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 эффектов погоды в сельской местности»

Студент ИУ7-52Б _____ Обревская В.В.
(Группа) (Подпись, дата) (фамилия, и.о.)

Руководитель курсовой работы _____ Кострицкий А.С.
(Подпись, дата) (фамилия, и.о.)

2022 г.

Оглавление

Введение

В настоящее время компьютерная графика используется достаточно широко. Типичная область ее применения – это кинематография и компьютерные игры.

На сегодняшний день большое внимание уделяется алгоритмам получения реалистичного изображения. Зачастую эти алгоритмы ресурсозатратны: чем более качественное изображение требуется получить, тем больше времени и памяти тратится на его синтез. Это становится проблемой при создании динамической сцены, где на каждом временном интервале необходимо производить расчеты заново.

Цель данной работы — реализовать построение трехмерной сцены и визуализацию погодных эффектов в сельской местности.

Чтобы достигнуть поставленной цели, требуется решить следующие задачи:

1) описать структуру трехмерной сцены, включая объекты, из которых состоит сцена, и дать описание выбранных погодных явлений, которые будут визуализированы;

2) провести анализ алгоритмов построения реалистичных изображений;

3) выбрать и/или модифицировать существующие алгоритмы трехмерной графики, которые позволят построить реалистичные изображения;

4) реализовать выбранные алгоритмы;

5) разработать программное обеспечение, которое позволит отобразить трехмерную сцену и визуализировать погодные эффекты в сельской местности.

1. Аналитическая часть

В данном разделе представлено описание объектов сцены, а также рассматриваются различные алгоритмы, использованные для визуализации сцены.

1.1. Описание объектов сцены

Сцена состоит из следующих объектов:

1) Источник света — материальная точка пространства, испускающая лучи света во все стороны, в зависимости от расположения источника, определяется тень от объектов, расположенных на сцене. Положение источника света задается относительно текущей точки наблюдения координатами x, y, z. Также к характеристикам относятся цвет и интенсивность излучения

2) Камера — характеризуется пространственным положением и направлением взгляда.

3) Площадка сцены — ограничивающая плоскость, на которой расставляются модели. Объекты располагаются только на одной из сторон площадки.

4) Молния — ломаная линию, которая имеет начало и конец, а также несколько ветвей.

5) Капли дождя — частички с заданной траекторией движения, интенсивностью и цветом. Интенсивность — количеством частичек. Траектория движения — прямая $Ax + By + C = 0$.

6) Ветер — вектор в трехмерном пространстве, который задает направление движения осадков.

7) Объект сцены — модель, расположенная на сцене, которая представляет собой набор граней, описываемых точками в пространстве, которые соединены ребрами. Предусмотрена загрузка файлов-описателей сетки на этапе выполнения программы.

1.2. Описание модели трёхмерного объекта на сцене

Отображением формы и размеров объектов являются модели. Обычно используются три формы задания моделей [1].

1) Каркасная (проволочная) модель.

   В этой модели хранится информация только о вершинах и рёбрах объектов. Недостатком данной моделя является то, что она не всегда правильно передает форму объекта.

2) Поверхностная модель.

   Поверхность может описываться аналитически или полигональной сеткой. Такая информационная модель содержит данные только о внешних геометрических параметрах объекта. Недостаток: отсутствует информация о том, с какой стороны поверхности находится материал.

3) Объёмная (твердотельная) модель.

   При твердотельном моделировании учитывается еще и материал, из которого изготовлен объект. К информации о поверхности добавляется информация о том, с какой стороны поверхности расположен материал. Это делается с помощью указания направления внутренней нормали.

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

1.3. Описание способа задания трёхмерного объекта на сцене

Поверхностную модель можно задать несколькими способами [2] :

1) Аналитический способ.

   Этот способ характеризуется описанием объекта в неявной форме. Для получения поверхности нужно вычислять функцию, зависящую от параметра.

2) Полигональная сетка.

   Данный способ характеризуется совокупностью вершин, граней и ребер, которые определяют форму многогранного объекта в трехмерной компьютерной графике. Гранями обычно являются треугольники, четырёхугольники или другие простые выпуклые многоугольники (полигоны). В данной работе грани имеют форму треугольников, так как любой полигон можно представить в виде треугольника.

   Существует несколько способов хранения информации о полигональной сетке:

   — Список граней.

     Характеризуется множеством граней и множеством вершин. В каждую грань входят как минимум три вершины.

   — «Крылатое» представление.

     Представляет вершины, грани и ребра сетки. Это представление широко используется в программах для моделирования для предоставления высочайшей гибкости в динамическом изменении геометрии сетки, потому что могут быть быстро выполнены операции разрыва и объединения. Их основной недостаток – высокие требования памяти и увеличенная сложность из-за содержания множества индексов.

— Полурёберные сетки.

  То же «крылатое» представление, но информация обхода хранится для половины грани.

— Вершинное представление.

  Хранятся лишь вершины, которые указывают на другие вершины. Простота представления даёт возможность проводить над сеткой множество операций.

— Таблица углов.

  Это таблица, хранящая вершины. Обход заданной таблицы неявно задаёт полигоны. Такое представление более компактно и более производительно для нахождения полигонов, но, в связи с тем, что вершины присутствуют в описании нескольких углов, операции по их изменению медленны.

При выборе способа задания объекта в курсовой работе определяющим фактором стала скорость выполнения геометрических преобразований.

Оптимальное представление – полигональная сетка. Такая модель позволит легко описывать сложные объекты сцены [3]. Наиболее удобным способом хранения информации о полигональной сетке является список граней, так как данные в нем можно эффективно преобразовывать в сравнении с другими перечисленными методами, представление позволяет явный поиск вершин грани и граней, окружающих вершину. Также оно является наиболее широко применимым среди остальных перечисленных представлений [4].

1.4. Анализ алгоритмов удаления невидимых линий и поверхностей

При выборе алгоритма удаления невидимых линий необходимо учитывать особенности поставленной задачи. Из-за наличия частичек осадков, которые будут двигаться, применяемый алгоритм должен работать быстро, иначе осадки будут выглядеть не как анимация, а как набор кадров.

Чтобы избавиться от данного явления можно добавить предзагрузку кадров анимации движения осадков и показывать их только после полного рендеринга анимации. Для ускорения расчетов следует рендерить только часть анимации, а потом воспроизводит ее несколько раз. Тогда нужно будет учесть, что первый кадр сформированного блока анимации должен совпадать с заключительным. Иначе будут заметны места «склейки» двух блоков. Кроме того, т.к. загруженные объекты остаются неподвижными во время анимации осадков, следует сначала отрисовывать сцену, а потом добавлять осадки поверх нее.

Рассмотрим ключевые алгоритмы удаления невидимых линий и поверхностей [5].

Алгоритм, использующий Z-буфер

Данный алгоритм работает в пространстве изображения [5]. Используется два буфера:

— буфер кадра, в котором хранятся атрибуты каждого пикселя в пространстве изображения;

— Z-буфер, куда помещается информация о координате z для каждого пикселя.

Первоначально в Z-буфере находятся минимально возможные значения z, а в буфере кадра располагаются пиксели, описывающие фон. Каждый многоугольник преобразуется в растровую форму и записывается в буфер кадра.

При вычислении глубины нового пикселя, она сравнивается со значением из Z-буфера. Если новый пиксель находится ближе к наблюдателю, чем предыдущий, то он заносится в буфер кадра и корректируется Z-буфер.

Для решения задачи вычисления глубины z каждый многоугольник описывается уравнением $ax + by + cz + d = 0$. При $c = 0$ многоугольник для наблюдателя вырождается в линию.

Для текущей сканирующей строки значение y является постоянным. Поэтому имеется возможность рекуррентно высчитывать z' для каждого

$x' = x + dx$:

$$z' - z = -\frac{ax' + d}{c} + \frac{ax + d}{c} = \frac{a(x - x')}{c}. \quad (1.1)$$

Получим: $z' = z - \frac{a}{c'}$, так как $x - x' = dx = 1$.

При этом стоит отметить, что для невыпуклых многогранников предварительно потребуется удалить нелицевые грани.

Особенностью алгоритма является простота его реализации, а также скорость работы с объектами. При этом для современных компьютеров затраты по памяти для буферов изображений будут некритичны.

Таким образом, данный алгоритм подходит для решения поставленной задачи, так как он позволяет быстро отрисовывать динамические сцены [5].

Алгоритм Робертса

Данный алгоритм работает в объектном пространстве, решая задачу только с выпуклыми телами [6].

Алгоритм выполняется в три этапа.

1) Этап подготовки исходных данных. На данном этапе должна быть задана информация о телах. Для каждого тела сцены должна быть сформирована матрица тела V. Размерность матрицы – $4/cdotn$, где n – количество граней тела. Каждый столбец матрицы представляет собой четыре коэффициента уравнения плоскости $ax+by+cz+d = 0$, проходящей через очередную грань. Таким образом, матрица тела будет представлена в следующем виде:

$$V = \begin{pmatrix} a_1 & a_2 & \dots & a_n \\ b_1 & b_2 & \dots & b_n \\ c_1 & c_2 & \dots & c_n \\ d_1 & d_2 & \dots & d_n \end{pmatrix} \quad (1.2)$$

Если заданная точка принадлежит плоскости, то в результате умножения координат данной точки на вектор коэффициентов уравнения плоскости будет получено нулевое значение. Иначе, будет получено ненулевое значение, а его знак будет определять, по какую сторону

от данной плоскости находится точка. В алгоритме Робертса предполагается, что если точка находится внутри тела, то все произведения координат данной точки на вектора коэффициентов уравнений плоскостей, составляющих это тело, будут положительны. Поэтому зачастую требуется производить коррекцию полученной матрицы.

В случае, если для очередной грани условие положительности описанного произведения не выполняется, соответствующий столбец матрицы необходимо умножить на -1. Для проведения проверки следует взять точку, координаты которой будут получены усреднением всех соответствующих координат вершин данного тела.

2) Этап удаления невидимых ребер, экранируемых другими телами сцены. На данном этапе для определения невидимых точек ребра требуется построить луч, соединяющий точку наблюдения с точкой на ребре. Точка будет невидимой, если луч на своем пути встречает в качестве преграды рассматриваемое тело. Если тело является преградой, то луч должен пройти через тело. Если луч проходит через тело, то он находится по положительную сторону от каждой грани тела.

3) Этап удаление линий пересечения тел, экранируемых самими телами, которым принадлежат эти линии, так как эти тела связаны отношением протыкания, и другими телами. В случае существования отношения протыкания появляются решения на границе $\alpha = 0$. Для этого необходимо запомнить все точки протыкания и добавить к сцене отрезки, образованные путём соединения каждой точки протыкания со всеми остальными точками протыкания для данной пары объектов. Затем проверяется экранирование этих отрезков данными и другими телами. Видимые отрезки образуют структуру протыкания.

Таким образом, алгоритм Робертса не подходит для решения поставленной задачи из-за трудностей, возникающих при работе с невыпуклыми телами (приходится разбивать их на выпуклые) [6].

Алгоритм обратной трассировки лучей

Алгоритмы трассировки лучей на сегодняшний день считаются наиболее мощными при создании реалистичных изображений. Метод прямой и обратной трассировки [7] заключается в том, что от момента испускания лучей источником света до момента попадания в камеру, траектории лучей отслеживаются, и рассчитываются взаимодействия лучей с лежащими на траекториях объектами. Луч может быть поглощен, диффузно или зеркально отражен или, в случае прозрачности некоторых объектов, преломлен. Для создания реалистичного изображения, по правилам обратной трассировки, каждую частицу в осадках нужно будет рассматривать, как отдельный объект сцены, внутри которого могут возникнуть явления дисперсии, преломления и внутреннего отражения.Тогда алгоритм трассировки лучей будет требовать большого количества вычислений, поскольку он предполагает поиск пересечений всех объектов сцены со всеми лучами. Поэтому время синтеза изображения окажется очень большим.

Таким образом, данный алгоритм не подходит для моей задачи, так как он не позволит быстро отрисовать динамические сцены, хотя его скорость может быть увеличена, например, при использовании параллельных вычислений.

Алгоритм Варнока

Алгоритм Варнока основывается на разбиении картинной плоскости на части, для каждой из которых исходная задача может быть решена достаточно просто. Рассматривается окно и решается, его содержимое достаточно просто для отрисовки или его нужно разбить на фрагменты, до тех пор пока содержимое не станет простым. Особенность алгоритма заключается в том, что он работает только в пространстве изображений и его эффективность зависит от расположения объектов на сцене(чем меньше пересечений объектов, тем быстрее будет отрисована сцена).

Вывод

В качестве алгоритма удаления невидимых рёбер и поверхностей был выбран алгоритм Z-буфера, так как он лучше подходит для прорисовки

динамических сцен из-за быстроты вычислений.

1.5. Учёт теней

В алгоритме трассировки лучей тени высчитываютмя по ходу выполнения алгоритма. Пиксель затенен, если луч после попадания на объект не попадает в другой объект или источник света.

Так как в качестве алгоритма удаления невидимых линий и поверхностей был выбран алгоритм Z-буфера, то для вычисления теней можно использовать модифицированный алгоритм Z-буфера для теневых карт, где предполагается, что освещены только те объекты, которые видны из положения источника. Данный подход поможет не усложнять структуру программы и быстро вычислить видимость объектов сцены.

1.6. Учёт освещения

Модель Ламберта моделирует идеальное диффузное освещение. Свет при попадании на поверхность рассеивается равномерно во все стороны. В простом методе освещения интенсивность рассчитывается по закону Ламберта:

$$I = I_0 * \cos(\alpha), \tag{1.3}$$

где

— I – результирующая интенсивность света в точке;

— I_0 – интенсивность источника;

— α – угол между нормалью к поверхности и вектором направления света.

Данная модель является одной из самых простых в реализации моделей.

1.7. Анализ алгоритмов моделирования осадков

Из-за большого числа движущихся капель дождя сложно быстро отрисовывать выпадение осадков. Ведь у каждой частицы есть свои физические свойства и для каждой нужно высчитывать расположение на сцене в данный момент времени.

Система частиц

Система частиц содержит в себе совокупность частиц осадков, которые рассматривает как материальные точки [8]. Обычно все частицы в системе меняют скорость или размер по общему закону. Для изменения интенсивности осадков изменяют общее количество частиц. Чтобы не проводить много рассчетов, предполагается, что частицы не поглощают свет и не отбрасывают тени.

Метод Кшитиза и Шри

Кшитиза и Шри предложили модель капли дождя, которая учитывает сложные взаимодействия света, положения наблюдателя [9]. Капля представляется колеблющейся формой, благодаря чему она выглядит реалистичнее. Особенность алгоритма в том, что предполагается, что капля падает строго вниз, то есть не предусмотрено наличия ветра, и из-за сложных взаимодействий света отрисовка динамических сцен затруднительна.

Вывод

Для реализации данной задачи была выбрана система частиц, так как она позволяет моделировать наличие ветра и усорять расчеты для динамических сцен.

1.8. Вывод

Для задания трёхмерных моделей была выбрана поверхностная модель, представление — полигональная сетка. Также в данном разделе были рассмотрены алгоритмы удаления невидимых линий и поверхностей, методы закрашивания поверхностей, модели освещения, алгоритмы построения теней и моделирования осадков. В качестве алгоритма удаленияневидимых линий был выбран Z-буфер, метод закраски — простой, построение теней будет выполняться с помощью теневых карт, построенных алгоритмом Z-буфера, для моделирования осадков была выбрана система частиц.

2. Конструкторская часть

В данном разделе будут рассмотрены требования к программе и алгоритмы визуализации сцены и погодных явлений.

2.1. Общий алгоритм визуализации трехмерной сцены

Рассмотрим алгоритм визуализации сцены:

1) Задать объекты сцены.
2) Задать источник света.
3) Используя Z-буфер получить изображение сцены.
4) Отобразить тени, используя теневые карты, полученные с помощью Z-буфера.
5) Изобразить молнию.
6) Если идет дождь, то пока в буфере есть частицы осадков, выполнить:
    1) Используя систему частиц, наложить осадки на изображение.
    2) Нарисовать изображение.
    3) Обновить буфер системы частиц.

2.2. Алгоритм Z-буфера

На рисунках 2.1 – 2.2 представлены схемы алгоритмов Z-буфера и модифицированного Z-буфера для теневых карт.

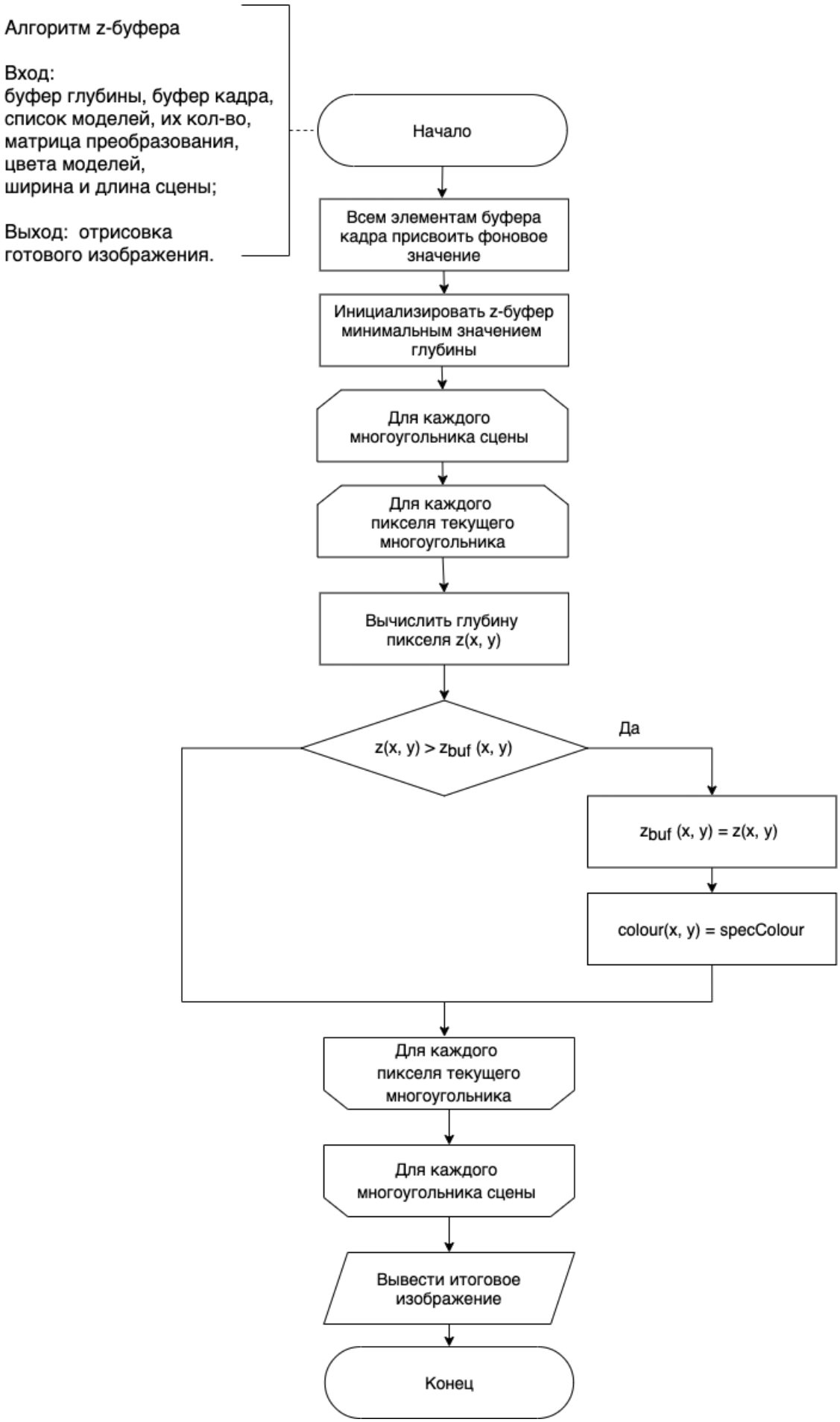

Рисунок 2.1 – Алгоритм Z-буфера

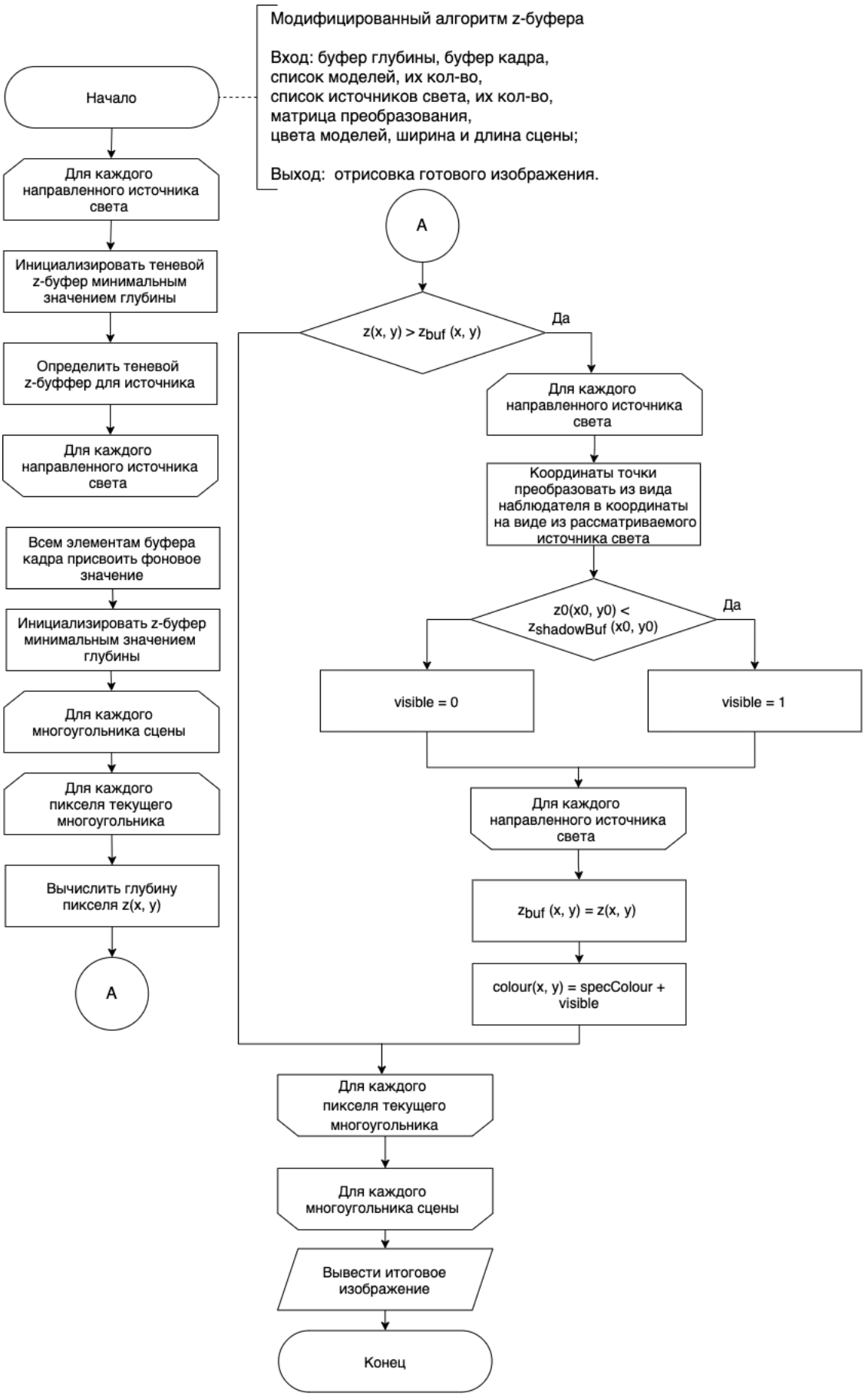

Рисунок 2.2 – Модифицированный алгоритма z-буфера

2.3. Алгоритм генерации молнии

На рисунке 2.3 представлена схема алгоритма генерации молнии.

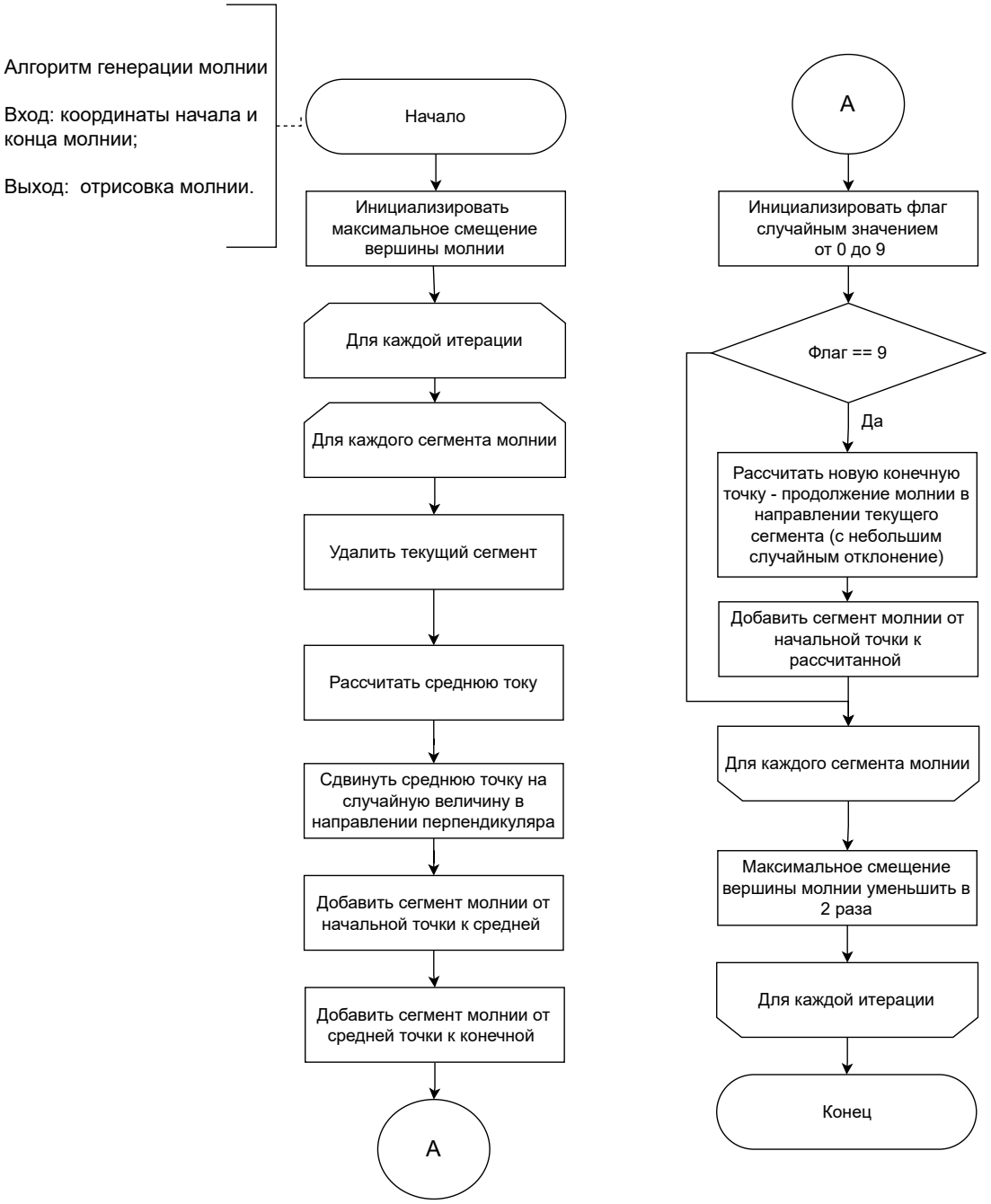

Рисунок 2.3 – Схема алгоритма генерации молнии

На рисунке 2.4 представлена генерации молнии.

Рисунок 2.4 – Генерация молнии

2.4. Алгоритм генерации осадков

На рисунке 2.5 представлена схема алгоритма генерации осадков.

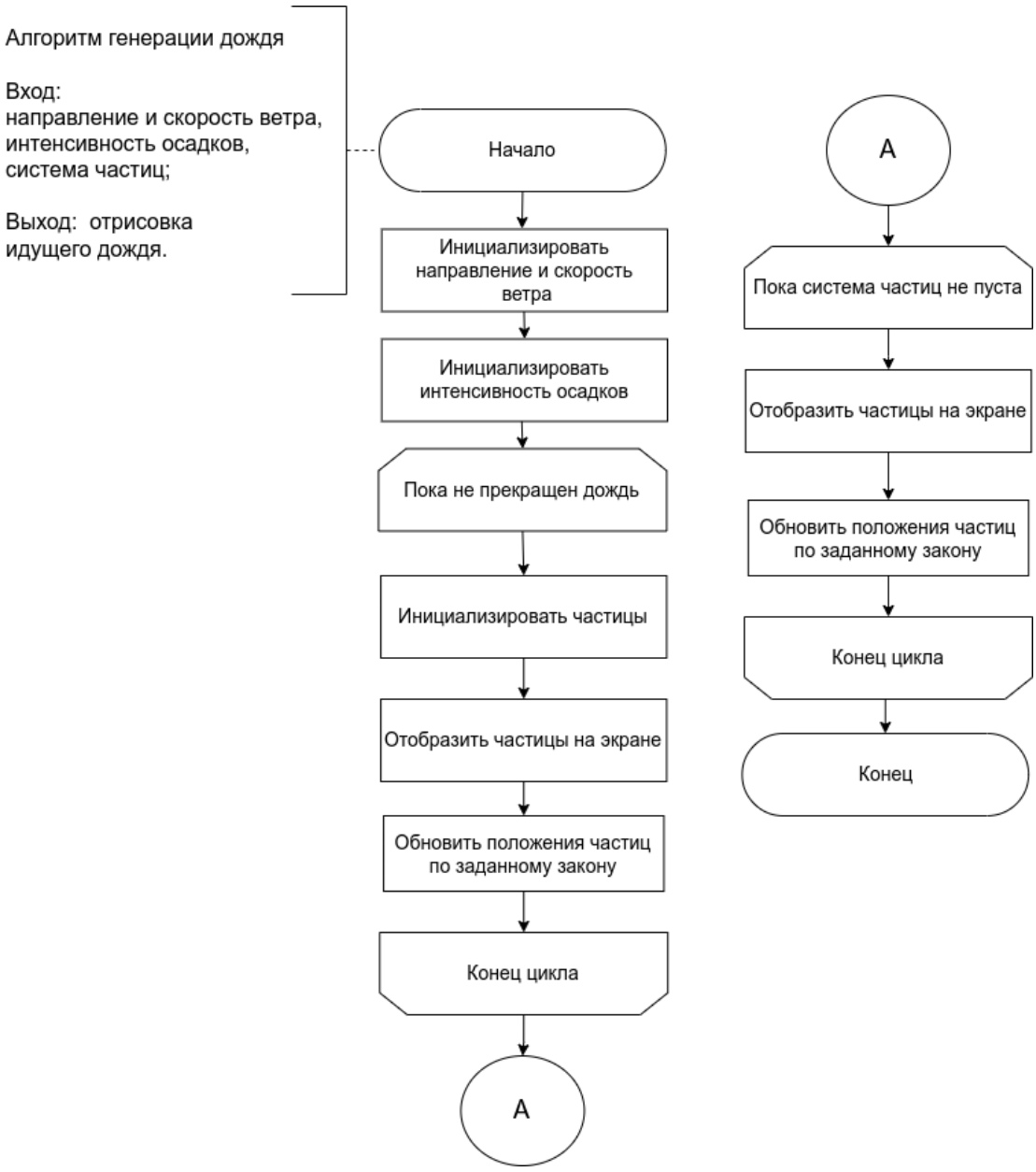

Рисунок 2.5 – Алгоритм генерации осадков

2.5. Разработка и обоснование используемых типов и структур данных

Для разрабатываемого ПО нужно будет реализовать следующие типы и структуры данных.

1) Структура сцены – список объектов сцены, список источников света.
2) Объекты сцены задаются массивом полигонов
3) Полигоны содержат три точки, цвет полигона
4) Цвет – вектор из трех чисел (три целочисленные переменные, характерезующие цветовую модель RGB).
5) Структура молнии содержит вектор, характеризующий главную и побочные ветви и содержащий точки изгибов.
6) Структура системы частиц содержит в себе:
   - Координаты положения частиц в пространстве.
   - Направление их движения.
7) Структура источника света содержит:
   - Координаты положения источника света в пространстве.
   - Интенсивность излучения (вещественное число).

2.6. Вывод

В данном разделе были рассмотрены реализованные алгоритмы и описаны представления данных и используемых структур.

3. Технологическая часть

В данном разделе будет представлена реализация решения поставленной задачи в формате листинга кода. Также будут указаны требования к ПО и средства реализации, описан пользовательский интерфейс и результаты проведенного тестирования программы.

3.1. Требования к ПО

Были учтены следующие требования к программе:

1) Программа выводит сообщение об ошибке и генерирует ненулевой код возврата в случае возникновения исключения в процессе выполнения.

2) Программа выполняет построение изображения сцены из трехмерных объектов.

3) Программа выполняет обработку только многогранников, геометрия которых описана с помощью *obj* файла.

4) Программа принимает на вход файлы, составленные по правилам, которые описаны ниже.

5) Программа может сохранять полученное изображение в *png* файл.

6) Программа предоставляет возможность отрисовывать анимацию дождя.

7) Программа предоставляет возможность отрисовывать анимацию молний.

3.2. Требования к входным данным

Требования к входному файлу:

1) На вход принимаются только *obj* файлы.

2) Модели в файле должны быть триангулированы.

3) Описание каждой модели начинается с буквы *o*.

3.3. Средства реализации

В качестве языка программирования для разработки ПО был выбран язык программирования $C++$[10]. Данный выбор обусловлен наличием необходимого функционала для работы с наборами объектов. Также был выбран фреймворк *Qt*[11] в связи с наличием необходимых для работы с компьютерной графикой библиотек и встроенных средств.

Для хранения информации о многогранниках, заданных при помощи полигональной сетки, были использованы *obj* файлы. Данный выбор обусловлен тем, что *obj* является одним из самых популярных форматов передачи трехмерной компьютерной геометрии[12]. Для хранения полученного изображения был выбран формат *png*, так как среднеквадратическая ошибка — среднее различие в квадрате между идеальным и фактическим пиксельными значениями — для, например, *jpeg* файлов на больших изображениях гораздо больше, чем у *png* файлов [13], и качество изображения формата *png* не меняется при любой степени сжатия.

3.4. Реализация разработанного ПО

В приведенных ниже листингах представлены следующие реализации:

1) модифицированный алгоритм Z-буфера (листинг 3.1);

2) алгоритм простой закраски (листинг 3.2);

3) алгоритм генерации молнии (листинг 3.3).

Листинг 3.1 – Реализация модифицированного алгоритма Z-буфера

```
void ZBuffer::putPolygon(std::vector<QVector3D> &points,
                         std::vector<LightSource> &ls, QColor c,
                         std::vector<std::vector<std::vector<
                            double>>> &shadows) {
  int ymax, ymin, xmax, xmin;
  ymax = ymin = points[0].y();
  xmax = xmin = points[0].x();
  for (size_t i = 1; i < points.size(); i++) {
    int x = points[i].x();
    int y = points[i].y();
    if (y > ymax)
      ymax = y;
    else if (y < ymin)
      ymin = y;
    if (x > xmax)
      xmax = x;
    else if (x < xmin)
      xmin = x;
  }
  ymin = (ymin < 0) ? 0 : ymin;
  ymax = (ymax < _sY) ? ymax : _sY - 1;
  xmin = (xmin < 0) ? 0 : xmin;
  xmax = (xmax < _sX) ? xmax : _sX - 1;
  std::tuple<int, int, int, int> coef = planeCoef(points);
  QVector3D N(std::get<0>(coef), std::get<1>(coef), std::get<2>(
     coef));
  N.normalize();
  if (QVector3D::dotProduct(N,QVector3D(0, 0, 5000)) < 0)
      N *= (-1);
  for (int i = xmin; i <= xmax; i++)
    for (int j = ymin; j <= ymax; j++) {
      if (isInside(i, j, points)) {
        double z = calculateZ(i, j, coef);
        if (z > _buf[i][j].z) {
          _buf[i][j].z = z;
          _buf[i][j].c = calculateColor(i,j,z, N, ls, c, shadows);
        }
      }
    }
}
```

Листинг 3.2 – Реализация алгоритма простой закраски

```
QColor calculateColor(int x, int y, double z,
                      QVector3D N,
                      std::vector<LightSource> ls, QColor c,
                      std::vector<std::vector<std::vector<double
                         >>> &shadows) {
    QVector3D point(x,y,z);
    double diffuse_light_intensity = 0;
      for (size_t i=0; i<ls.size(); i++) {
          QVector3D light_dir      = (ls[i].getPos() - point);
         if (light_dir.length() > shadows[i][x][y])
             continue;
          light_dir.normalize();
          diffuse_light_intensity  += ls[i].getIntensity() * std::
             max(0.f, QVector3D::dotProduct(light_dir,N));
      }
   QVector3D cv(c.red() / 255, c.green() / 255, c.blue() / 255);
   cv = cv * diffuse_light_intensity;
   QColor color = QColor(255 * std::max(0.f, std::min(1.f, cv.x())
      ), 255 * std::max(0.f, std::min(1.f, cv.y())), 255 * std::
      max(0.f, std::min(1.f, cv.z())));

  return color;
}
```

Листинг 3.3 – Реализация алгоритма генерации молнии

```
void Lightning::generateLightning(QPoint &point1, QPoint &point2)
{
    _branches.clear();
    int mx_var = sqrt((point1.x() - point2.x()) * (point1.x() -
       point2.x()) + (point1.y() - point2.y()) * (point1.y() -
       point2.y())) / 2;
    if (mx_var < MIN_DIST) mx_var = 15;
    _branches.push_back(line_t{point1, point2});
    int k = 3 + rand() % 8;
    for(int i = 0; i < k && i < _branches.size(); i++){
        QVector2D p1 = QVector2D(_branches[i].p1.x(), _branches[i
           ].p1.y());
        QVector2D p2 = QVector2D(_branches[i].p2.x(), _branches[i
           ].p2.y());
        _branches.erase(_branches.begin() + i);
        QVector2D mid = (p1 + p2) / 2;
        if ((mid - p1).length() < MIN_DIST){
            _branches.push_back({QPoint(p1.x(), p1.y()), QPoint(p2
               .x(), p2.y())});
            break;
        }
        QVector2D N(p2.y() - p1.y(), -(p2.x() - p1.x()));
        N.normalize();
        bool flag = rand()%2;
        if (flag){
            mid += N * (7 + rand()%5);
            QVector2D newPoint(mid.x() + 2 + rand()%80, mid.y() +
               2 + rand()%80);
            _branches.push_back({QPoint(newPoint.x(), newPoint.y()
               ), QPoint(mid.x(), mid.y())});
        }
        else mid -= N * (7 + rand()%5);
        _branches.push_back({QPoint(p1.x(), p1.y()), QPoint(mid.x
           (), mid.y())});
        _branches.push_back({QPoint(mid.x(), mid.y()), QPoint(p2.x
           (), p2.y())});
        if ((mid - p1).length() < 3 * MIN_DIST) continue;
        i--;
    }
}
```

3.5. Интерфейс программы

Программа запускается с помощью среды разработки *QtCreator* при нажатии на кнопку сборки и запуска в левом нижнем углу.

На рисунке 3.1 представлен пользовательский интерфейс разработанной программы.

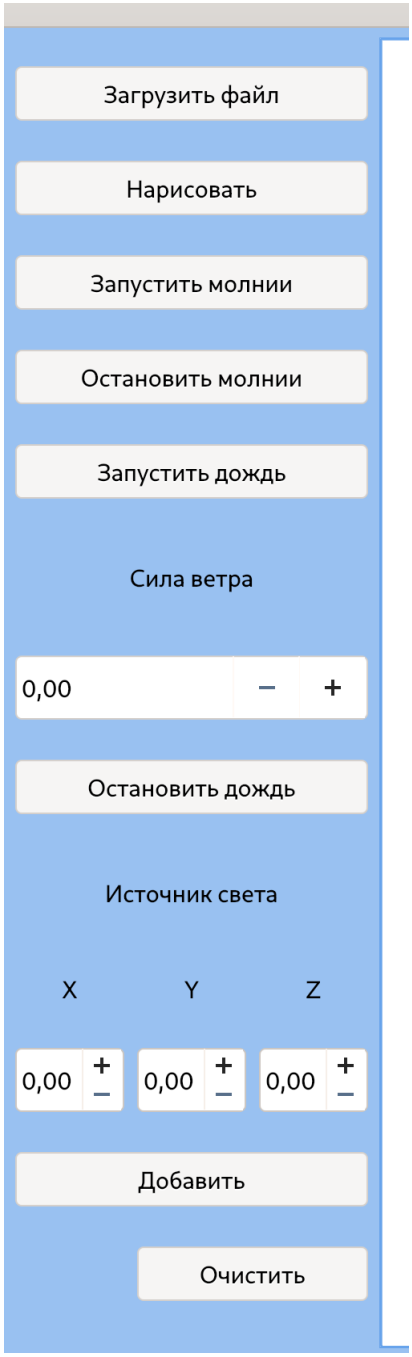

Рисунок 3.1 – Интерфейс программы

3.6. Примеры работы программы

На рисунках 3.2 — 3.8 представлены примеры работы реализованной программы.

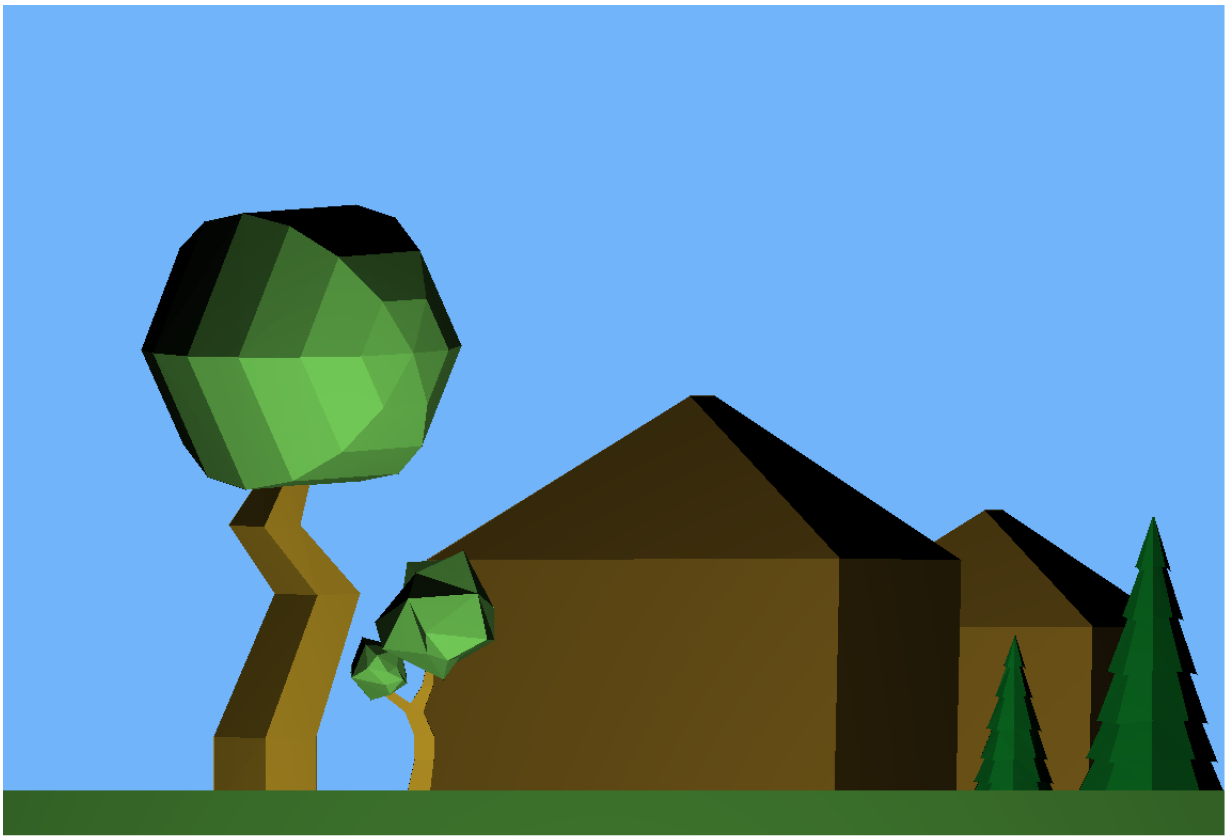

Рисунок 3.2 – Пример работы программы №1

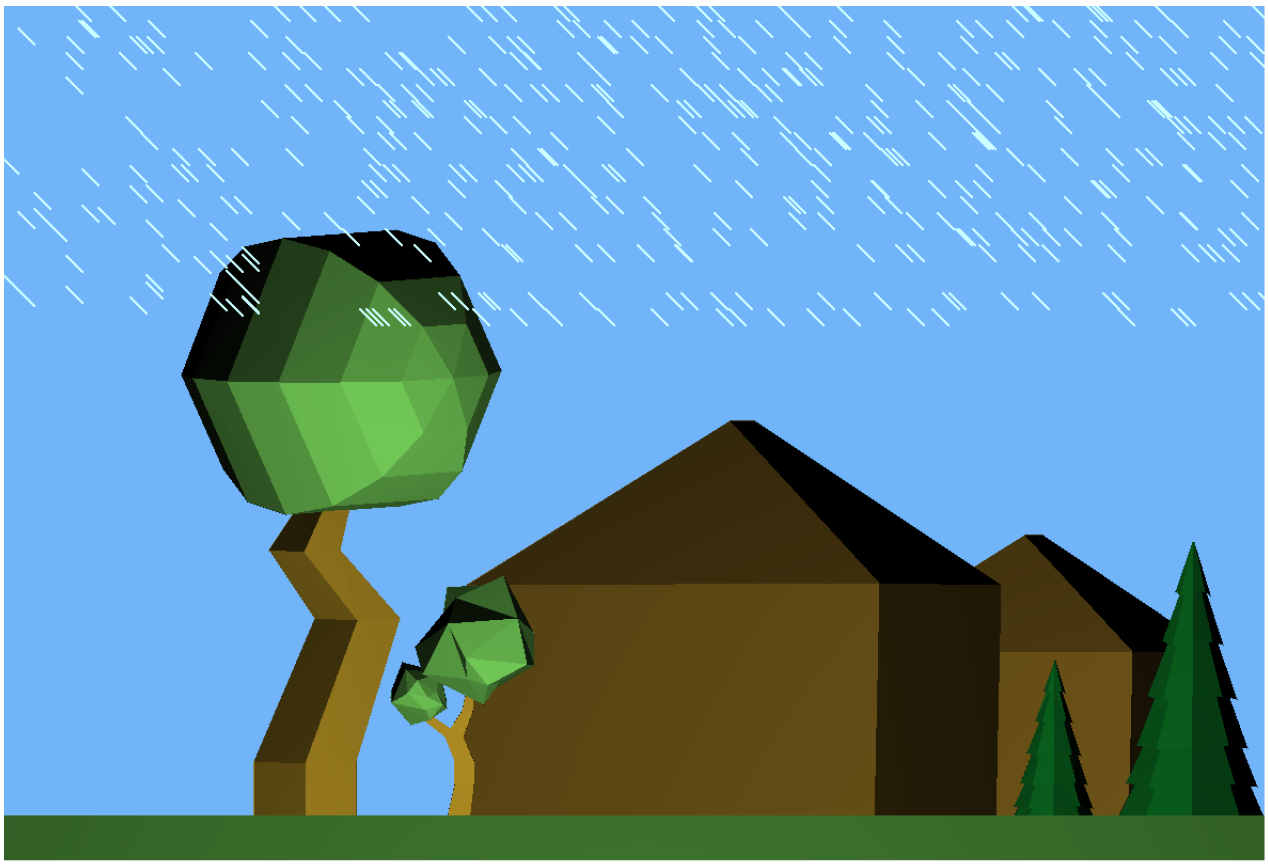

Рисунок 3.3 – Пример работы программы №2

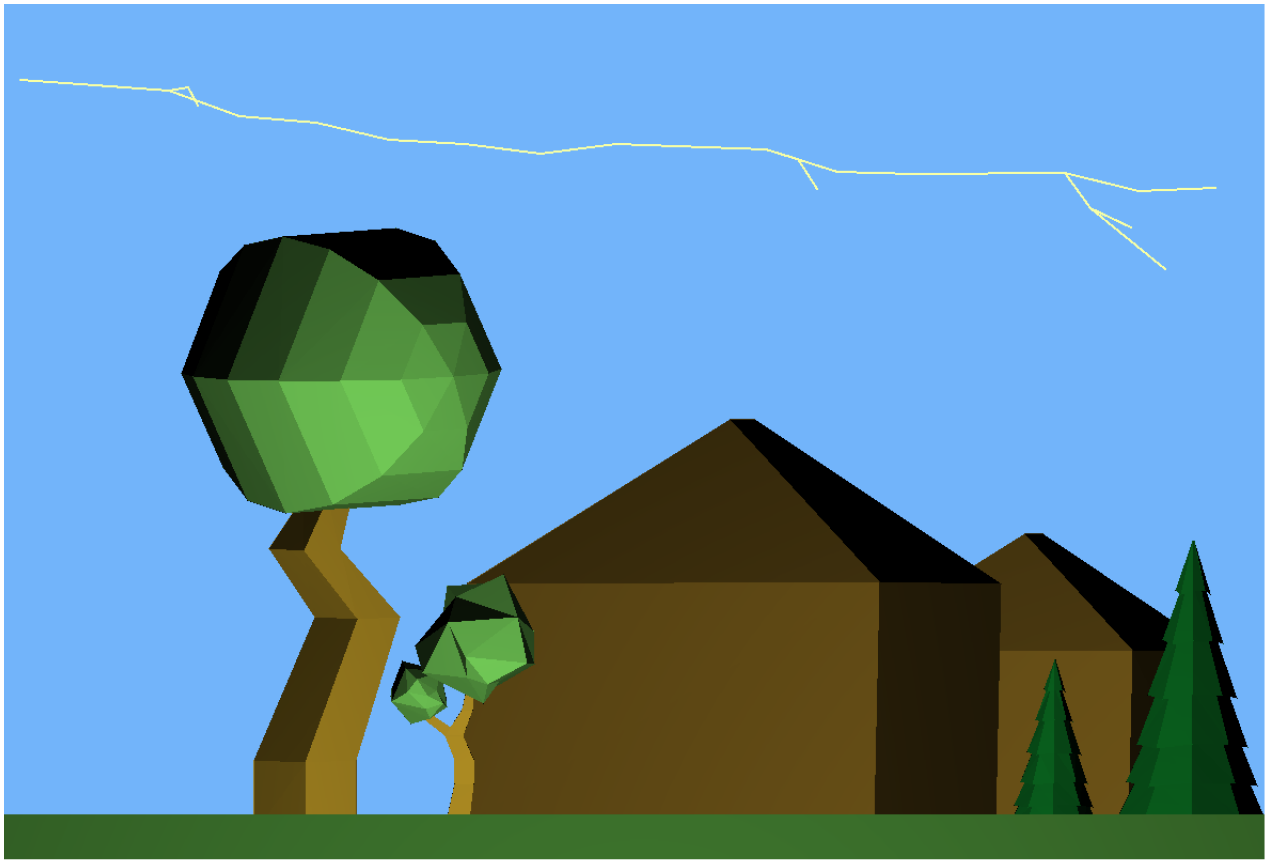

Рисунок 3.4 – Пример работы программы №3

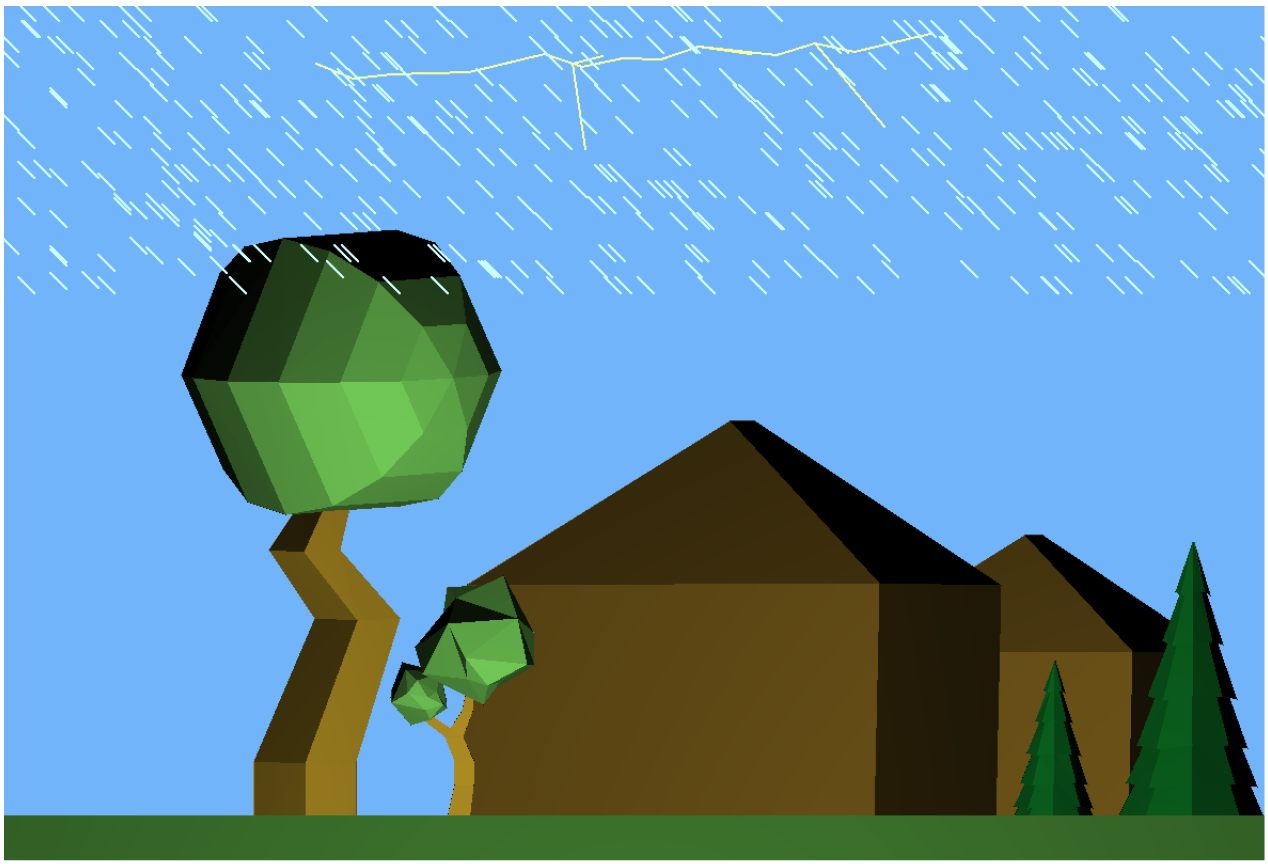

Рисунок 3.5 – Пример работы программы №4

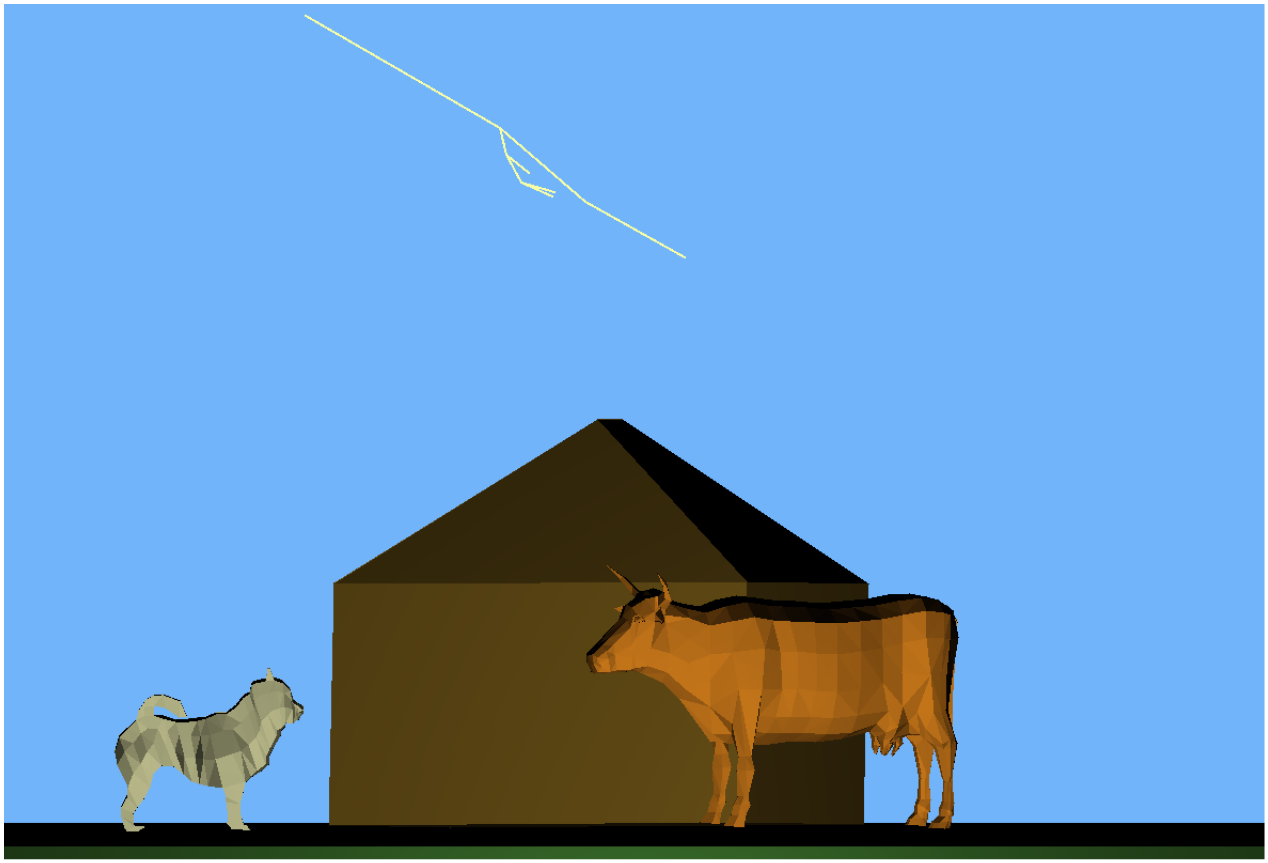

Рисунок 3.6 – Пример работы программы №5

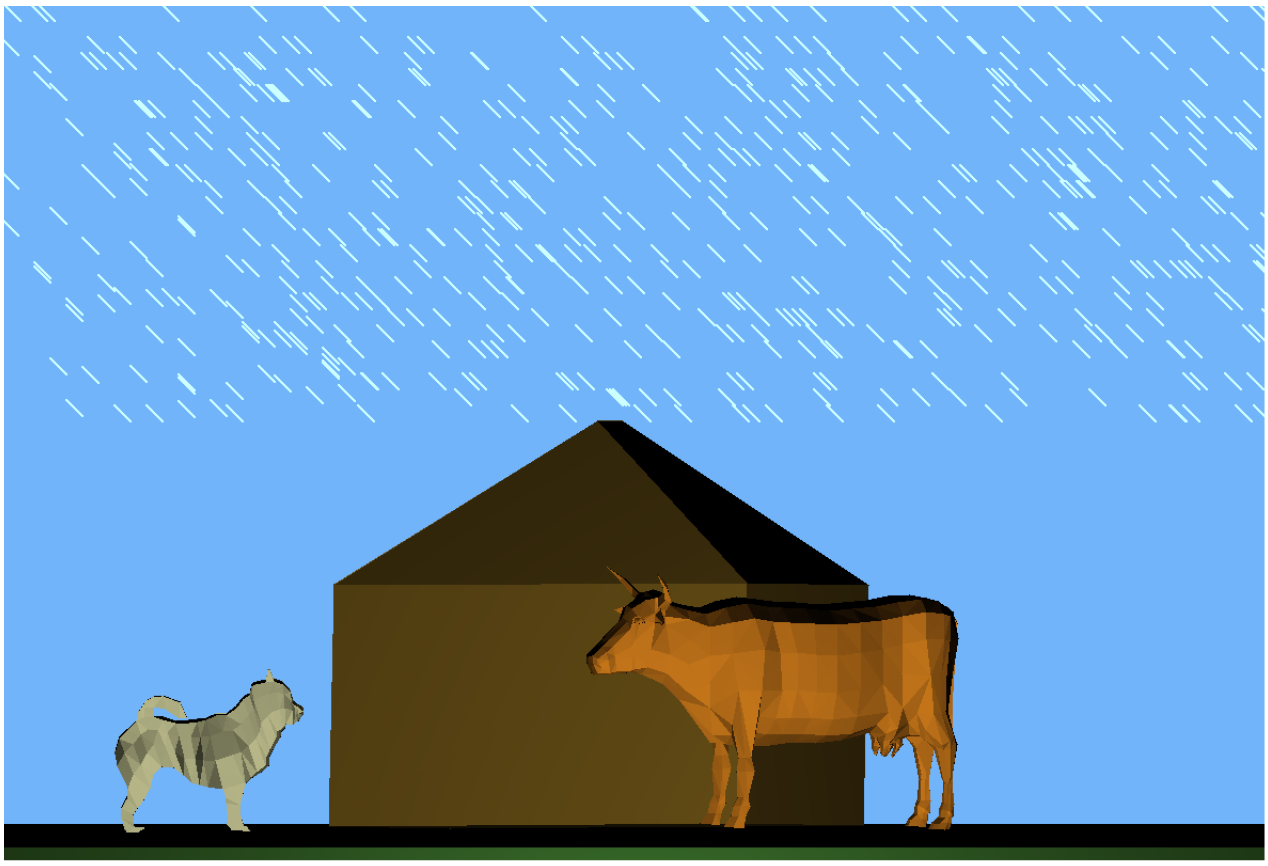

Рисунок 3.7 – Пример работы программы №6

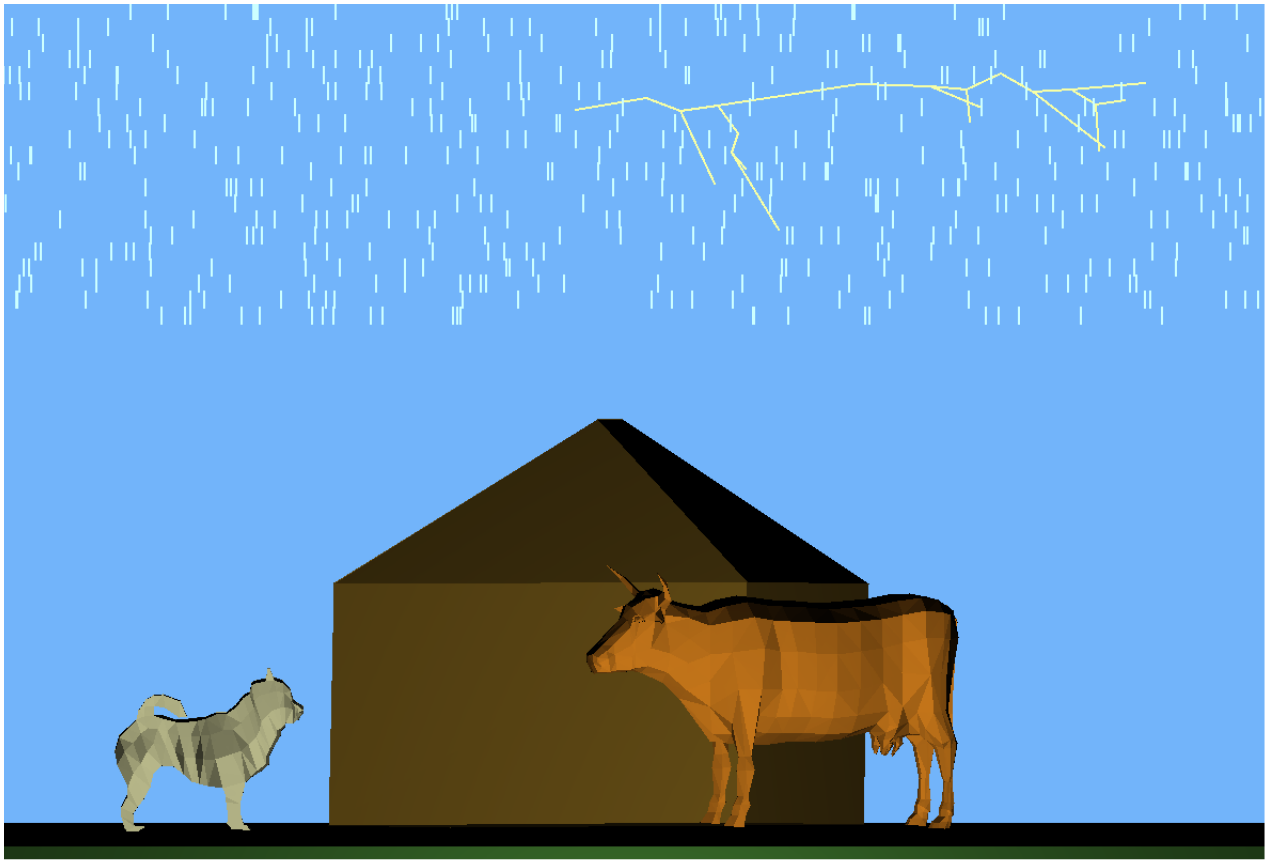

Рисунок 3.8 – Пример работы программы №7

3.7. Вывод

В данном разделе были рассмотрены средства реализации, описаны основные моменты программной реализации, рассмотрен интерфейс программного продукта.

Заключение

В ходе выполнения курсовой работы была достигнута поставленная цель: было спроектированно программное обеспечение для визуализации эффектов погоды в сельской местности.

Для достижения поставленной цели были решены все задачи:

- был проведен анализ и выбор существующих алгоритмов трехмерной графики, которые позволят визуализировать трехмерный объект;
- были выбраны типы и структуры данных;
- был реализованн программный продукт, в частности:
  - был выбран оптимальный метод представления модели;
  - были реализованны выбранные алгоритмы удаления невидимых ребер и поверхностей, алгоритмы закраски полигонов;
  - были реализованны выбранные алгоритмы удаления невидимых ребер и поверхностей, алгоритмы закраски полигонов;
  - был спроектирован интерфейс пользователя.

Список использованных источников

1. Куров А. В. Конспект курса лекций по компьютерной графике. — 2022.
2. Бабич В. Н. Геометризация форм и структур инженерно-строительных объектов и процессов. — Архитектон: известия вузов. — 2016.
3. Mario Botsch, Mark Pauly, Leif Kobbelt, Pierre Alliez, Bruno Lévy, et al. Geometric Modeling Based on Polygonal Meshes. — 2007.
4. Способы представления и размещения трёхмерных моделей для прототипирования ювелирных изделий : Материалы VI Международной научно-практической конференции (школы-семинара) молодых ученых. — 2020 — 840-845с.
5. Роджерс Д., Алгоритмические основы машинной графики. — 2013 — 457-517с.
6. Цапко И. В., Цапко С. Г. Алгоритмы и методы обработки информации в задачах трехмерного сканирования объектов. — Известия Томского политехнического университета. Инжиниринг георесурсов. — 2010.
7. Янова Р. Ю. Метод прямой и обратной трассировки. — Вестник науки и образования. — 2016.
8. Карабчевский В. В., Лунтовская А. А. Реалистичная визуализация атмосферных осадков в приложениях реального времени. — 2015.
9. Kshitiz Garg, Shree K. Nayar. Photorealistic Rendering of Rain Streaks. — 2003.
10. Справочник по языку C++ [Электронный ресурс]. Режим доступа: https://learn.microsoft.com/ru-ru/cpp/cpp/cpp-languagereference?view=msvc-170 (дата обращения: 01.02.2023).
11. Документация библиотеки Qt [Электронный ресурс]. Режим доступа: https://doc.qt.io (дата обращения: 27.12.2022).
12. An Overview of 3D Data Content, File Formats and Viewers [Электронный ресурс]. Режим доступа:

http://isda.ncsa.illinois.edu/peter/publications/techreports/2008/NCSA-ISDA-2008-002.pdf (дата обращения: 27.12.2022).

13. Comparison of different image formats using LSB Steganography [Электронный ресурс]. Режим доступа: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=arnumber=8269657 (дата обращения: 27.12.2022).

ПРИЛОЖЕНИЕ А