таких сложных функций графического интерфейса, как плавная анимация, просмотр веб-страниц, воспроизведение аудио и видео, а также использование CSS стилей.

Преимущества:

- кроссплатформенность;
- больше встроенных возможностей;
- меньше кода.

Недостатки:

- технология еще молодая и "незрелая";
- в значительной степени зависит от огромной инфраструктуры, которая окружает Java.

Вывод

Уже более 10 лет разработчики приложений считают Swing высокоэффективным инструментарием для создания графических пользовательских интерфейсов (GUI) и добавления интерактивности в Java-приложения. Однако некоторые из самых популярных на сегодняшний день функций графического интерфейса не могут быть легко реализованы с помощью Swing в отличие от JavaFX.

Также можно писать программы на JavaFX, используя гораздо меньше кода, потому что JavaFX выполняет за нас всю работу. Не нужно регистрировать event listeners, и это делает тело функций более кратким. Кроме того, с помощью механизма привязки JavaFX легко интегрировать компоненты графического интерфейса с базовой моделью. Основываясь на вышесказанном в качестве библиотеки для работы с GUI была выбрана JavaFX.

3.3 Описание используемых типов и структур данных

Для реализации частиц используются следующие типы и структуры данных:

- *VirusParticle* – класс, хранящий всю информацию о частице.

Point – класс для работы с координатами частицы;

EnvironmentType – перечисление сред, в которых может находиться частица в данный момент (для отслеживания времени жизни вирусной частицы).

Листинг 3.1 – Структуры данных для хранения информации о частицах вируса

```
1 class VirusParticle {  
2     Point[] arrayName;  
3     EnvironmentType environmentType;  
4     int intensity;
```



```

5 }
6
7 enum EnvironmentType {
8     WOOD, PAPER, PLASTIC, AIR
9 }
10
11 class Point {
12     int x;
13     int y;
14     int z;
15 }

```

Для реализации объектов сцены используются следующие типы и структуры данных:

- *Box* – отображение стен и пола в помещении;
- *Person* – отображение абстрактной фигуры человека;
- *Texture* – обеспечивает загрузку из файла текстуры, ее интерпретацию на простую поверхность;
- *Scene* – характеризует набор объектов и их свойств

Вывод

В данном разделе были рассмотрены средства, с помощью которых было решено реализовывать ПО, технологии, которые будут использованы при его реализации. В качестве языка была выбрана *Java*. Для реализации пользовательского интерфейса выбрана библиотека *JavaFX*. Среда разработки - *IntelliJ IDEA*. Также были описаны используемые типы и структуры данных.

Заключение

Проделанная работа помогла закрепить полученные навыки в области компьютерной графики и проектирования программного обеспечения.

В процессе выполнения данной работы были выполнены следующие задачи:

- анализ алгоритмов удаления невидимых линий и поверхностей;
- анализ методов закрашивания;
- анализ алгоритмов моделирования броуновского движения;
- выбор подходящих для решения поставленной задачи алгоритмов;
- выявление основных требований для программного обеспечения;
- формализация модели и описание выбранных типов и структур данных;
- погружение в возможности языка *Java*;
- знакомство с библиотекой *JavaFX* и ее изучение.

Литература

- [1] Авдеева С.М., Куров А.В. Алгоритмы трехмерной машинной графики: учебное пособие. - М.: Издательство МГТУ им. Н.Э. Баумана, 1996. - 60 с., ил.
- [2] Давыдов А.В., Ерофеева Е.А. Графический пользовательский интерфейс на Java / А. В. Давыдов, Е. А. Ерофеева. – Евразийский научный журнал. 2016. № 6. 265-267 с.
- [3] Дёмин А.Ю., Основы компьютерной графики: учебное пособие – Томск: Изд-во Томского политехнического университета, 2011. – 191 с.
- [4] Кроновер, Р. М. Фракталы и хаос в динамических системах. Основы теории / Р. М. Кроновер. – М. : Постмаркет, 2000. – 352 с.
- [5] Сухов К. JavaFX – Reach internet application от Sun прощай, унылый Swing? / К. Сухов. – Системный администратор. 2009. № 4 (77). 67-73 с.
- [6] Barnsley, M. F. Superfractals / M. F. Barnsley. – Cambridge : Cambridge University Press, 2006. – 453 p.
- [7] Jackson, E. A. Perspectives of Nonlinear Dynamics / E. A. Jackson. – New York : Cambridge University Press, 2008. – 496 p.
- [8] Karatzas, I. Brownian Motion and Stochastic Calculus (secon edition) / I. Karatzas, S. E. Shreve. – New York ; Berlin : Springer-Verlag, 1991. – 493 p.
- [9] Mandelbrot, B. B. Fractional brownian motions, fractional noises and applications / B. B. Mandelbrot, J. W. V. Ness // SIAM Review. – 1968. – October. – Vol. 10, no. 4. – 422–437 p.