Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Визуализация реалистичного дождя в разное время суток»

Студент ИУ7-53Б _____ В. Марченко
(Группа) (Подпись, дата) (И. О. Фамилия)

Руководитель курсовой работы _____ А. С. Кострицкий
(Подпись, дата) (И. О. Фамилия)

2022 г.

Оглавление

Введение

Атмосферные эффекты — такие как дождь, туман, огонь, дым, облака, свечение звезд и снег — важны для создания реалистичной среды в интерактивных приложениях (обучающих системах, играх, фильмах и т. п.). Поскольку скорость вычислений современных графических аппаратных средств увеличивается, для погружения пользователя в визуально реалистичную среду также требуется и высокая степень реализма изображения. Однако визуализация атмосферных осадков, особенно в реальном времени, является сложной задачей [1, с. 1].

Дождь представляет собой очень сложное атмосферное физическое явление и состоит из многочисленных эффектов. Дождь может быть небольшим, умеренным, ситным, полосовым или грозовым. На улицах образуются лужи и разбрызгиваются капли. Можно увидеть рябь, падающие дождевые капли, стекающие с поверхностей предметов и т. д. [2]

Визуальные эффекты дождя содержат сложные физические механизмы, отражающие физические, оптические и статистические характеристики капель. Кроме того, капли дождя претерпевают сильные искажения формы при падении, называемые колебаниями. Из-за колебаний отражение и преломление света через падающую каплю дождя создают сложные картины яркости в пределах одной размытой в движении полосы дождя, снятой камерой или наблюдаемой человеком. Яркостная картина полосы дождя обычно включает в себя крапинки, множественные размытые блики и изогнутые контуры яркости [3].

Целью курсовой работы является реализация программного обеспечения для визуализации дождя в реальном времени с возможностью изменения с помощью графического интерфейса таких характеристик, как плотность дождя, размер капель, скорость падения дождя и направление падения дождевых капель. Также должна быть реализована возможность изменения положения камеры и источника света по осям X и Y и смена дня и ночи.

Задачами данной работы являются:

1) выбор способа представления объектов на сцене;

2) выбор модели дождевых капель;

3) анализ алгоритмов удаления невидимых линий и поверхностей и выбор наиболее подходящего;

4) анализ алгоритмов создания реалистичного освещения, отражений и теней и выбор наиболее подходящего;

5) анализ и выбор средств программной реализации;

6) реализация выбранных алгоритмов для создания программы визуализации дождя в реальном времени;

7) создание графического интерфейса для возможности изменения характеристик дождя и сцены пользователем.

1 Аналитическая часть

1.1 Описание объектов сцены

Сцена должна состоять из нескольких объектов.

Источник света (точечный) — материальная точка, излучающая свет во всех направлениях, причем интенсивность света уменьшается с расстоянием. Положение источника света задается тремя координатами (x, y, z) относительно начала координат.

Время суток. Оно не имеет собственной модели, однако оказывает влияние на восприятие человеком сцены. В программе должны быть реализованы день и ночь. День соответствует яркой и светлой цветовой гамме, а ночь — темной и приглушенной.

Дождевые капли — основной объект сцены, так как главной целью данной работы является визуализация дождя. Модель дождевой капли описана в аналитической части в пункте 1.4.

Земля — параллелепипед зеленого цвета внизу сцены, куда должны падать капли дождя.

1.2 Способы задания моделей

Существует три способа задания моделей:

1) каркасная модель;

2) поверхностная модель;

3) объемная твердотельная модель [4].

Каркасная модель представляет форму деталей в виде конечного множества линий. Для каждой линии известны координаты концевых точек и функция линии [4].

Поверхностная модель представляет форму деталей с помощью ограничивающих ее поверхностей (данные о гранях, вершинах, ребрах и функции поверхностей) [4].

Объемные твердотельные модели дополнительно содержат в явной форме сведения о принадлежности элементов внутреннему или внешнему по отношению к детали пространству [4].

Каркасная модель для визуализации дождя в реальном времени не подходит, так как капли дождя (как и другие осадки) сложно воспринимаются человеком в таком виде, а совокупность подобных моделей превращается в неразборчивую сцену. Так как цель курсовой работы не предполагает взаимодействия с объектами на сцене, объемные твердотельные модели также не подходят для использования. По приведенным выше причинам оптимальным способом задания моделей является поверхностная.

1.3 Способы задания поверхностных моделей

Поверхностные модели могут задаваться двумя способами: параметрическим представлением и полигональной сеткой.

Параметрическое представление — для получения поверхности необходимо вычислить функцию, которая зависит от параметра.

Полигональная сетка — совокупность вершин, ребер и граней, которые определяют форму объекта.

В свою очередь второй способ подразделяется на три варианта реализации:

1) вершинное представление — хранятся вершины, указывающие на другие вершины, с которыми они соединены;

2) список граней — объект представляется как множество граней и вершин;

3) таблица углов — хранит вершины в предопределенной таблице.

Наиболее важной характеристикой для выбора способа задания поверхностной модели в курсовой работе является скорость. Поэтому была выбрана модель, заданная полигональной сеткой, которая позволяет избежать проблем при описании сложных объектов сцены. Оптимальный способ, позволяющий эффективно преобразовывать модели, является способ хранения полигональной сетки при помощи списка граней.

1.4 Модель дождевой капли

Существует множество способов представления дождевых капель в зависимости от используемых физических свойств: геометрических, динамических и оптических.

Один из способов реалистичной и эффективной визуализации осадков (не только дождя) на сценах с движущейся камерой — наложение текстур на двойной конус. Данный метод предлагает более точный контроль над факторами движения и внешним видом капель.

Еше один вариант визуализации капли — использование сферы или эллипса. Последний способ позволяет улучшить восприятие сцены, так как при помощи масштабирования объекта, можно воссоздать эффект воздействия гравитации на капли.

1.5 Алгоритм анимации дождя

Дождь традиционно моделируется одним из двух способов: либо как система частиц, либо как геометрия, ориентированная на камеру, с прокручивающимися текстурами [1, с. 2].

Методы, использующие второй способ анимации дождя, обычно используются при моделировании в реальном времени. Эти методы быстрые, но результаты могут выглядеть так, как будто им не хватает глубины. Кроме того, используя алгоритмы такого рода сложно показать комплексную динамику, такую как штормовой ветер, или реагировать на локальное освещение, такое как уличные фонари [1, с. 2].

Дождь также можно моделировать как систему частиц, но в прошлом этот подход считался медленным для приложений реального времени, особенно для сцен, изображающих сильный дождь [1, с. 2].

Упрощенная анимация падения дождевых капель. Для создания эффекта падения дождя нужно менять координаты y у всех капель через определенные равные интервалы времени. Чтобы изменить направления падения дождя, нужна изменить координаты x и z. Таким образом, создается эффект падения дождя под определенным углом к горизонту.

1.6 Алгоритмы удаления невидимых линий и поверхностей

Сложность задачи удаления невидимых линий и поверхностей привела к появлению большого числа различных способов ее решения. Наилучшего решения обшей задачи удаления невидимых линий и поверхностей не существует. Учет эффектов прозрачности, фактуры, отражения и т. п. не входит в задачу удаления невидимых линий или поверхностей. Естественнее считать их частью процесса визуализации изображения. Однако многие из этих эффектов встроены в алгоритмы удаления невидимых поверхностей. Существует тесная взаимосвязь между скоростью работы алгоритма и детальностью его результата. Ни один из алгоритмов не может достигнуть хороших оценок для этих двух показателей одновременно. По мере создания все более быстрых алгоритмов можно строить все более детальные изображения [5].

Алгоритмы удаления невидимых линий или поверхностей можно классифицировать по способу выбора системы координат или пространства, в котором они работают. Выделяют три класса алгоритмов удаления невидимых линий или поверхностей:

1) алгоритмы, работающие в объектном пространстве;

2) алгоритмы, работающие в пространстве изображения;

3) алгоритмы, формирующие список приоритетов [5].

Алгоритмы, работающие в объектном пространстве, имеют дело с физической системой координат, в которой описаны эти объекты. При этом получаются весьма точные результаты, ограниченные лишь точностью вычислений. Полученные изображения можно свободно увеличивать во много раз. Алгоритмы, работающие в объектном пространстве, особенно полезны в тех приложениях, где необходима высокая точность [5].

Алгоритмы же, работающие в пространстве изображения, имеют дело с системой координат того экрана, на котором объекты визуализируются. При этом точность вычислений ограничена разрешающей способностью экрана.

Обычно разрешение экрана бывает довольно низким. Результаты, полученные в пространстве изображения, а затем увеличенные во много раз, не будут соответствовать исходной сцене [5].

Алгоритмы, формирующие список приоритетов, работают попеременно в обеих упомянутых системах координат [5].

В рамках данной курсовой работы разрабатывается программа визуализации дождя в реальном времени, поэтому быстродействие алгоритмов является самым важным фактором. Исходя из этого, нужно выбирать алгоритмы удаления невидимых линий и поверхностей, которые работают в пространстве изображений.

Алгоритм Робертса характеризуется высокой точностью вычислений, но существует ограничение на выпуклость тел и он работает менее эффективно с появлением новых объектов на сцене.

Алгоритм, использующий Z-буфер, использует относительно много памяти (так как использует массивы) и с помощью него сложно реализовать эффекты прозрачности. Кроме того, недостаток состоит в трудоемкости и высокой стоимости устранения лестничного эффекта. Но он обладает простой реализацией, не требует временных затрат на сортировку объектов сцены и скорость работы алгоритма растет линейно при появлении новых объектов.

Алгоритм обратной трассировки лучей позволяет работать с поверхностями, заданными в математической форме, а также позволяет получить высокую реалистичность синтезируемого изображения. С другой стороны, алгоритм обладает низкой производительностью.

Таким образом, наилучшим алгоритмом для удаления невидимых линий и поверхностей при визуализации дождя будет алгоритм, использующий Z-буфер.

1.7 Алгоритм построения теней

Целью текущей куросовой работы является визуализация дождя. Таким образом, есть возможность пренебречь тенями и использовать для визуализации объектов алгоритм, выбранный в пункте 1.6.

Вывод из аналитической части

В ходе выполнения аналитической части курсовой работы были рассмотрены способы представления объектов и алгоритмы, необходимые для построения реалистичных изображений — алгоритмы, удаляющие невидимые линии и поверхности и алгоритмы построения теней.

В итоге были выбраны:

1) поверхностная модель (путем хранения списка граней) для представления объектов;

2) сфера для представления дождевых капель;

3) алгоритм, использующий Z-буфер для удаления невидимых линий и поверхностей.

2 Конструкторская часть

В данной части описаны все необходимые алгоритмы и средства для визуализации дождя.

2.1 Общий алгоритм визуализации сцены

1. Установить начальную скорость падения дождя.
2. Установить начальный размер дождевых капель.
3. Установить направление падения дождя.
4. Установить плотность падения дождя.
5. Установить соответствующую цветовую гамму исходя из выбранного времени суток (по умолчанию — ночь).
6. Установить начальное положение источника света.
7. Установить начальное положение камеры.
8. С учетом текущего положения камеры и источника света визуализировать падение дождя.

2.2 Графический конвейер

Графический конвейер — комплекс визуализации трехмерной графики, последовательность этапов, выполняющихся в фиксированном порядке. Каждое состояние принимает информацию из предыдущего состояния и отправляет в следующее. Стандартный графический конвейер обрабатывает вершины, геометрические примитивы, а также пиксели конвейерным способом.

На рисунке 2.1 изображена схема графического конвейера, с помощью которого происходит преобразование локальных координат всех объектов в экранные координаты.

Рисунок 2.1 – Графический конвейер

Все преобразования происходят за счет последовательного умножения следующих матриц: модельной, видовой и проекционной. Для составления модельной матрицы нужно умножить матрицы масштабирования, поворота и параллельного переноса.

Матрица масштабирования:

$$S = \begin{bmatrix} kx & 0 & 0 & 0 \\ 0 & ky & 0 & 0 \\ 0 & 0 & kz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{2.1}$$

где kx, ky, kz — коэффициенты масштабирования.

Матрица поворота вокруг оси X:

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\alpha) & -sin(\alpha) & 0 \\ 0 & sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{2.2}$$

где α — угол поворота вокруг оси X.

Матрица поворота вокруг оси Y:

$$R_y = \begin{bmatrix} cos(\beta) & 0 & sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(\beta) & 0 & cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{2.3}$$

где β — угол поворота вокруг оси Y.

Матрица поворота вокруг оси Z:

$$R_z = \begin{bmatrix} cos(\gamma) & -sin(\gamma) & 0 & 0 \\ sin(\gamma) & cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{2.4}$$

где γ — угол поворота вокруг оси Z.

«Полную» матрицу поворота можно получить по следующей формуле:

$$R = R_x \cdot R_y \cdot R_z. \tag{2.5}$$

Матрица параллельного переноса:

$$T = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{2.6}$$

где dx, dy, dz — расстояния, на которые перемещается объект вдоль соответствующих осей.

Таким образом, модельная матрица может быть посчитана по следующей формуле:

$$M = T \cdot R \cdot S, \tag{2.7}$$

где T — матрица параллельного переноса, R — матрица поворота, S — матрица масштабирования.

Чтобы составить видовую матрицу, нужны несколько векторов. Вектор $From$, который описывает направление «взгляда» камеры, вектор $Target$, который указывает куда смотрит камера (этот вектор направлен в обратную сторону относительно вектора $From$), и вектор Up — вектор, направленный вверх. Вектор $Forward$ можно найти как разность векторов $From$ и $Target$. Вектор $Right$ можно найти как векторное произведение векторов Up и $Forward$. Вектор U можно найти как векторное произведение векторов $Forward$ и $Right$.

Видовая матрица:

$$V = \begin{bmatrix} Right_x & Right_y & Right_z & 0 \\ U_x & U_y & U_z & 0 \\ Forward_x & Forward_y & Forward_z & 0 \\ From_x & From_y & From_z & 1 \end{bmatrix}. \tag{2.8}$$

Пусть z_{near} — положение ближней к камере плоскости, а z_{far} — дальней. Матрица проекции:

$$P = \begin{bmatrix} \frac{fov}{aspect} & 0 & 0 & 0 \\ 0 & fov & 0 & 0 \\ 0 & 0 & -f & 0 \\ 0 & 0 & 0 & f \cdot z_{near} \end{bmatrix}, \tag{2.9}$$

где fov — угол обзора в радианах, $aspect$ — отношение ширины сцены к высоте, а f — величина, которую можно посчитать по формуле:

$$f = \frac{-z_{far}}{(z_{far} - z_{near})}. \tag{2.10}$$

Таким образом, полная матрица, которая описывает все преобразования, может быть посчитана по формуле:

$$MVP = P \cdot V \cdot M, \tag{2.11}$$

где P — матрица проекции, V — видовая матрица, M — модельная матрица.

Чтобы получить преобразованные координаты, нужно исходный вектор V умножить на матрицу MVP:

$$V' = MVP \cdot V. \tag{2.12}$$

2.3 Описание геометрии

Для выполнения курсовой работы был выбран формат файлов **OBJ** — это простой формат данных, который содержит только трехмерную геометрию, а именно: позицию каждой вершины, связь координат текстуры с вершиной, нормаль для каждой вершины, а также параметры, которые создают полигоны. В качестве полигонов будут использоваться треугольники.

Для считывания объектов из **OBJ** файла используются строки, начинающиеся на **v** и **f**. Числа, расположенные после **v**, являются координатами текущей вершины, а числа, расположенные после **f**, показывают, из каких

вершин состоит текущая грань.

2.4 Описание алгоритмов

2.4.1 Алгоритм, использующий Z-буфер

Формальное описание алгоритма, использующего Z-буфер.

1. Заполнить буфер кадра фоновым значением интенсивности или цвета.
2. Заполнить Z-буфер минимальным значением z.
3. Преобразовать каждый треугольник в растровую форму.
4. Для каждого пикселя (x, y) в треугольнике вычислить его глубину $z(x, y)$.
5. Сравнить глубину $z(x, y)$ со значением $z_{buffer}(x, y)$, хранящимся в Z-буфере в этой же позиции. Если $z(x, y) > z_{buffer}(x, y)$, то записать атрибут этого треугольника в буфер кадра и заменить $z_{buffer}(x, y)$ на $z(x, y)$. В противном случае никаких действий не производить [5].

Вывод из конструкторской части

В ходе выполнения конструкторской части куросовой работы был описан общий алгоритм визуализации сцены, графический конвейер, описана геометрия и формально описан алгоритм, использующий Z-буфер.

3 Технологическая часть

В текущем разделе приведены средства реализации необходимых структур данных и алгоритмов, а также листинги кода.

3.1 Требования к программному обеспечению

Программа должна предоставлять следующие функции.

1. Смена времени суток — дня и ночи.
2. Поворот камеры вдоль осей X и Y.
3. Изменение скорости падения дождя.
4. Изменение размера капель дождя.
5. Изменение плотности падения дождя.
6. Изменение направления падения дождя.
7. Изменение положения источника света вдоль осей X и Y.

3.2 Средства реализации

Для реализации программного обеспечения был выбран язык **C++** и фреймворк **Qt** для разработки кроссплатформенного программного обеспечения на языке программирования **C++** ввиду следующих причин:

1) язык **C++** является объектно-ориентированным, что позволит ускорить и упростить разработку программного обеспечения с помощью использования классов;
2) существует возможность создания производных классов на основе родительских (наследование);
3) в библиотеке стандартных шаблонов имеется контейнер **std::vector**, который можно использовать для хранения вершин и граней объектов;

4) фреймворк **Qt** дает возможность создавать различные виджеты для реализации графического пользовательского интерфейса;

5) фреймворк предоставляет классы **QGraphicsView** и **QImage** для визуализации сцены;

6) у объектов класса **QImage** есть метод **fill()**, с помощью которого можно менять цвет заднего фона для создания эффекта смены времени суток;

7) фреймворк предоставляет доступ к классу **QTimer**, который можно использовать для изменения позиции капли дождя через определенные интервалы времени (анимация дождя).

Таким образом, с помощью языка **C++** и фреймворка **Qt** можно реализовать программное обеспечение, которое соответствует перечисленным выше требованиям.

Для создания моделей было выбрано программное обеспечение для создания трехмерной компьютерной графики под названием **Blender**, которое позволяет создавать трехмерные модели, состоящие из треугольных полигонов, что подходит под требования, описанные в пункте 2.3.

3.3 Реализация структур данных

3.3.1 Вершины

В листинге 3.1 показана реализация класса, который отвечает за хранение координат одной вершины объекта.

Листинг 3.1 – Класс вершин

```
1 class Vertex
2 {
3 public:
4     Vertex();
5     Vertex(double x, double y, double z);
6     double get_x() const;
7     double get_y() const;
8     double get_z() const;
```

```
9       void set_x(double x);
10      void set_y(double y);
11      void set_z(double z);
12      void change_x(double x);
13      void change_y(double y);
14      void change_z(double z);
15      void normalize(void);
16      Vertex operator + (const Vertex &vertex);
17      Vertex operator - (const Vertex &vertex);
18      Vertex operator * (const double multiplier);
19      Vertex operator ^ (const Vertex &vertex);
20      double operator * (const Vertex &vertex);
21      ~Vertex();
22
23  private:
24      double x, y, z;
25  };
```

3.3.2 Грани

В листинге 3.2 показана реализация класса, который отвечает за хранение одной грани объекта. В массиве **vertices** хранится три порядковых номера вершин, которые образуют один полигон (треугольник).

Листинг 3.2 – Класс граней

```
1   class Face
2   {
3   public:
4       Face() { }
5       Face(int a, int b, int c);
6       Face(const Face &face);
7
8       int getVertex(int index) const;
9
10      ~Face() { }
11
12  private:
13      int vertices[3];
14  };
```

3.3.3 Векторы

В листинге 3.3 показана реализация класса, который представляет собой четырехмерный вектор. Класс предоставляет несколько методов для совершения математических операций над векторами. Четырехмерные векторы нужны для аффинных преобразований и графического конвейера.

Листинг 3.3 – Класс векторов

```
class Vector4d
{
public:
    Vector4d();
    Vector4d(const double x, const double y, const double z,
             const double w);
    Vector4d(const Vertex &vertex);
    void normalize(void);
    Vector4d operator + (const Vector4d &vertex);
    Vector4d operator - (const Vector4d &vertex);
    Vector4d operator * (const double multiplier);
    Vector4d operator ^ (const Vector4d &vertex);
    double operator * (const Vector4d &vertex);
    ~Vector4d();

public:
    double x, y, z, w;
};
```

3.3.4 Матрицы

В листинге 3.4 показана реализация класса, который представляет собой квадратную матрицу размером 4×4. Класс предоставляет операцию умножения двух таких матриц и операцию умножения матрицы на четырехмерный вектор. Кроме того, у класса есть несколько статических методов, которые возвращают тот или иной тип матрицы, который используется в графическом контейнере. Четырехмерные матрицы нужны для аффинных преобразований и графического конвейера.

Листинг 3.4 – Класс матриц

```
#define SIZE 4

class Matrix
{
public:
    Matrix();
    static Matrix getScalingMatrix(const Object& object);
    static Matrix getTranslationMatrix(const Object& object);
    static Matrix getTranslationMatrix(const double x,
                                       const double y,
                                       const double z);
    static Matrix getRotationMatrix(const Object& object);
    static Matrix getLookAtMatrix(Vertex& eye, Vertex& target,
                                  Vertex& up);
    static Matrix getProjectionMatrix(double fov, double aspect,
                                      double znear, double zfar);
    Matrix operator * (const Matrix &matrix);
    Vector4d operator * (const Vector4d &vector);
    ~Matrix() { }

private:
    double elements[SIZE][SIZE];
};
```

3.3.5 Объекты

В листинге 3.5 показана реализация базового класса, который представляет собой объект сцены. Классы других объектов можно создавать на основе класса **Object**. Класс хранит текущее смещение объекта, коэффициенты масштабирования и углы поворотов вокруг трех осей. Кроме того, для **Object** реализованы методы для преобразований объекта и визуализации (метод **draw**).

Листинг 3.5 – Класс объектов

```
class Object
{
public:
    Object();
```

```
5       Object(const Object&) = default;
6       Object(const char *const filename);
7       void draw_polygon(Vertex t0, Vertex t1, Vertex t2, const int
            width,
8                         int *z_buffer,
9                         QImage *scene, QColor color);
10      void draw(const std::size_t width, const std::size_t height,
11                uint8_t red, uint8_t green, uint8_t blue, QImage
                    *scene);
12      std::size_t getVerticesNumber();
13      std::size_t getFacesNumber();
14      void addVertex(Vertex vertex);
15      void addFace(const Face face);
16      Vertex getVertex(std::size_t number);
17      Face getFace(std::size_t number);
18      double get_dx() const;
19      double get_dy() const;
20      double get_dz() const;
21      double get_kx() const;
22      double get_ky() const;
23      double get_kz() const;
24      double get_phi_x() const;
25      double get_phi_y() const;
26      double get_phi_z() const;
27      void set_dx(double dx);
28      void set_dy(double dy);
29      void set_dz(double dz);
30      void set_kx(double kx);
31      void set_ky(double ky);
32      void set_kz(double kz);
33      void set_phi_x(double phi_x);
34      void set_phi_y(double phi_y);
35      void set_phi_z(double phi_z);
36      void translate(double dx, double dy, double dz);
37      void rotate(double phi_x, double phi_y, double phi_z);
38      void scale(double kx, double ky, double kz);
39      ~Object();
40
41  private:
```

```
42      std::vector<Vertex> vertices;
43      std::vector<Face> faces;
44      double dx, dy, dz;
45      double kx, ky, kz;
46      double phi_x, phi_y, phi_z;
47  };
```

3.4 Реализация алгоритмов

3.4.1 Алгоритм, использующий Z-буфер

В листинге 3.6 показана реализация метода **draw_polygon** для класса **Object**, который представляет собой алгоритм, использующий Z-буфер. Метод в качестве параметров принимает три вершины полигона, массив, представляющий собой Z-буфер, указатель на сцену и цвет полигона. Чтобы визуализировать весь объект, нужно вызвать метод **draw_polygon** для всех полигонов.

Листинг 3.6 – Алгоритм, использующий Z-буфер

```
1   void Object::draw_polygon(Vertex t0, Vertex t1, Vertex t2,
2                             int *z_buffer, QImage *scene,
3                             QColor color)
4   {
5       if (t0.get_y() == t1.get_y() &&
6           t0.get_y() == t2.get_y())
7           return;
8       if (t0.get_y() > t1.get_y())
9           std::swap(t0, t1);
10      if (t0.get_y() > t2.get_y())
11          std::swap(t0, t2);
12      if (t1.get_y() > t2.get_y())
13          std::swap(t1, t2);
14      int total_height = t2.get_y() - t0.get_y();
15      for (int i = 0; i < total_height; i++)
16      {
17          bool second_half = i > t1.get_y() - t0.get_y() ||
18                                 t1.get_y() == t0.get_y();
19          int segment_height = second_half ? t2.get_y() -
```

```
20                                  t1.get_y() : t1.get_y() -
                                       t0.get_y();
21          double alpha = (double)i / total_height;
22          double beta  = (double)(i - (second_half ? t1.get_y() -
23                                  t0.get_y() : 0)) /
                                     segment_height;
24          Vertex A = t0 + (t2 - t0) * alpha;
25          Vertex B = second_half ? t1 + (t2 - t1) *
26                     beta : t0 + (t1 - t0) * beta;
27          if (A.get_x() > B.get_x())
28              std::swap(A, B);
29          for (int j = A.get_x(); j <= B.get_x(); j++)
30          {
31              double phi = B.get_x() == A.get_x() ? 1. :
32                           (double)(j - A.get_x()) /
33                           (double)(B.get_x() - A.get_x());
34              Vertex P = A + (B - A) * phi;
35              int k = P.get_x() + P.get_y() * width;
36
37              if (P.get_x() < 0 || P.get_x() >= WIDTH ||
38                      P.get_y() < 0 || P.get_y() >= HEIGHT)
39                  continue;
40              P.set_x(j);
41              P.set_y(t0.get_y() + i);
42              if (z_buffer[k] < P.get_z())
43              {
44                  z_buffer[k] = P.get_z();
45                  scene->setPixel(P.get_x(), HEIGHT -
46                                  P.get_y() - 1, color.rgb());
47              }
48          }
49      }
50  }
```

3.4.2 Алгоритм генерации дождевых капель

В листинге 3.7 показана реализация метода **generateRain**, который отвечает за первоначальное размещение на сцене всех капель дождя.

Листинг 3.7 – Алгоритм генерации дождевых капель

```
void MainWindow::generateRain()
{
    bool even = 0;
    int cube = cbrt(NUM_OF_DROPLETS);
    double init_dx = -0.8, init_dy = 1.11, init_dz = 0.5;
    double delta_x = 1.5 / (cube - 2);
    double delta_y = 0.05;
    double delta_z = 0.1;
    int cube_2 = cube * cube;
    for (std::size_t i = 0; i < cube; i++)
    {
        for (std::size_t j = 0; j < cube; j++)
        {
            for (std::size_t k = 0; k < cube; k++)
            {
                droplets[i * cube_2 + j * cube +
                   k]->translate(init_dx, init_dy, init_dz);
                init_dx += delta_x;
            }
            if (even)
                init_dx = -0.8;
            else
                init_dx = -0.75;
            init_dz -= delta_z;
        }
        init_dy -= delta_y;
        if (even)
        {
            even = 0;
            init_dx = -0.8;
            init_dz = 0.5;
        }
        else
        {
            even = 1;
            init_dx = -0.75;
            init_dz = 0.45;
```

```
37              }
38          }
39      }
```

Вывод из технологической части

В данном разделе был написан исходный код необходимых структур данных и алгоритмов визуализации сцены и анимации объектов.

4 Исследовательская часть

4.1 Примеры работы программы

На рисунках 4.1–4.4 показаны примеры работы программы при различных параметрах дождя и сцены.

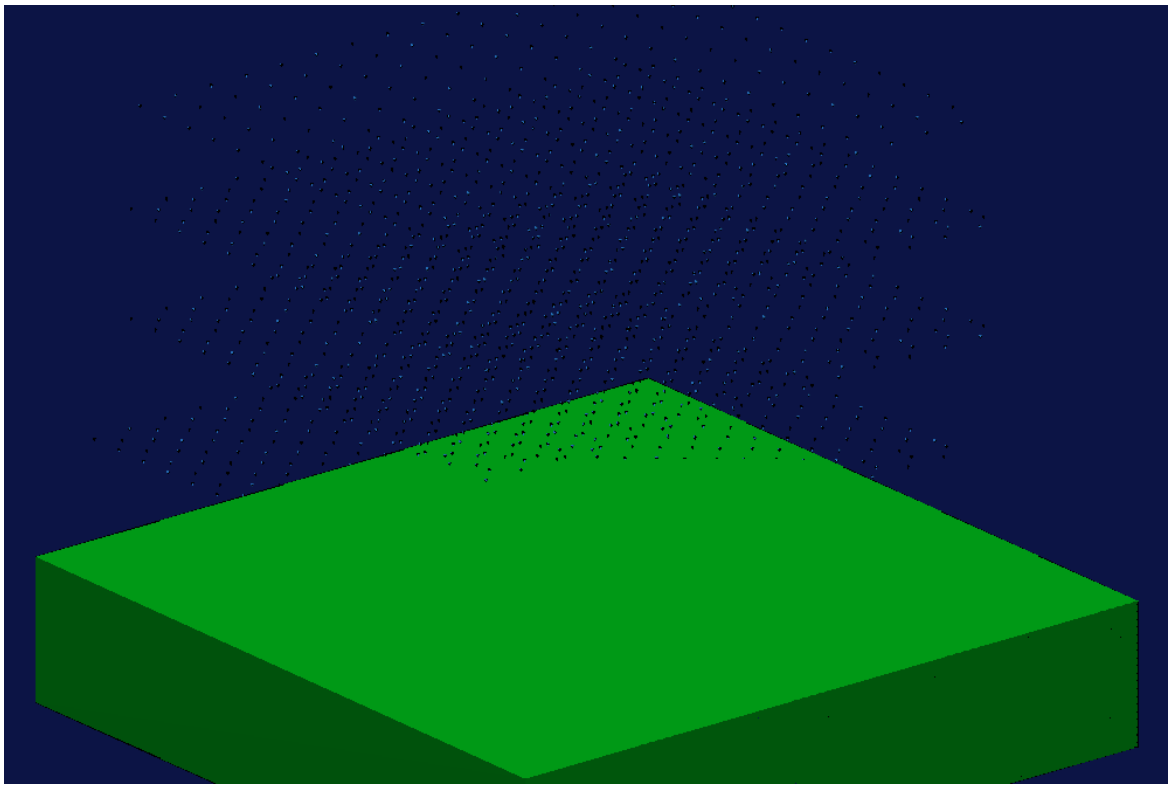

Рисунок 4.1 – Дождь в ночное время суток

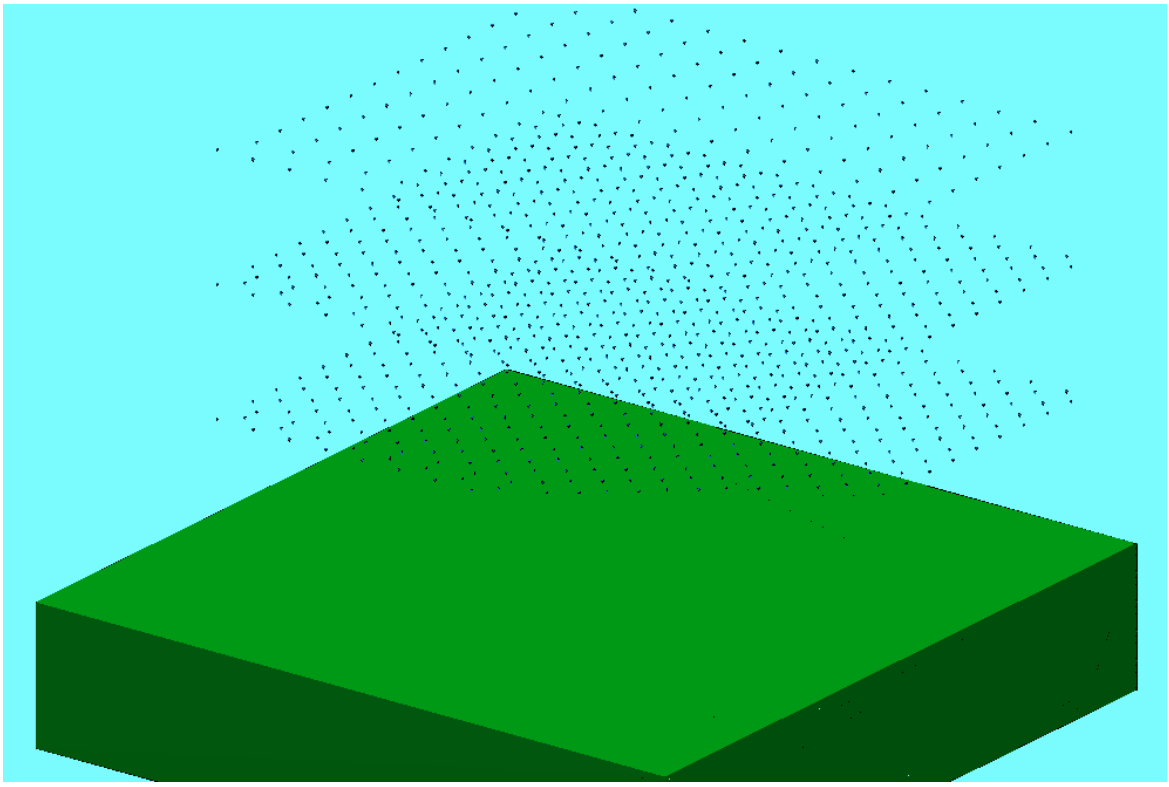

Рисунок 4.2 – Дождь в дневное время суток

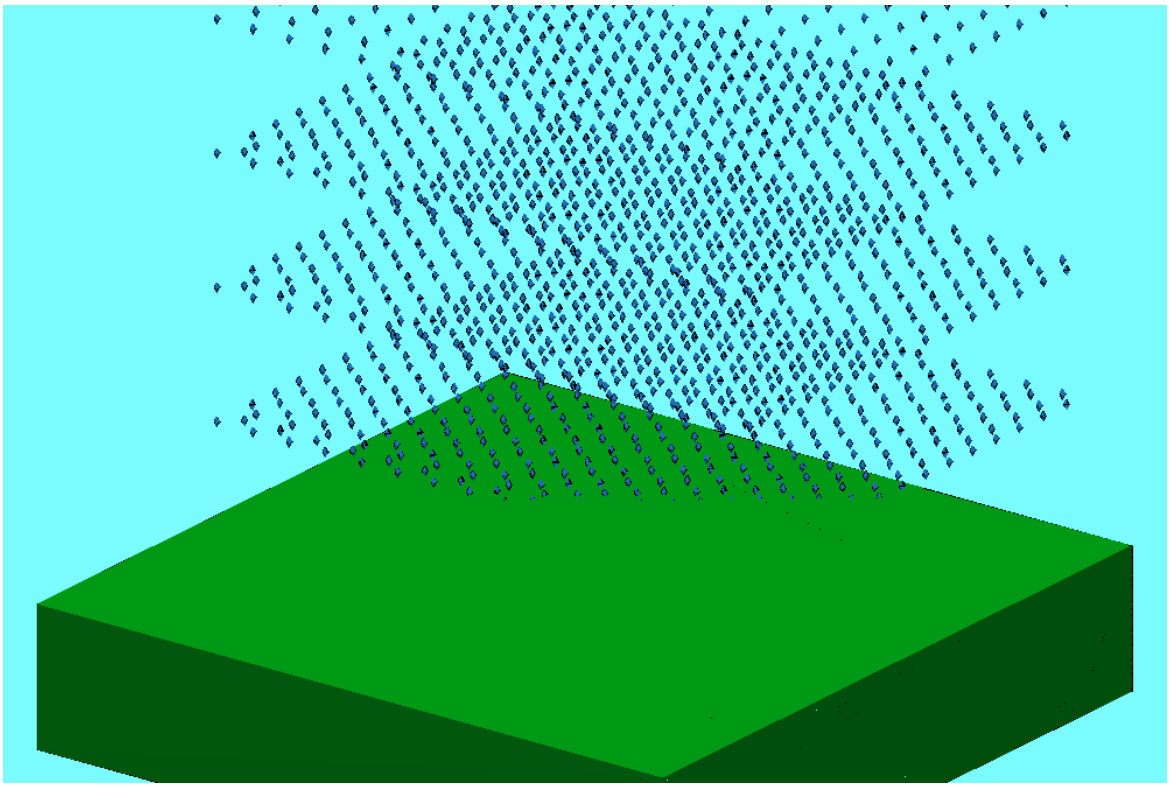

Рисунок 4.3 – Увеличенный размер капель дождя

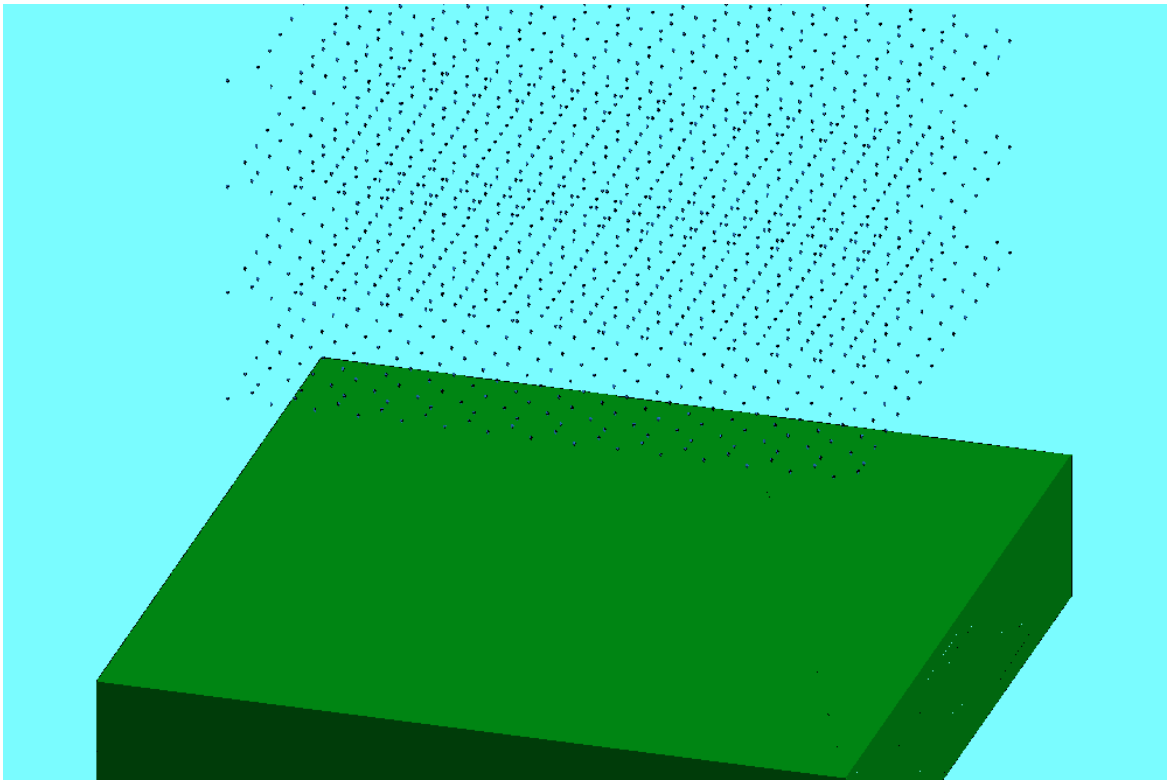

Рисунок 4.4 – Измененное положение камеры и источника света

Вывод из исследовательской части

В данном разделе представлены примеры работы программы при различных параметрах дождя и сцены.

Заключение

В результате выполнения курсовой работы было реализовано программное обеспечение для визуализации дождя в реальном времени с возможностью изменения с помощью графического интерфейса таких характеристик, как размер капель, скорость падения дождя и направление падения дождевых капель. Также была реализована возможность изменения положения камеры и источника света вдоль осей X и Y и смена дня и ночи. Не была реализована возможность изменения плотности дождя.

Были выполнены следующие задачи:

1) выбран способ представления объектов на сцене;
2) выбрана модель дождевых капель;
3) проанализированы алгоритмы удаления невидимых линий и поверхностей и выбран наиболее подходящий;
4) проанализированы алгоритмы создания реалистичного освещения, отражений и теней и выбран наиболее подходящий;
5) проанализированы и выбраны средства программной реализации;
6) реализованы выбранные алгоритмы для создания программы визуализации дождя в реальном времени;
7) создан графический интерфейс для возможности изменения характеристик дождя и сцены пользователем.

Список использованных источников

1. *Tariq S.* Rain // NVIDIA Corporation. — 2007.
2. *Hao P.* Algorithms for Atmospheric Special Effects in Graphics and their Implementation // Kanwal Rekhi School of Information Technology Indian Institute of Technology. — 2008. — С. 1.
3. *Garg K.*, *Nayar S. K.* Photorealistic Rendering of Rain Streaks // Columbia University. — 2006. — С. 996.
4. *Курицына В. В.* Системы автоматизированного проектирования // Российский государственный технологический университет имени К. Э. Циолковского. — 2011. — С. 37.
5. *Демин А. Ю.*, *Кудинов А. В.* Компьютерная графика // Томский политехнический университет. — 2005.