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 ИУ «Информатика и системы управления»

КАФЕДРА ИУ-7 «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Визуализация озера с растительностью и фламинго»

Студент ИУ7-54Б _____ Пронина Л.Ю.
(Группа) (Подпись, дата) (Фамилия И.О.)

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

2023 г.

Оглавление

ВВЕДЕНИЕ

Цель данной работы — разработка программного обеспечения для построения трехмерной сцены и визуализации озера с растительностью и фламинго.

Для достижения поставленной цели требуется выполнить следующие задачи:

1) формализовать задачу;
2) провести анализ существующих алгоритмов компьютерной графики: удаления невидимых линий и поверхностей, построения теней, закраски и освещения;
3) спроектировать программное обеспечение для построения трехмерной сцены и визуализации озера с растительностью и фламинго;
4) выбрать средства реализации спроектированного программного обеспечения и разработать его;
5) исследовать характеристики разработанного программного обеспечения.

1. Аналитическая часть

В данной части проводится анализ объектов сцены и существующих алгоритмов построения изображений, а также выбор более подходящих из них для решения поставленной задачи.

1.1. Формализация задачи

1.1.1. Описание объектов сцены

Сцена разрабатываемого программного обеспечения состоит из следующих объектов:

1) Источник света — невидимый точечный объект, который описан тремя координатами положения и коэффициентом освещенности.

2) Источник света на бесконечности — задает фоновое освещение, чтобы при отсутствии точечных источников света было видно сцену.

3) Фламинго — невыпуклый объект, предварительно заданной формы, который характеризуется множествами точек и полигонов. Для каждого множества определен материал, который задается цветом и оптическими характеристиками.

4) Растительность — множество точек и полигонов, которые определяют каждый куст на сцене.

5) Объекты сцены — тела, которые представляются множеством точек в пространстве и полигонов. Каждое тело задается определенными характеристиками, такими как цвет, коэффициенты рассеянного, диффузного и зеркального отражения, коэффициент пропускания и преломления.

1.1.2. Формализация процесса создания видео

Входными данными являются количество и расположение каждого точечного источника света, количество и расположение фламинго, плотность

растительности на озере и оптические свойства поверхностей. Выходными данными является последовательность изображений размером 1000×900 с визуализацией фламинго, которые ходят по озеру. Ограничения на входные данные:

1) расположение каждого точечного источника света задается 3 координатами (x, y, z), каждая из которых является вещественным числом;

2) расположение каждого фламинго задается 2 координатами (x, y), каждая из которых является вещественным числом, а так же точка, определяемая этими координатами, должна находиться внутри сцены $(0 \leq x \leq$ ширина сцены, $0 \leq y \leq$ высота сцены$)$;

3) плотность растительности и оптические свойства поверхностей являются вещественными числами, каждое от 0 до 1.

Формализованная постановка задачи создания последовательности кадров в виде IDEF0 диаграммы 0 уровня изображена на рисунке (1.1).

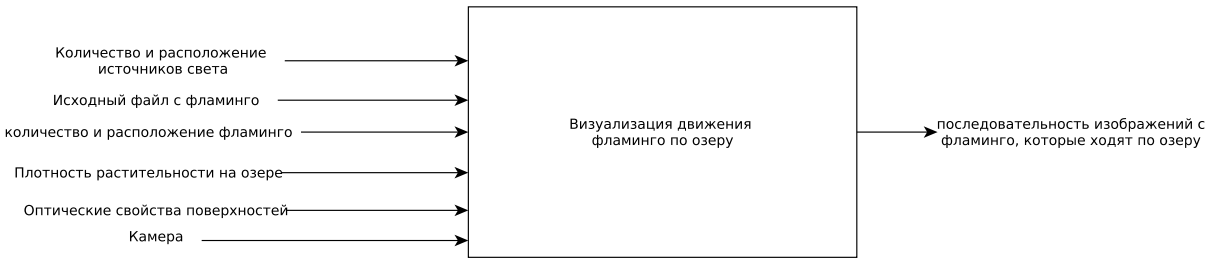

Рисунок 1.1 – IDEF0 диаграмма, 0 уровень

1.1.3. Требования к программному обеспечению

К разрабатываемому программному обеспечению будут предъявлены следующие требования:

1) наличие графического интерфейса с функционалом, который имеет возможность задать входные данные;

2) программа принимает входные данные, учитывая ограничения, указанные выше, иначе выдает ошибку;

3) программа может сохранять полученное изображение в файл с форматом .png;

4) программа предоставляет возможность запустить непрерывную отрисовку кадров, чтобы можно было наблюдать ходьбу фламинго на них;

1.2. Обзор способов задания трехмерной модели

Модель – отображение форм и размеров объекта [1].

Разделяют три вида модели: каркасная, поверхностная и твердотельная [1] [2] [3].

1) Каркасная модель.

   В этой модели хранится информация только о вершинах и рёбрах объектов. Недостатком данной модели является то, что она может неправильно передавать форму объекта.

2) Поверхностная модель.

   Поверхность может описываться аналитически или полигональной сеткой. Такая информационная модель содержит данные только о внешних геометрических параметрах объекта. Недостатком модели является отсутствие информации о том, с какой стороны поверхности находится материал.

3) Твердотельная модель.

   При твердотельном моделировании учитывается еще и материал, из которого изготовлен объект. С помощью указания направления внутренней нормали к информации о поверхности добавялется то, с какой стороны поверхности расположен материал.

Для достижения поставленной цели лучше всего подходит поверхностная модель, так как она позволит достичь требуемой визуализации озера с растительностью и фламинго, не требуя большого количества вычислительных операций, как твердотельная.

1.3. Способы задания поверхностных моделей

Существует 2 основных способа для задания поверхностной модел [2]:

1) Аналитический способ — характеризуется тем, что для получения поверхности нужно дополнительно вычислять функцию, зависящую от параметра.

2) Полигональная сетка — характеризуется тем, что информация о модели хранится в виде совокупности вершин, ребер и граней.

Из двух представленных вариантов наиболее оптимальным является использование полигональной сетки, так как такой вариант задания поверхности позволит быстро выполнять операции над объектами.

Существует несколько способов хранения полигональной сетки [2]:

1) Список граней — характеризуется множеством граней и вершин. В каждую грань входят как минимум три вершины.

2) Вершинное представление — характеризуется хранением информации о вершинах, которые указывают на другие вершины, с которыми они соединены.

3) Таблица углов — это таблица, хранящая вершины. Ее обход неявно задаёт полигоны. Такое представление занимает меньше места и более производительно для нахождения полигонов, но операции по их изменению медленные.

4) «Крылатое» представление — характеризуется заданием точек, каждая из которых указывает на 2 вершины, 2 грани и 4 ребра, которые ее касаются, благодаря чему обход поверхностей выполняется за постоянное

время. Однако метод требует много памяти и изменения геометрических характеристик приводит к формированию списка индексов граней. Данный способ полезен для определения столкновений объектов.

Для хранения полигональной сетки будет использоваться список граней, так как у этого способа простое и наглядное хранение информации о гранях объекта, а также этот способ обеспечивает простой доступ к геометрическим параметрам и связям с другими гранями, что упрощает операции по обходу, изменению и анализу поверхности объекта.

1.4. Анализ алгоритмов удаления невидимых линий и поверхностей

Представленные далее алгоритмы работают в одном из двух пространств: изображения или объекта [2]. Пространство изображения — это пространство, в котором каждая точка представляет отдельный пиксель или элемент изображения. Пространство объекта — это пространство, в котором располагаются объекты в трехмерной форме.

Будут рассмотрены 4 основных алгоритма для решения задачи удаления невидимых линий и поверхностей: использующий z-буфер, Робертса, обратной трассировки лучей и Варнока [2].

1.4.1. Алгоритм, использующий Z-буфер

Данный алгоритм работает в пространстве изображения [2].
Используются два буфера:

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

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

Пошаговое описание алгоритма Z-буфера:

1) создаются 2 буфера: буфер кадра и буфер глубины (Z-буфер);

2) задаются начальные значения для каждого пикселя в буфере глубины;

3) для каждого полигона сцены вычисляется нормаль к нему;

4) вычисляется глубина $z(x, y)$ для каждого пикселя с координатами (x, y);

5) сравнивается глубина $z(x, y)$ с глубиной $z_{buf}(x, y)$, которая хранится в буфере;

6) если $z(x, y) < z_{buf}(x, y)$, то значение в буфере заменяется $z_{buf}(x, y) = z(x, y)$.

Главной особенностью алгоритма является его простота реализации и высокая скорость обработки объектов. При этом использование памяти для буферов изображения для современных компьютеров является незначительным и не вызывает проблем.

1.4.2. Алгоритм Робертса

Данный алгоритм работает в пространстве объекта. Для классического алгоритма Робертса требуется, чтобы все тела были выпуклыми. Так как в ходе текущей работы нужно будет работать с невыпуклыми телами, то они предварительно должны быть разбиты на выпуклые части [2].

Алгоритм состоит из 3 этапов.

1) Во-первых, происходит подготовка исходной матрицы V, представляющей информацию о каждом теле. Матрица имеет размерность $4 * n$, где n — количество граней тела, и содержит коэффициенты уравнений плоскостей, проходящих через грани.

2) Во-вторых, выполняется удаление ребер, которые закрыты самим телом. Для этого вектор взгляда E умножается на матрицу V, и отрицательные компоненты вектора указывают на невидимые грани.

3) В-третьих, происходит удаление невидимых ребер, закрытых другими телами на сцене. Для определения невидимых точек ребра, строится

луч, соединяющий точку наблюдения с точкой на ребре. Если луч проходит сквозь тело, то ребро невидимо.

Главным недостатком данного алгоритма является его вычислительная сложность, которая составляет $O(n^2)$, где n - количество объектов на сцене. Также все тела на сцене должны быть выпуклыми, что требует дополнительных проверок. Однако, работа в объектном пространстве и высокая сложность вычислений обеспечивают высокую точность результата.

1.4.3. Алгоритм трассировки лучей

Алгоритм работает в пространстве изображений. Основная идея заключается в анализе лучей, попадающих в наблюдателя от объектов. Однако, из-за того, что не все лучи достигают наблюдателя, было предложено рассматривать лучи в обратном направлении - от наблюдателя к объектам. Трассировка лучей включает отражение, преломление и прохождение лучей через поверхности объектов. Она завершается, когда луч пересекает поверхность непрозрачного объекта, видимого для наблюдателя [2].

Однако, данный метод требует множества вычислений, так как необходимо найти пересечение каждого луча с объектами сцены. Количество лучей пропорционально размеру изображения, что сказывается на производительности и скорости алгоритма. Таким образом, данный метод не подходит для решения моей задачи, где требуется быстрая отрисовка динамических сцен.

1.4.4. Алгоритм Варнока

Алгоритм Варнока оперирует в пространстве изображения и позволяет определить, какие грани или части граней объектов сцены видимы, а какие скрыты другими объектами [2].

Основная идея алгоритма заключается в разделении области изображения на более мелкие окна. Для каждого окна определяются связанные многоугольники, а затем определяется их видимость на сцене.

В алгоритме обычно используются выпуклые многоугольники в качестве граней, так как эффективность работы с ними выше, чем с произвольными многоугольниками.

Окно, в котором требуется отобразить сцену, должно быть прямоугольным. Алгоритм работает рекурсивно: на каждом шаге происходит анализ видимости граней, и если для определения видимости требуются дополнительные вычисления, то окно делится на 4 части, и анализ повторяется отдельно для каждой части.

Главный недостаток данного алгоритма состоит в том, что использование рекурсии может привести к большому числу шагов рекурсии и значительной вычислительной сложности.

Вывод

В качестве алгоритма удаления невидимых рёбер и поверхностей был выбран алгоритм Z-буфера, так как он работает в пространстве изображения и быстро производит вычисления.

Таблица 1.1 – Сравнение алгоритмов удаления невидимых линий

Критерии	Алгоритмы			
	Использующий Z-буфер	Робертса	Обратной трассировки лучей	Варнока
Вычислительная трудоемкость (n — кол-во граней объектов, N — кол-во пикселей)	O($N * n$)	O(n^2)	O($N * n$)	O($N * n$)
Работает с невыпуклыми	да	нет	да, с доп. проверками	да, с доп. обработкой сцены
Рабочее пространство	изображение	объект	изображение	изображение
Применение для сцен в реальном времени	может быть эффективным, но потреблять больше памяти	используется, но может быть не эффективным для сложных сцен	обеспечивает высокое качество изображений, но может быть более затратным по времени для рендеринга сложных сцен	используется и позволяет обрабатывать большие сцены за счет эффективной сортировки и однократного прохода по полигонам

1.5. Анализ алгоритмов закраски

Существует два наиболее распространённых метода закраски [2].

1.5.1. Закраска Гуро

Метод Гуро, основанный на интерполяции интенсивности, заключается в закрашивании разных точек грани с разными значениями интенсивности. Для этого вычисляется вектор нормали в каждой вершине грани, а затем значения интенсивности интерполируются по всем точкам примыкающих граней [2].

Закрашивание граней по методу Гуро выполняется в четыре этапа.

1) Вычисляются нормали к каждой грани.

2) Определяются нормали в вершинах, которые вычисляются как усреднение нормалей примыкающих граней.

3) На основе нормалей в вершинах вычисляются значения интенсивностей с учетом выбранной модели отражения света.

4) Полигоны граней закрашиваются цветом, соответствующим линейной интерполяции значений интенсивности в вершинах.

Метод Гуро применим только для небольших граней, которые находятся на значительном расстоянии от источника света. Если размер грани большой, расстояние от источника света до центра грани будет меньше, чем до вершин, что должно привести к более яркой освещенности центра грани по сравнению с ребрами, но из-за линейного закона интерполяции, используемого в методе, это не удается достичь, что приводит к неестественной освещенности в некоторых участках грани.

1.5.2. Закраска Фонга

Закраска Фонга, подобно закраске Гуро, осуществляет интерполяцию, но в отличие от метода Гуро, в методе Фонга интерполируются векторы нор-

малей, а затем используются для определения значения интенсивности для каждой точки [1].

Процесс закраски в методе Фонга включает следующие этапы:

1) Вычисляются нормали к граням.

2) На основе нормалей к граням определяются нормали в вершинах грани. Используя эти нормали, для каждой точки закрашиваемой грани вычисляется интерполированный вектор нормали.

3) Направление векторов нормали используется для определения цвета точек грани в соответствии с выбранной моделью отражения света.

Метод Фонга требует больших вычислительных затрат по сравнению с методом Гуро, однако он обеспечивает более точное приближение кривизны поверхности и, следовательно, позволяет получить более реалистичное изображение.

Вывод

В данной работе будет использоваться метод закраски Фонга, так как он дает наиболее реалистичное изображение, в частности зеркальных бликов.

1.6. Анализ алгоритмов построения теней

В рассмотренном алгоритме трассировки лучей, тени создаются в процессе выполнения алгоритма: пиксели затеняются, если луч пересекает объект, но не достигает источника света. При использовании алгоритма с z-буфером, для нахождения теней добавляется дополнительной теневой z-буфер, основанный на точке наблюдения, которая совпадает с источником света. Такая модификация позволяет избежать усложнения структуры программы и, следовательно, сократить время отладки программного продукта.

1.7. Анализ моделей освещения

Модели освещения позволяют определить, как свет будет влиять на объекты в сцене. Основные виды освещений представлены далее [2].

1) Фоновое освещение — это равномерное и небольшое освещение, которое создается отражением света от окружающих поверхностей. Оно дает объектам общее освещение и предотвращает полное затемнение в тени.

2) Диффузное освещение — это освещение, которое происходит, когда свет попадает на поверхность объекта и равномерно рассеивается во все стороны. Оно создает матовый эффект освещения и отображает интенсивность света в зависимости от угла падения на поверхность.

3) Зеркальное освещение — это освещение, которое создает блеск и блики на поверхности объекта. Оно происходит, когда свет падает на гладкую и отражающую поверхность, и отображает интенсивность света в зависимости от угла отражения.

Введем следующие обозначения для коэффициентов:

— I_a — интенсивность рассеянного света;

— K_a — коэффициент диффузного отражения рассеянного света;

— I_d — интенсивность диффузного освещения (diffuse),

— K_d — коэффициент отражения диффузного освещения,

— I_s — интенсивность зеркального освещения,

— K_s — коэффициент отражения зеркального освещения,

— n — бликовый коэффициент;

— $\vec{L}$ — вектор от точки к источнику;

— $\vec{N}$ — вектор нормали;

— $\vec{R}$ — вектор отраженного луча;

— $\vec{V}$ — вектор от точки к наблюдателю.

Коэффициенты K_a, K_d и K_s задаются отдельно для каждого материала и определяют его отражательные свойства для фонового, диффузного и зеркального освещения соответственно. Чем больше значение этих коэффициентов, тем сильнее будет отражаться соответствующий тип освещения от поверхности.

Коэффициент n также определяется для каждого материала и отвечает за бликовость поверхности. Чем больше значение n, тем более "острые"и интенсивные будут зеркальные блики.

1.7.1. Модель Ламберта

Модель Ламберта моделирует диффузное освещение [2].
Свет от точечного источника отражается по закону Ламберта (1.1).

$$I = I_i \cdot k_d \cdot (\vec{L}, \vec{N}), \tag{1.1}$$

Модель Ламберта является простой в реализации моделью, однако основной недостаток — одинаковая интенсивность во всех точках, принадлежащих одной грани.

1.7.2. Модель Фонга

Модель Фонга является моделью освещения, которая комбинирует фоновое освещение с диффузной и зеркальной составляющими. Благодаря зеркальному отражению на блестящих предметах появляются световые блики [2].

Коэффициент зеркального отражения зависит от угла падения, однако даже при перпендикулярном падении зеркально отражается только часть света, а остальное либо поглощается, либо отражается диффузно. Эти соотношения определяются свойствами вещества и длиной волны. Объединяя эти

результаты с формулой рассеянного света и диффузного отражения, получим модель освещения (1.2).

$$I = I_a * K_a + I_d * K_d * (\vec{L} \cdot \vec{N}) + I_s * K_s * (\vec{R} \cdot \vec{V})^n, \quad (1.2)$$

Вывод

Была выбрана модель освещения Фонга, так как была поставлена задача реалистичной визуализации, а для нее нужно не только диффузное освещение, как в модели Ламберта.

1.8. Визуализация ходьбы фламинго

Для визуализации ходьбы фламинго нужно расчитать движение соответствующих вершин полигонов, которые входят в бедро и голень. У фламинго при ходьбе сначала одна нога стоит на месте, а другая поднимается и опускается, затем вторая стоит, а первая поднимается. И этот процесс повторяется уже для противоположных ног.

Введем следующие обозначения:

1) x_{start} — начальная координата точки по оси OX;

2) y_{start} — начальная координата точки по оси OY;

3) z_{start} — начальная координата точки по оси OZ;

4) x_{end} — конечная координата точки по оси OX;

5) y_{end} — конечная координата точки по оси OY;

6) z_{end} — конечная координата точки по оси OZ;

7) t_{full} — полное время совершения рассматриваемой части шага;

8) t — текущий момент времени (от 0 до t_{full}).

Для задания движения точки в трехмерном пространстве необходимо использовать формулу (1.3) [4].

$$\begin{aligned} x(t) &= x_{start} + (x_{end} - x_{start}) \cdot \left(\frac{t}{t_{full}}\right); \\ y(t) &= y_{start} + (y_{end} - y_{start}) \cdot \left(\frac{t}{t_{full}}\right); \\ z(t) &= z_{start} + (z_{end} - z_{start}) \cdot \left(\frac{t}{t_{full}}\right), \end{aligned} \tag{1.3}$$

где $x(t), y(t), z(t)$ — координаты точки в момент времени t.

Эта формула позволяет рассчитывать координаты точки в трехмерном пространстве в любой момент времени от начального до конечного положения.

1.9. Вывод к аналитической части

Для задания трёхмерных моделей была выбрана поверхностная модель, она будет задаваться полигональной сеткой. В качестве алгоритма удаления невидимых линий был выбран алгоритм, использующий Z-буфер, построение теней будет выполняться с помощью модифицированного алгоритма z-буфера, освещение и закраска с помощью алгоритма Фонга.

2. Конструкторская часть

В этой части будут представлены требования к программному обеспечению, описание структур данных и схемы выбранных алгоритмов.

2.1. Описание структур данных

В разрабатываемом программном обеспечении будут использоваться следующие структуры данных:

1) сцена состоит из списка объектов и списка источников света;
2) каждый объект сцены задается списком его вершин и полигонов;
3) полигон включает в себя индексы 3 точек, цвет и оптические свойства поверхности, которую задает этот полигон;
4) цвет состоит из 4 чисел;
5) источник света определяется 3 координатами, которые задают положение в пространстве, и интенсивностью света.

2.2. Общий алгоритм визуализации трехмерной сцены

Формализованная модель разрабатываемого программного обеспечения в виде IDEF0 диаграммы 1 уровня изображена на рисунке 2.1.

2.3. Алгоритм, использующий z-буффер

Схема алгоритма представлена на рисунке 2.2.

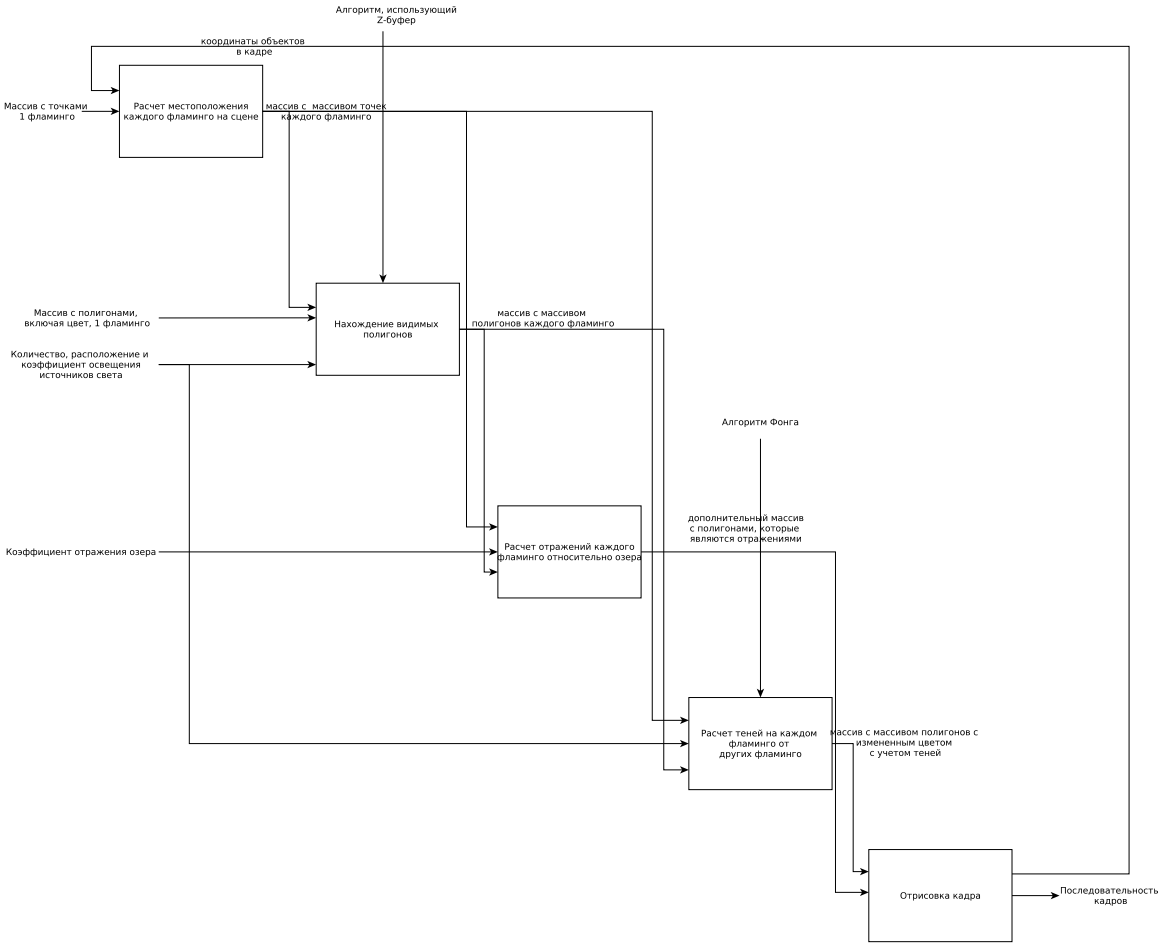

Рисунок 2.1 – IDEF0 диаграмма, 1 уровень

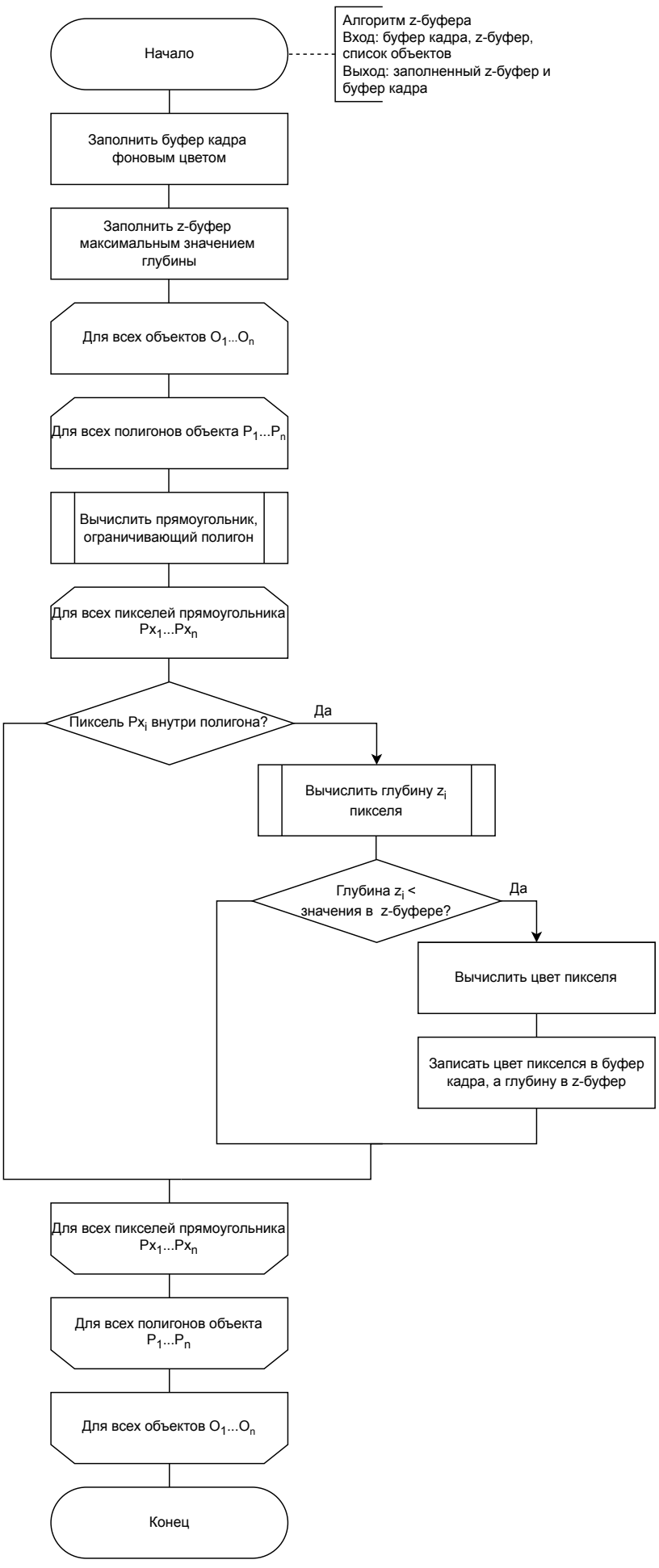

Рисунок 2.2 – Схема алгоритма, использующего z-буфер

2.4. Модификация алгоритма, использующего z-буффер, для нахождения теней

Схема модицифицированного алгоритма, использующего z-буффер, для нахождения теней представлена на рисунке 2.3.

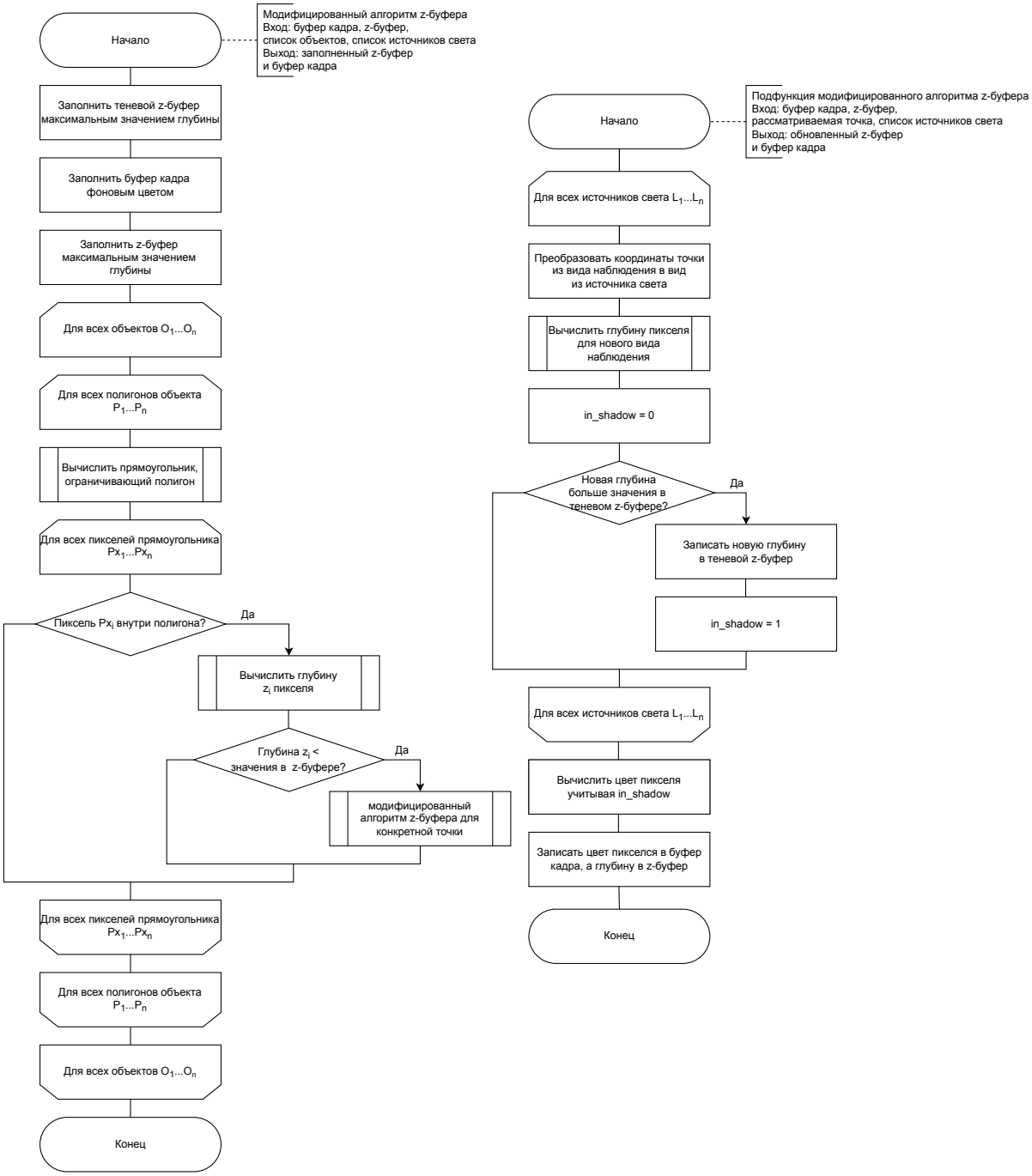

Рисунок 2.3 – Схема модифицированного алгоритма, использующего z-буфер

2.5. Алгоритм для отображения отражений

Точка P и ее отраженная относительно плоскости отражения точка P' связаны следующим выражением:

$$P' = P - 2\frac{N \cdot P}{\|N\|^2} \cdot N, \tag{2.1}$$

где N - вектор нормали к плоскости отражения [3].

Схема алгоритма для расчета отражений представлена на рисунке 2.4.

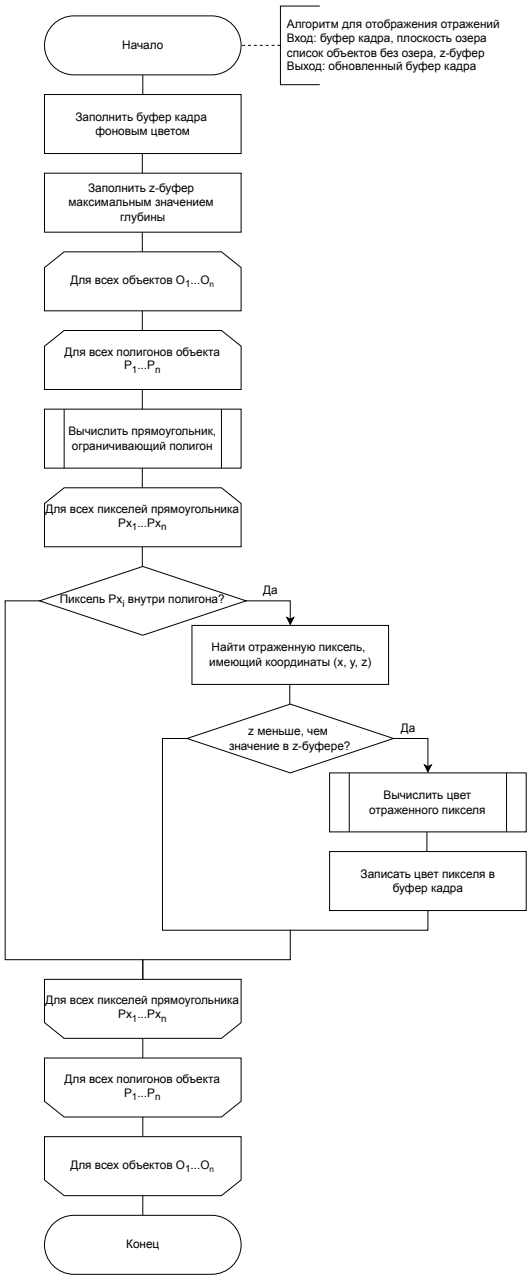

Рисунок 2.4 – Схема алгоритма для отображения отражений

2.6. Выводы к конструкторской части

Было спроектировано программное обеспечение для построения трехмерной сцены и визуализации озера с растительностью и фламинго.

3. Технологическая часть

3.1. Средства реализации

В качестве языка программирования был выбран *C++* [5], так как в стандартной библиотеке языка присутствует поддержка всех структур данных, выбранных по результатам проектирования, а так же средствами языка можно реализовать все алгоритмы, выбранные в результате проектирования.

В качестве среды разработки была выбрана среда $QtCreator$ вместе с Qt [6], так как у него есть встроенные функции для создания пользовательского интерфейса и встроенный отладчик. Для тестирования был выбран фреймворк *GoogleTest* [7], так как я уже использовала его ранее и он предоставляет достаточный функционал для написания тестов для программы на *C++*.

Для измерения времени будут использоваться функции $std :: chrono :: system\_clock :: now(...)$ и $std :: chrono :: duration\_cast < std :: chrono :: milliseconds >$ из библиотеки $chrono$ [8].

В качестве формата входного файла с фламинго был выбран формат obj [9], так как это распространенный формат для хранения 3D объектов и его обработка не требует большого количества вычислительных действий или использования дополнительных библиотек.

3.2. Описание структуры программы

На рисунке (3.1) изображена диаграмма классов программы.

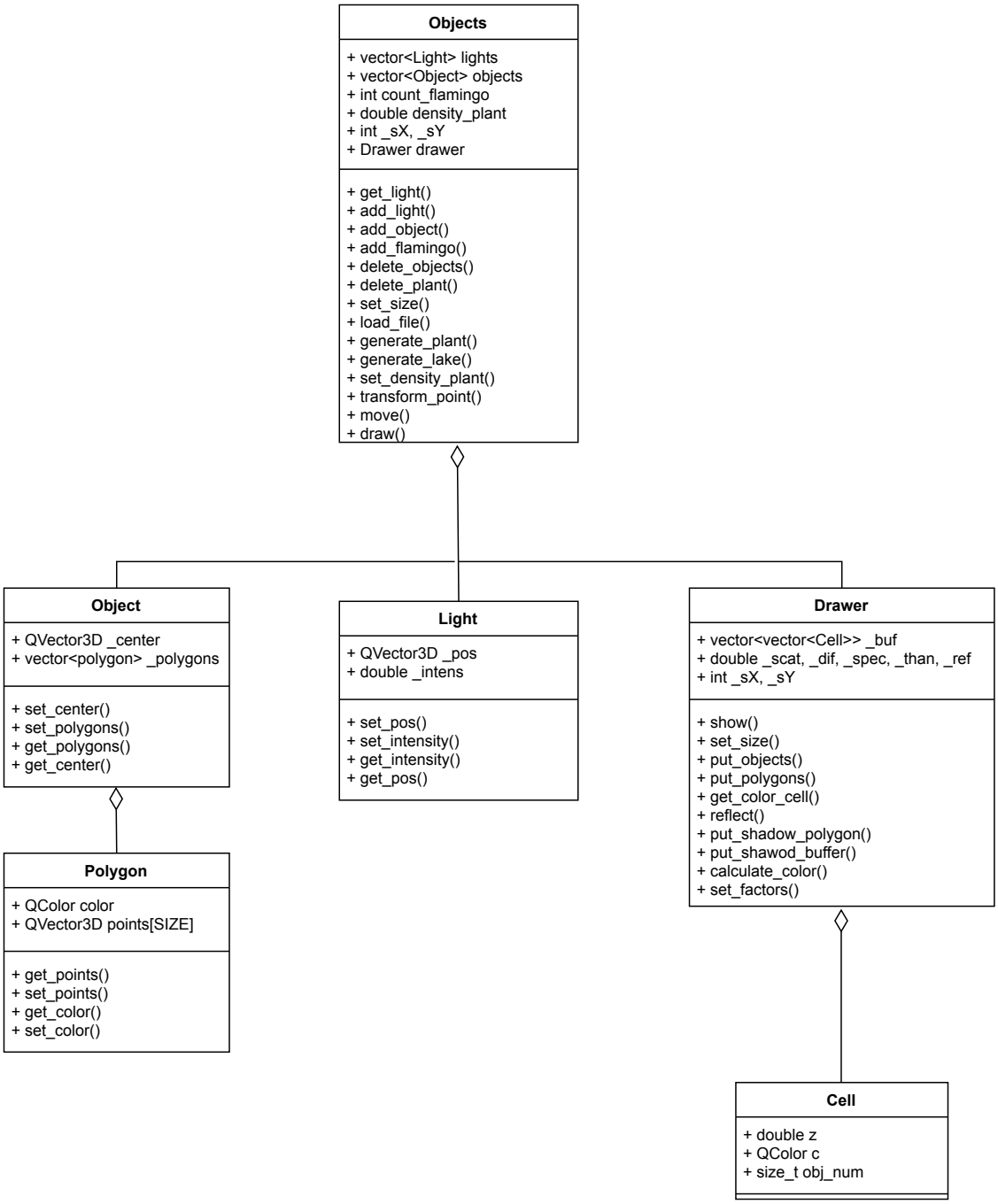

Рисунок 3.1 – Диаграмма классов

3.3. Графический интерфейс программы

Пример пользовательского интерфейса представлен на рисунке (3.2).

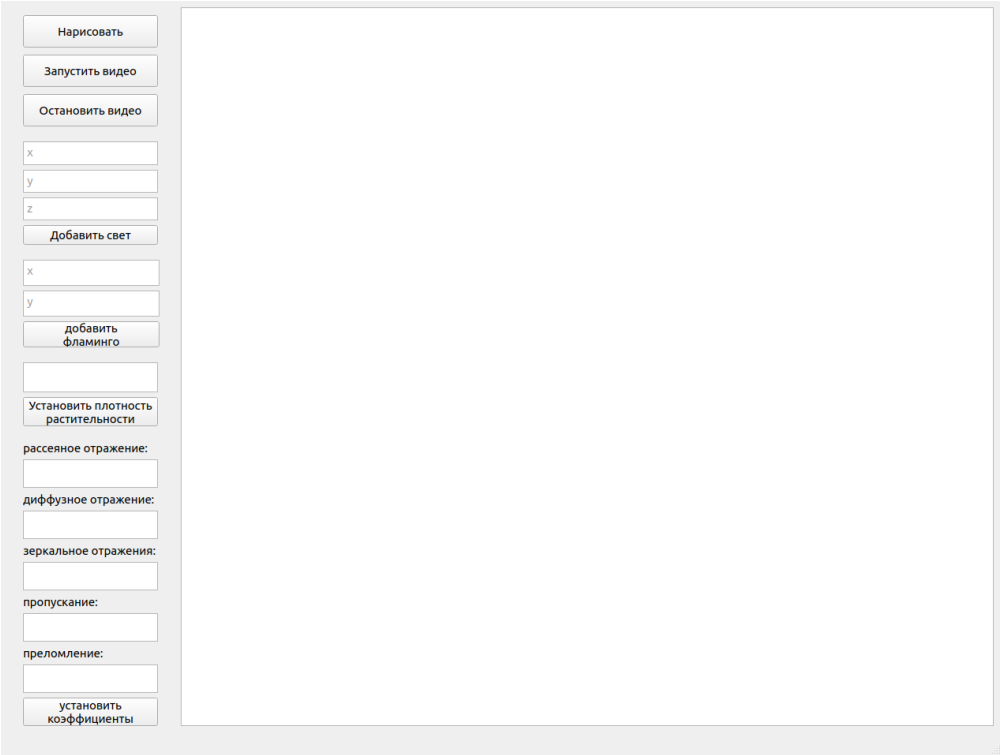

Рисунок 3.2 – Пример интерфейса разработанной программы

3.4. Модульное тестирование

Для модульного тестирования был использован фреймворк $GoogleTest$. В качестве меры, используемой при тестировании программного обеспечения выбрано покрытие кода.

Используя *Google Test* покрытие считается автоматически, при включении соответствующих флагов, после выполнения тестирования. В данной работе создан набор модульных тестов, обеспечивающий покрытие, равное 40%.

Примеры модульных тестов, написанных для данной работы представлены в листингах 3.1–3.3.

Листинг 3.1 – Тест для функции, которая определяет принадлежность точки полигону

```
TEST(IsInsideTest, IsInsideTest) {
  drawer dr;
  std::vector<QVector3D> points = {
    QVector3D(0, 0, 0),
    QVector3D(2, 1, 0),
    QVector3D(3, 5, 0)
  };
  int x = 1;
  int y = 2;
  bool result = dr.is_inside(x, y, points);
  EXPECT_TRUE(result);
}
```

Листинг 3.2 – Тест для функции, которая определяет прямоугольник, в который вписан полигон

```
TEST(DrawerTest, BorderTest2) {
  drawer dr = drawer(100, 100);
  std::vector<QVector3D> points = {
    QVector3D(-23, 0, 13),
    QVector3D(2, 102, 43),
    QVector3D(3, 5, 0),
    QVector3D(6, 4, 0)
  };
  std::tuple<int, int, int, int> result = dr.border(points);
  EXPECT_EQ(std::get<0>(result), 99); // ymax
  EXPECT_EQ(std::get<1>(result), 0); // ymin
  EXPECT_EQ(std::get<2>(result), 6); // xmax
  EXPECT_EQ(std::get<3>(result), 0); // xmin
}
```

Листинг 3.3 – Тест для функции, которая вычисляет коэффициенты плоскости

```
TEST(PlaneCoefTest, PlaneCoefTest) {
  drawer dr;
  std::vector<QVector3D> points = {
    QVector3D(4, 0, 0),
    QVector3D(67, 1, 9),
    QVector3D(2, 8, 3)
  };
  std::tuple<int, int, int, int> result = dr.plane_coef(points);
  EXPECT_EQ(std::get<0>(result), -69);   // a
  EXPECT_EQ(std::get<1>(result), -207); // b
  EXPECT_EQ(std::get<2>(result), 506);   // c
  EXPECT_EQ(std::get<3>(result), 276); // d
}
```

3.5. Функциональное тестирование

Функциональным тестированием в данном случае является визуальное сравнение ожидаемого результата с полученным программой.

Алгоритм проведения функционального тестирования:

1) разработать тестовые случаи для программы;

2) составить входные данные для каждового тестового случая;

3) запустить программу на этих входных данных;

4) визуально оценить результат.

Функциональные тесты:

1) на сцене 1 фламинго с растительностью на озере;

2) на сцене одновременно несколько фламинго (2, 6);

3) на сцене ни одного фламинго;

4) нескольких источников света;

5) установка значения прозрачности фламинго в 0.5;

6) разная плотность растительности (0, 0.4).

Примеры работы программы, которые подтверждают выполнение всех разработанных функциональных тестов, представлены на рисунках (3.3)–(3.8).

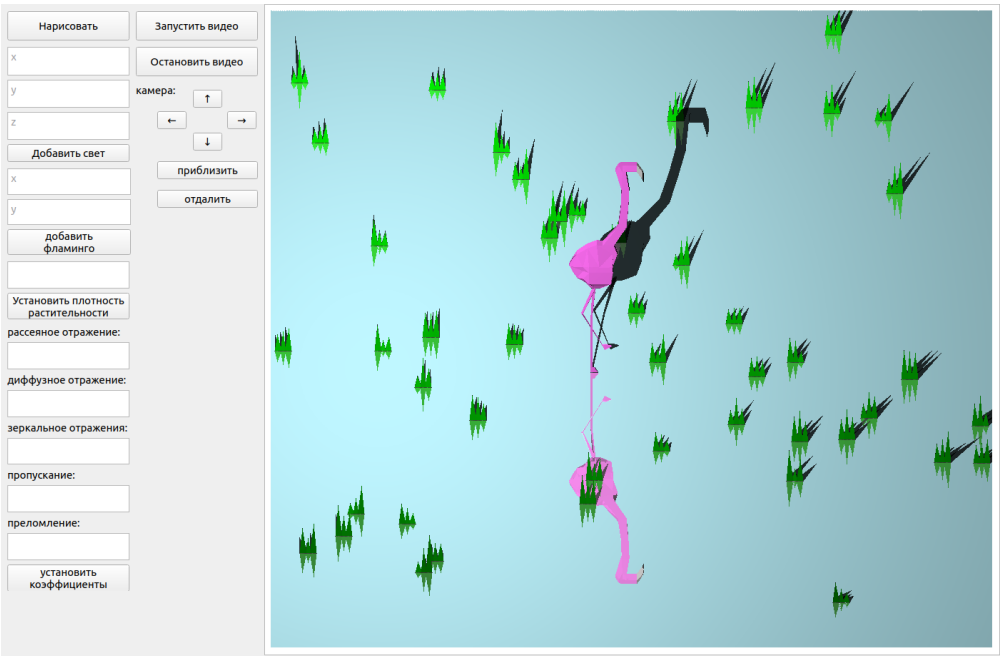

Рисунок 3.3 – Пример работы программы 1

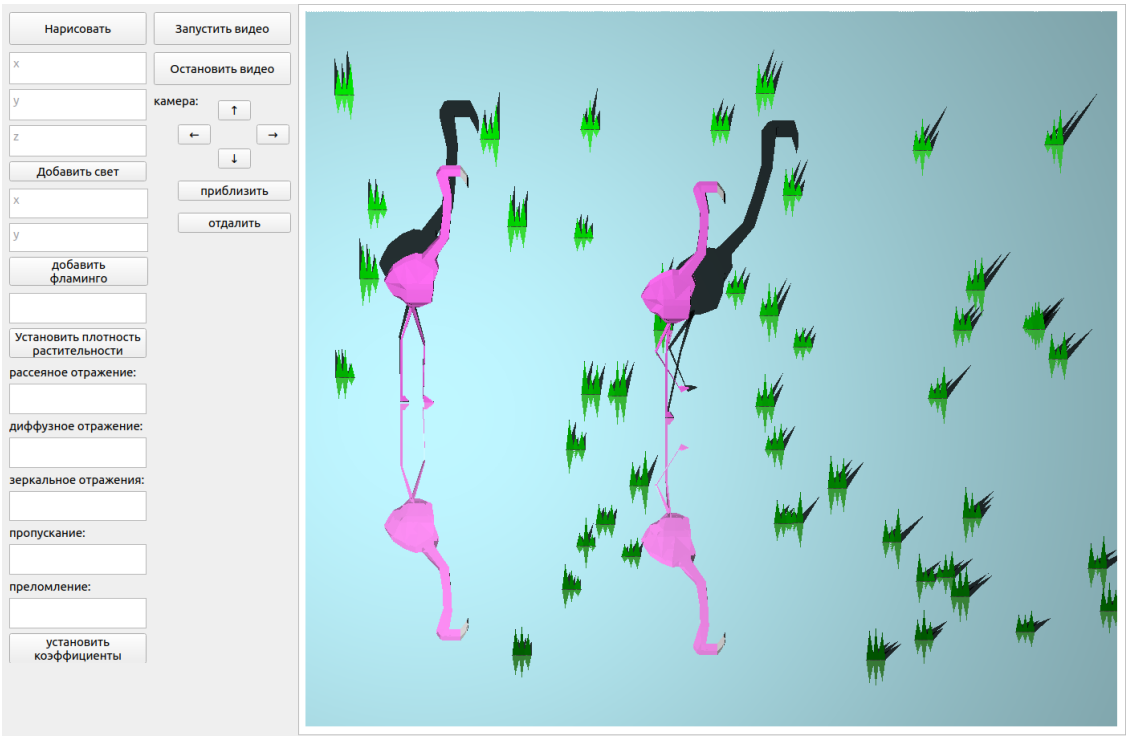

Рисунок 3.4 – Пример работы программы 2

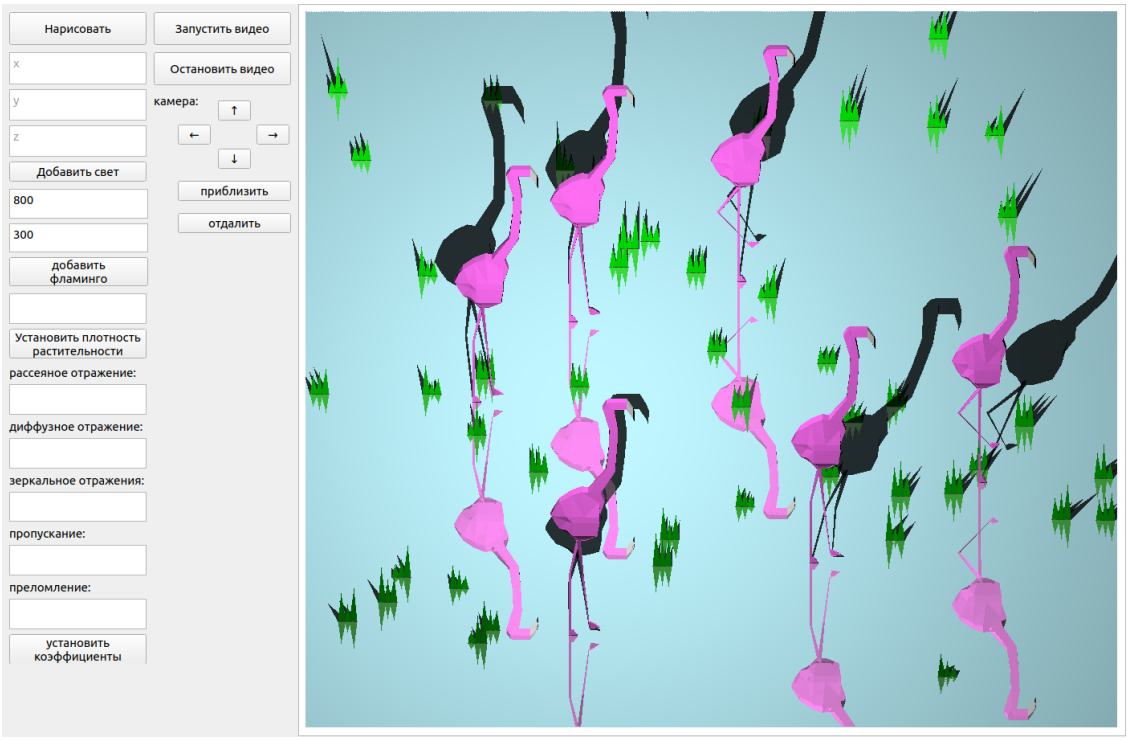

Рисунок 3.5 – Пример работы программы 3

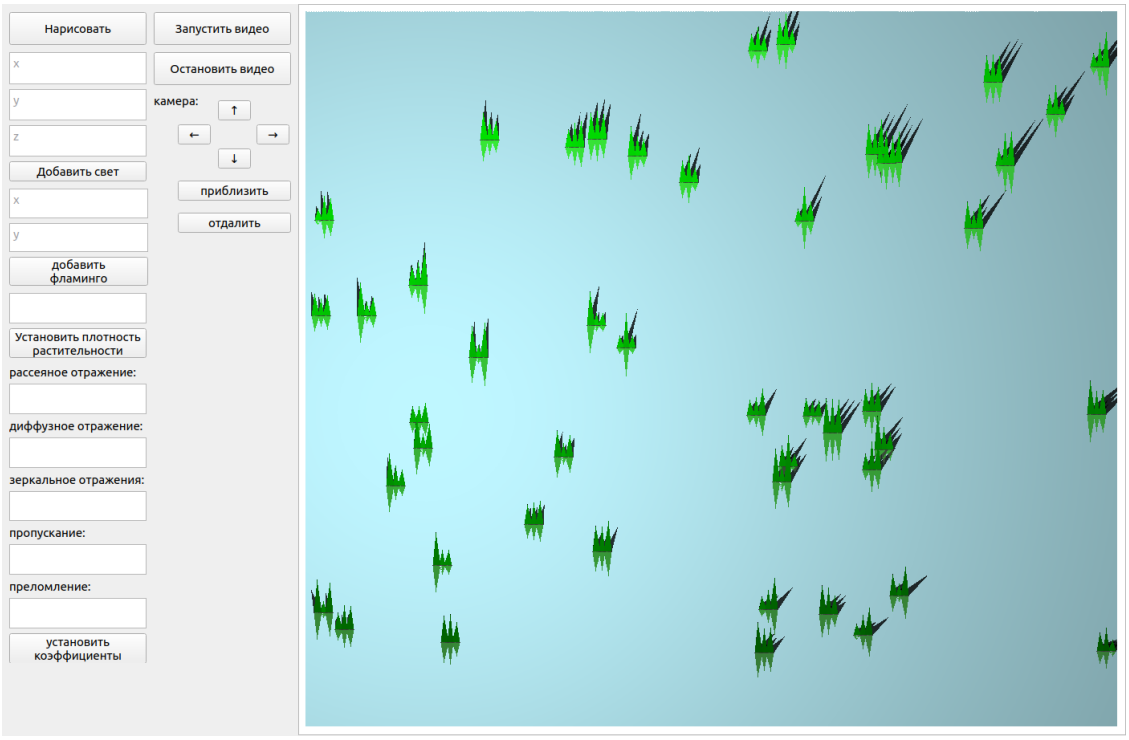

Рисунок 3.6 – Пример работы программы 4

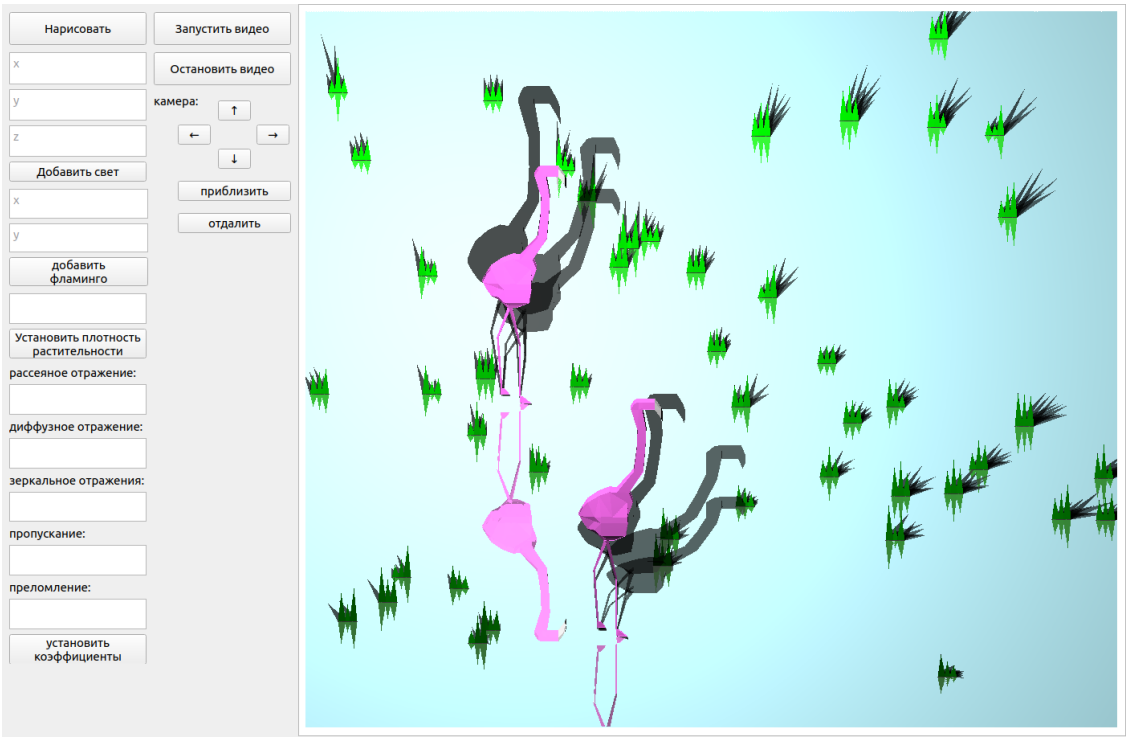

Рисунок 3.7 – Пример работы программы 5

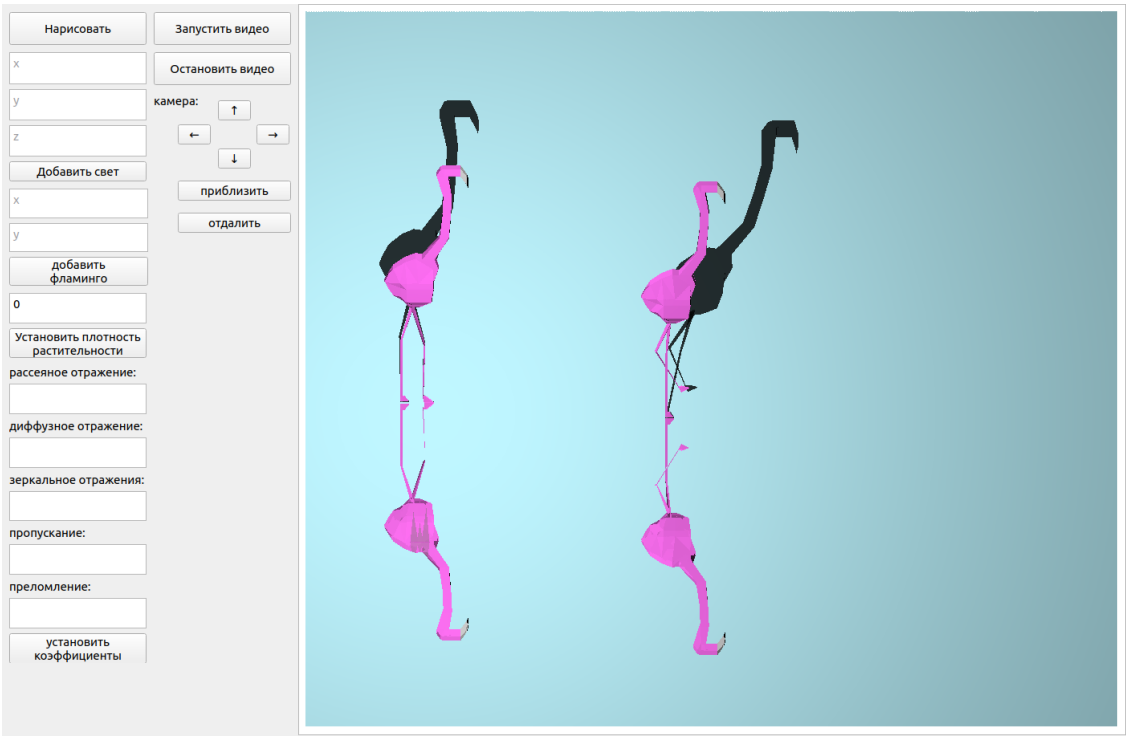

Рисунок 3.8 – Пример работы программы 6

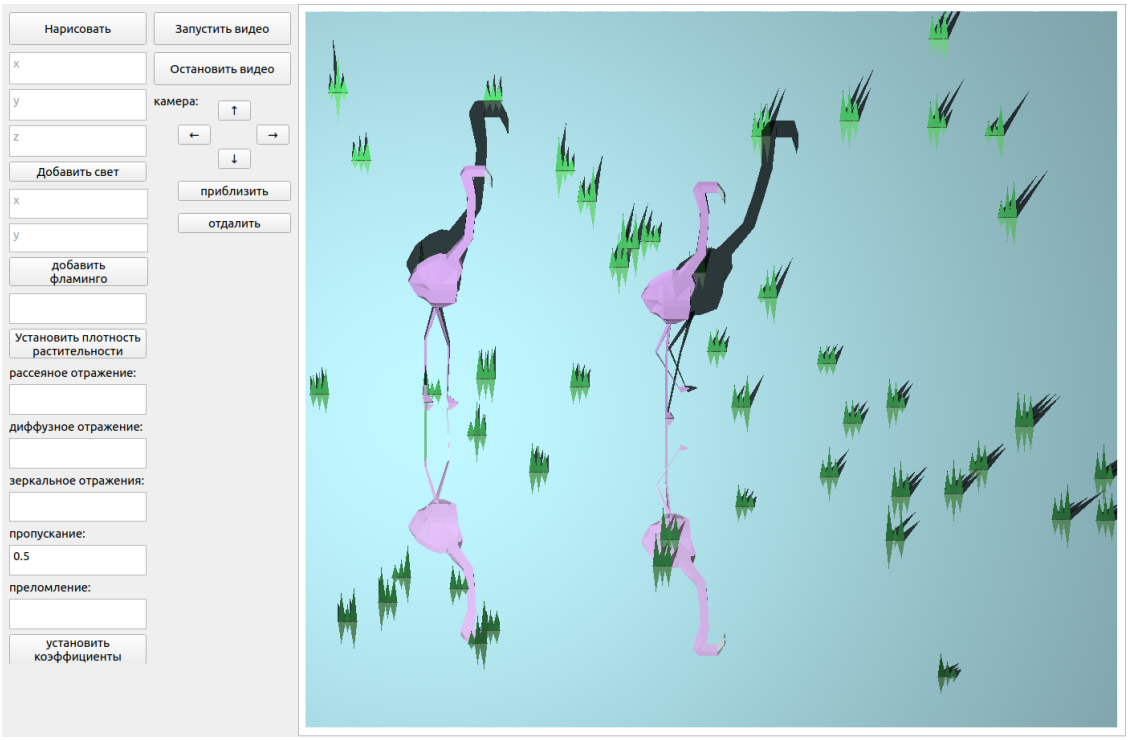

Рисунок 3.9 – Пример работы программы 7

3.6. Выводы к технологической части

В данной части были рассмотрены средства реализации, разработано программное обеспечение, рассмотрен интерфейс и сценарии тестирования.

4. Исследовательская часть

В данной части будет проведен анализ времени отрисовки кадра при различных входных данных.

4.1. Технические характеристики

Технические характеристики устройства, на котором выполнялось исследование представлены далее:

1) операционная система — Ubuntu 22.04.3 [10] Linux x86_64;

2) память — 16 Гб;

3) процессор — Intel® Core™ i5-1135G7 @ 2.40 ГГц.

4.2. Время генерации кадра

Используются функции замера процессорного времени $std :: chrono :: system\_clock :: now(...)$ и $std :: chrono :: duration\_cast < std :: chrono :: milliseconds >$ из библиотеки $chrono$ на *C++*. Функция возвращает процессорное время в миллисекундах.

Замеры проводились для разного количества фламинго на сцене — от 1 до 15 и для разного количества источников света — от 1 до 10.

Результаты замеров времени приведены в таблицах 4.1 и 4.2 (время в мс). А их графическое представление на рисунках 4.1 и 4.2 соответсвенно.

Таблица 4.1 – Результаты замеров времени (1-5 источников света)

Кол-во фламинго	Количество источников света				
	1	2	3	4	5
1	325.04	483.8	612.4	951.4	1169.04
2	347.562	515.552	704.496	979.096	1196.68
3	359.622	536.262	767.9	1041.52	1232.87
4	374.705	564.41	797.116	1081.74	1268.95
5	386.948	581.216	818.725	1118.27	1428.08
6	401.118	646.689	855.869	1148.33	1318.48
7	414.805	682.748	932.195	1182.57	1425.22
8	431.992	703.39	917.808	1246.54	1537.09
9	493.04	722.936	1045.51	1279.62	1544.96
10	496.762	757.037	1084.02	1292.98	1568.52
11	509.95	775.841	1184.6	1353.12	1577.66
12	522.278	804.074	1152.9	1418.92	1746.19
13	539.571	847.923	1185.88	1444.88	1738.25
14	556.383	887.557	1212.68	1494.04	1747.57
15	570.041	910.32	1250.13	1532.452	1790.42

Таблица 4.2 – Результаты замеров времени (6-10 источников света)

Кол-во фламинго	Количество источников света				
	6	7	8	9	10
1	1248.68	1539.35	1734.08	1789.25	496.762
2	1304.91	1603.49	1740	1885.53	757.037
3	1349.08	1625.62	1782.72	2016.54	1084.02
4	1396.44	1651.1	1860.19	2185.9	1152.9
5	1457.22	1702.88	1914.93	2275.28	1185.88
6	1509.89	1765.72	1981.04	2318.69	1318.48
7	1569.4	1830.79	2054.04	2387.47	1425.22
8	1639.82	1889.27	2154.76	2416.02	1537.09
9	1685.39	1952.49	2224.71	2554.56	1544.96
10	1730.46	2046.66	2273.11	2614.62	1568.52
11	1783.34	2084.91	2339.2	2796.38	1577.66
12	1824.53	2126.68	2492.73	2862.62	1746.19
13	1881.78	2209.03	2558.71	3029.14	1738.25
14	1921.89	2265.88	2635.27	3113.45	1747.57
15	1979.87	2312.32	2698.32	3212.42	1790.42

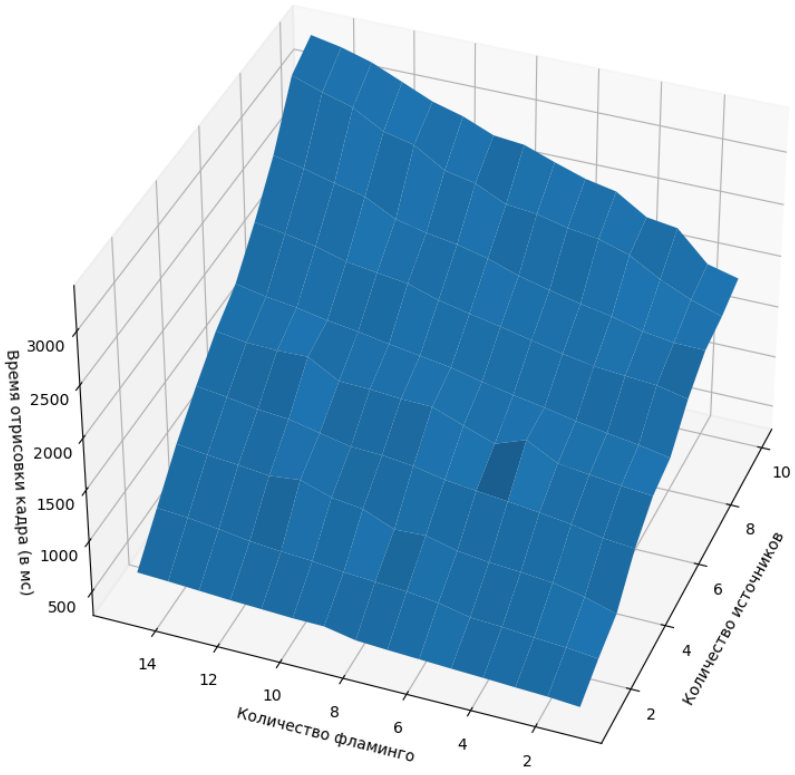

Рисунок 4.1 – Сравнение времени отрисовки кадра при разном количестве фламинго и источников света

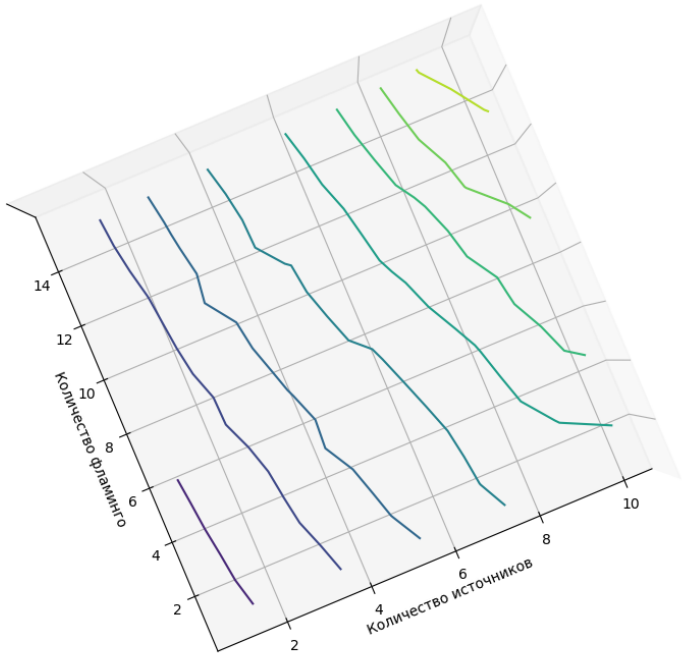

Рисунок 4.2 – Линии уровня графика сравнения времени отрисовки кадра при разном количестве фламинго и источников света

Эта функция зависимости времение от количества фламинго и источников света на сцене можно аппроксимируется функцией (4.1).

$$f(x,y) = 0.833 \cdot y^2 + 0.038 \cdot x^2 + 8.598 \cdot x \cdot y + 169.857 \cdot y + 8.693 \cdot x + 153.645, \quad (4.1)$$

где x — количество фламинго, y — количество источников света, $f(x,y)$ — функция зависимости времени.

На рисунке 4.3 изображены исходные точки синим цветом и аппроксимирущая поверхность, заданная функцией (4.1), зеленым цветом.

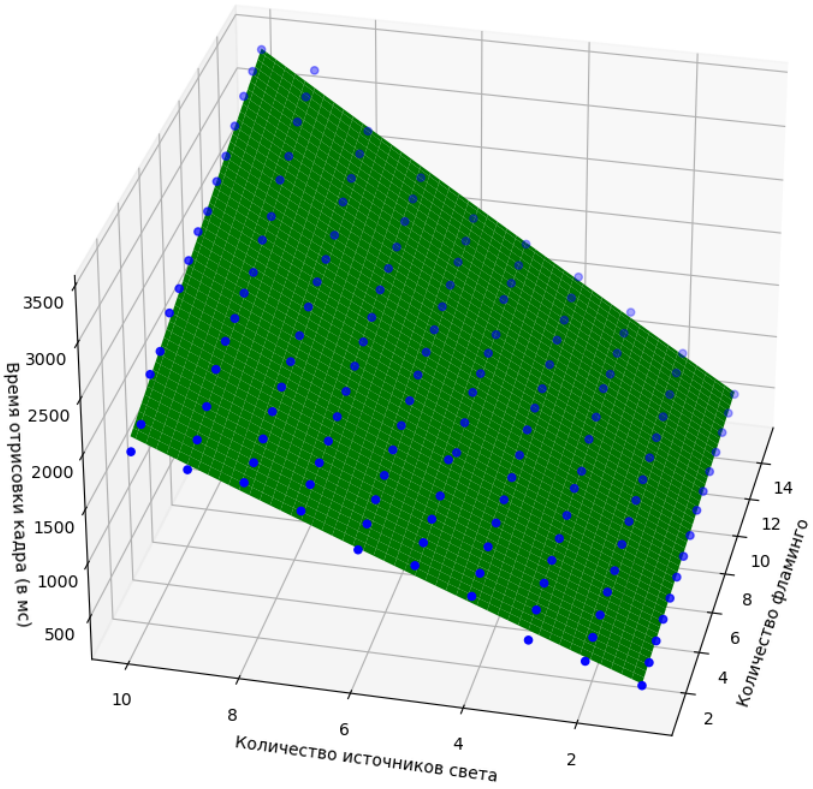

Рисунок 4.3 – Аппроксимирующая поверхность

4.3. Выводы к исследовательской части

В данной части был проведен сравнительный анализ времени выполнения работы программы для отрисовки 1 кадра при различном количестве

фламинго и источников света на сцене. Из проведенного эксперимента можно сделать вывод, что функция зависимости времени от количества фламинго и источников света на сцене является полиномом 2 степени. При минимальном количестве фламинго и источников света время отрисовки кадра равняется примерно 0.3 секунды, а при 10 фламинго и 10 источников света время становится уже около 1.5 секунд.

ЗАКЛЮЧЕНИЕ

В ходе работы были выполнены следующие задачи:

1) формализована задача в виде IDEF0 диаграммы;
2) проведен анализ существующих алгоритмов компьютерной графики: удаления невидимых линий и поверхностей, построения теней, закраски и освещения;
3) спроектировано программное обеспечение для построения трехмерной сцены и визуализации озера с растительностью и фламинго;
4) выбраны средства реализации спроектированного программного обеспечения, а также разработано это программного обеспечение;
5) исследована зависимость скорости генерации кадра разработанного программного обеспечения от числа объектов на сцене и количества источников света.

Все задачи были выполнены, следовательно, поставленная цель достигнута.

Список использованных источников

1. А.В., Куров. Конспект лекций по компьютерной графике / Куров А.В. — 2023.

2. Д., Роджерс. Алгоритмические основы машинной графики. / Роджерс Д. — М.Мир, 1989. — С. 512.

3. В.А., Ярошевич. Компьютерная графика / Ярошевич В.А. — М.: Издательство МИЭТ, 2015.

4. А. Н. Канатников, А. П. Крищенко. Аналитическая геометрия: Учебник для вузов / А. П. Крищенко А. Н. Канатников. — М.: Издательство МГТУ им. Н.Э. Баумана, 2019. — С. 376.

5. Документация языка C++ [Электронный ресурс]. — Режим доступа: `https://docs.microsoft.com/ru-ru/cpp/cpp/?view=msvc-170` (дата обращения: 11.12.2023).

6. Документация библиотеки Qt [Электронный ресурс]. — Режим доступа: `https://doc.qt.io` (дата обращения: 11.12.2023).

7. Документация фреймворка GoogleTest [Электронный ресурс]. — Режим доступа: `https://google.github.io/googletest/` (дата обращения: 11.12.2023).

8. Date and time utilities [Электронный ресурс]. — Режим доступа: `https://en.cppreference.com/w/cpp/chrono` (дата обращения: 11.12.2023).

9. Документация формата obj [Электронный ресурс]. — Режим доступа: `https://docs.fileformat.com/ru/3d/obj/` (дата обращения: 11.12.2023).

10. Ubuntu 20.04.3 LTS (Focal Fossa) [Электронный ресурс]. — Режим доступа: `https://releases.ubuntu.com/20.04/` (дата обращения: 11.12.2023).

ПРИЛОЖЕНИЕ А

Презентация к курсовой работе.