

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1 Аналитическая часть	4
1.1 Алгоритмы удаления невидимых поверхностей или линий	4
1.2 Отражение	7
1.3 Модель освещения	8
1.4 Закраска	10
1.5 Представление объектов сцены	11
1.6 Сравнение алгоритмов удаления невидимых поверхностей и линий	12
1.7 Генерация горного ландшафта	12
1.8 Формализация задачи	15
1.9 Вывод	16
2 Конструкторская часть	17
2.1 Алгоритм построения изображения	17
2.2 Математические основы алгоритмов	19
2.3 Вывод	21
3 Технологическая часть	22
3.1 Средства реализации	22
3.2 Диаграмма классов	22
3.3 Пример работы программы	23
4 Исследовательская часть	31
4.1 Технические характеристики	31
4.2 Зависимость времени создания изображения от количества потоков	31
4.3 Вывод	33
ЗАКЛЮЧЕНИЕ	34
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	35

ВВЕДЕНИЕ

Современные методы компьютерной графики находят применение в самых разных областях, от разработки видеоигр до научной визуализации сложных физических процессов. Одной из ключевых задач является создание фотorealистичных изображений, передающих естественное представление объекта. В рамках данной работы рассматривается визуализация воздушного шара на фоне горного ландшафта.

Визуализация такой сцены требует учета множества аспектов: генерации сложной поверхности горного ландшафта, корректного вычисления освещения, теней, наложения текстур.

Цель работы заключается в разработке программной визуализации сцены с использованием современных методов компьютерной графики, обеспечивающей качественное изображение сцены с учетом световых эффектов и особенностей поверхностей объектов.

Для достижения поставленной цели планируется решить следующие задачи:

1. необходимо разработать программу для визуализации горного ландшафта с высоты полета воздушного шара;
2. Ландшафт генерировать процедурно;
3. В качестве неподвижного источника освещения должно выступать солнце;
4. Требуется провести исследование времени работы программы от количества потоков.

1 Аналитическая часть

1.1 Алгоритмы удаления невидимых поверхностей или линий

Для корректного отображения объектов необходимо обеспечить удаление невидимых поверхностей и линий. Поверхности могут скрываться как самим объектом, так и другими объектами.

Алгоритм Робертса работает в объектном пространстве. В первую очередь он удаляет те грани и линии, которые скрываются самим объектом. После чего оставшиеся грани выпуклого многогранника сравниваются с оставшимися гранями другого выпуклого многогранника. Если же многогранник не является выпуклым, то его придется разбивать на выпуклые, что приводит к дополнительным затратам.

Алгоритм, использующий Z-буффер, работает в пространстве изображений. В нем используется буффер для запоминания глубины каждого видимого пикселя. При обработке нового пикселя его глубина сравнивается с глубиной, записанной в буфере. Основной недостаток – потребление большого количества памяти, требуемой для самого Z-буфера. На рисунке 1.1 представлено полученное в процессе рендеринга с применением алгоритма, использующего Z-буффер, 3D изображение. На рисунке 1.2 представлена визуализация Z-буфера, хранящего Z-координаты 3D объектов сцены.

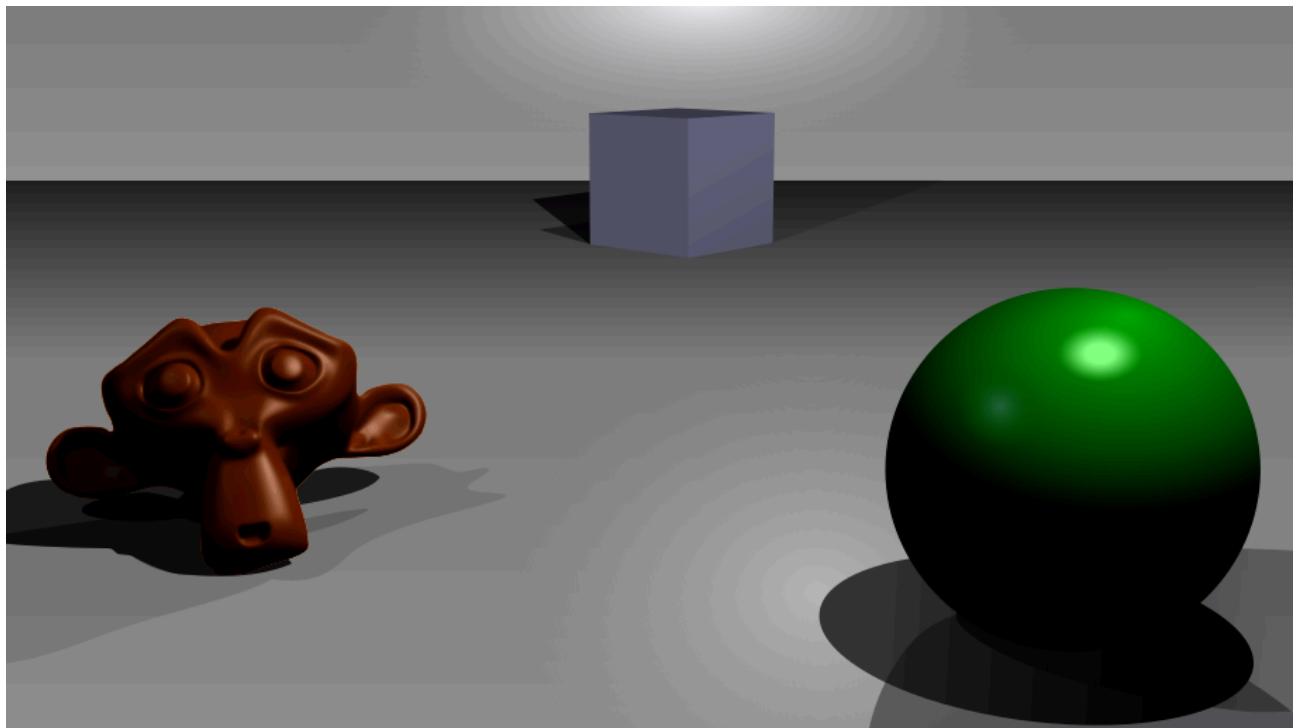


Рисунок 1.1 – Визуализация 3D сцены

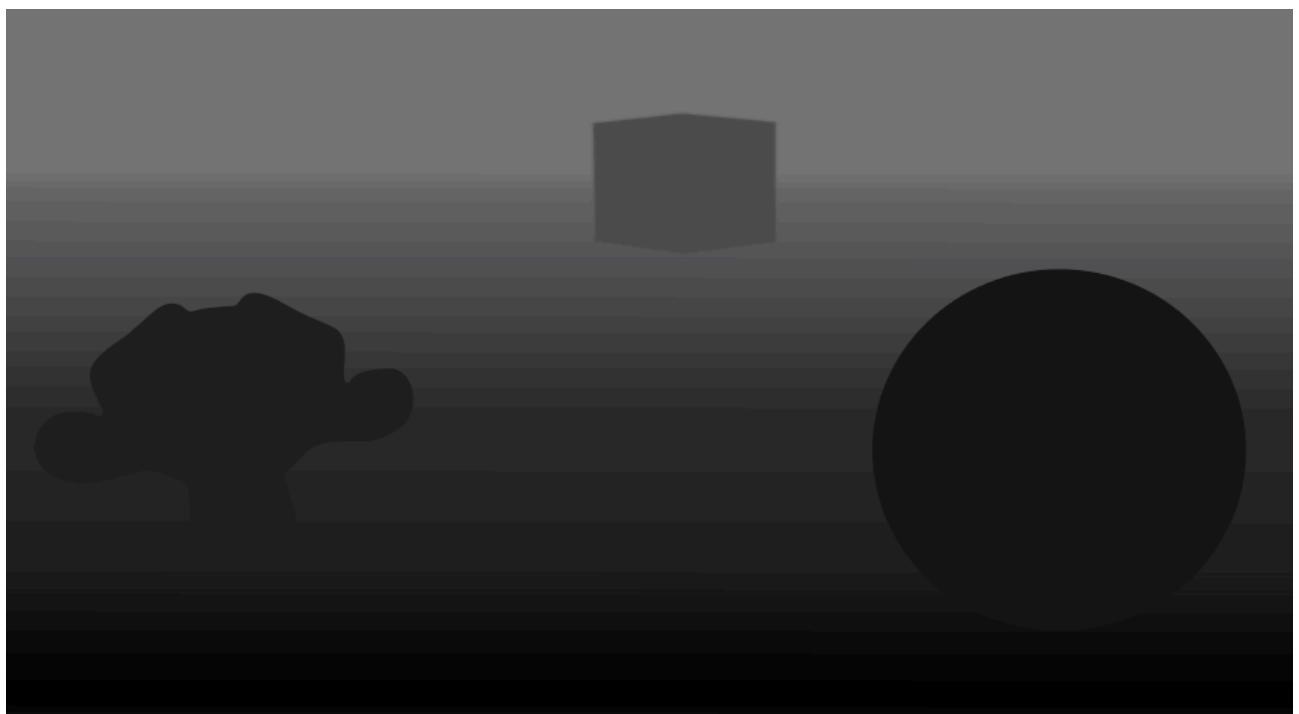


Рисунок 1.2 – Визуализация Z-буффера, представляющего сцену

В алгоритме, использующем трассировку лучей, из положения камеры испускается луч через каждый пиксель изображения. Каждый луч проверяется на пересечение с объектом 3D сцены. В точке пересечения определяется освещенность поверхности по параметрам. Для расчета теней испускаются

тестовые лучи из точки пересечения к источникам света, если тестовый луч блокирует другие объекты сцены, то точка затеняется по выбранному методу затенения. На рисунке 1.3 представлено изображение, полученное с помощью алгоритма, использующего Z-буффер. На рисунке 1.4 представлено изображение, полученное с помощью алгоритма, использующего трассировку лучей. В алгоритме, использующем трассировку лучей не требуется вычислять проекции точек объекта на экран, не требуется проводить отсечение объектов, находящихся вне зоны видимости камеры [1].



Рисунок 1.3 – Изображение, полученное с помощью алгоритма, использующего Z-буффер



Рисунок 1.4 – Изображение, полученное с помощью алгоритма, использующего трассировку лучей

1.2 Отражение

Прямое освещение объекта происходит равномерным количеством света, взаимодействующего с поверхностью. После того как свет падает на объект, он отражается в зависимости от свойств поверхности объекта, а также угла падения света. Свет, падающий перпендикулярно, создает более яркое освещение. Матовые поверхности (дерево, камень, штукатурка) отражают больше диффузного света, чем глянцевые, в результате чего они выглядят более мягкими. При таком отражении положение наблюдателя не имеет значения, так как диффузно отраженный свет рассеивается равномерно по всем направлениям.

Окружающий свет не имеет направления. Его интенсивность определяется свойствами материалов поверхностей объектов, а именно их коэффициентами отражения окружающего света. Так же окружающий свет не отбрасывает тени и не имеет какого-то конкретного источника. Окружающий свет действует на поверхности объектов независимо от их ориентации. Он

добавляет базовый уровень освещенности, чтобы избежать полной темноты на теневых участках. Окружающий свет не требует сложных вычислений, таких как расчет направления света.

Зеркальное отражение создает яркие пятна на объектах, исходя из интенсивности коэффициента зеркального отражения поверхности, а также угла падения. Каждый материал имеет свой коэффициент отражения.

1.3 Модель освещения

Модели освещения используются для визуализации световых эффектов, где все расчеты воспроизводятся на основе законов физики. Главной целью использования модели освещения является вычисления цвета каждого пикселя на основе освещенности объекта, представляемого в этом пикселе.

Локальные модели освещения учитывают световые эффекты, происходящие на объекте, исходя из источников света, находящихся рядом. В таких моделях освещённость пикселей определяется в зависимости от угла падения света, свойств поверхности и её взаимодействия с различными типами отражения (диффузного, зеркального, окружающего). Локальные модели упрощают расчёты, так как не учитывают влияние отражённых лучей от других объектов. На рисунке 1.5 представлено изображение, построенное с использованием локальной модели освещения.

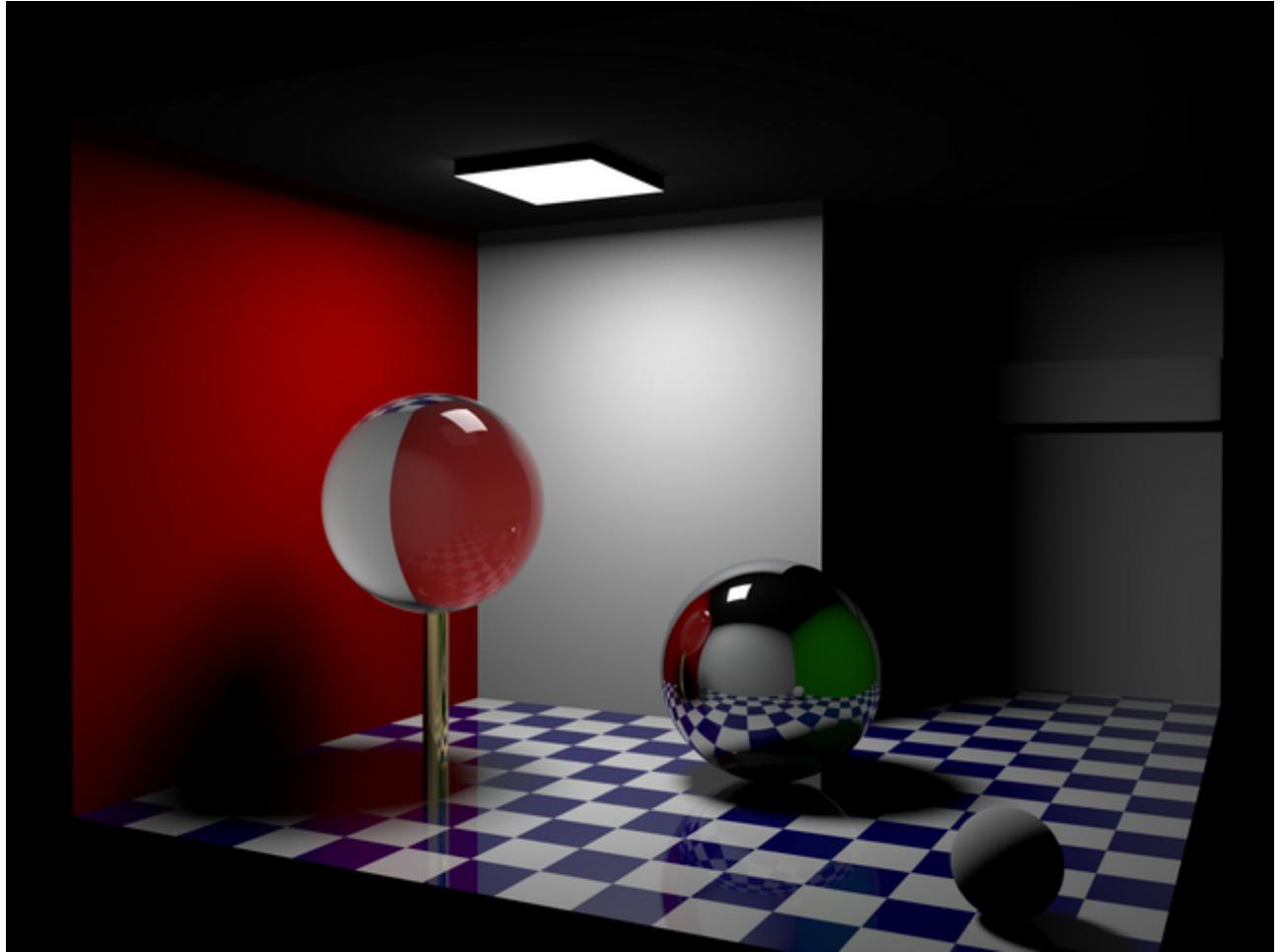


Рисунок 1.5 – Изображение, построенное с использованием локальной модели освещения

Глобальная модель освещения учитывает не только лучи, выпущенные из источника, но и отраженные лучи от других объектов. В результате получаются более реалистичные изображения, создание которых требует больших затрат [2]. На рисунке 1.6 представлено изображение, построенное с использованием глобальной модели освещения.

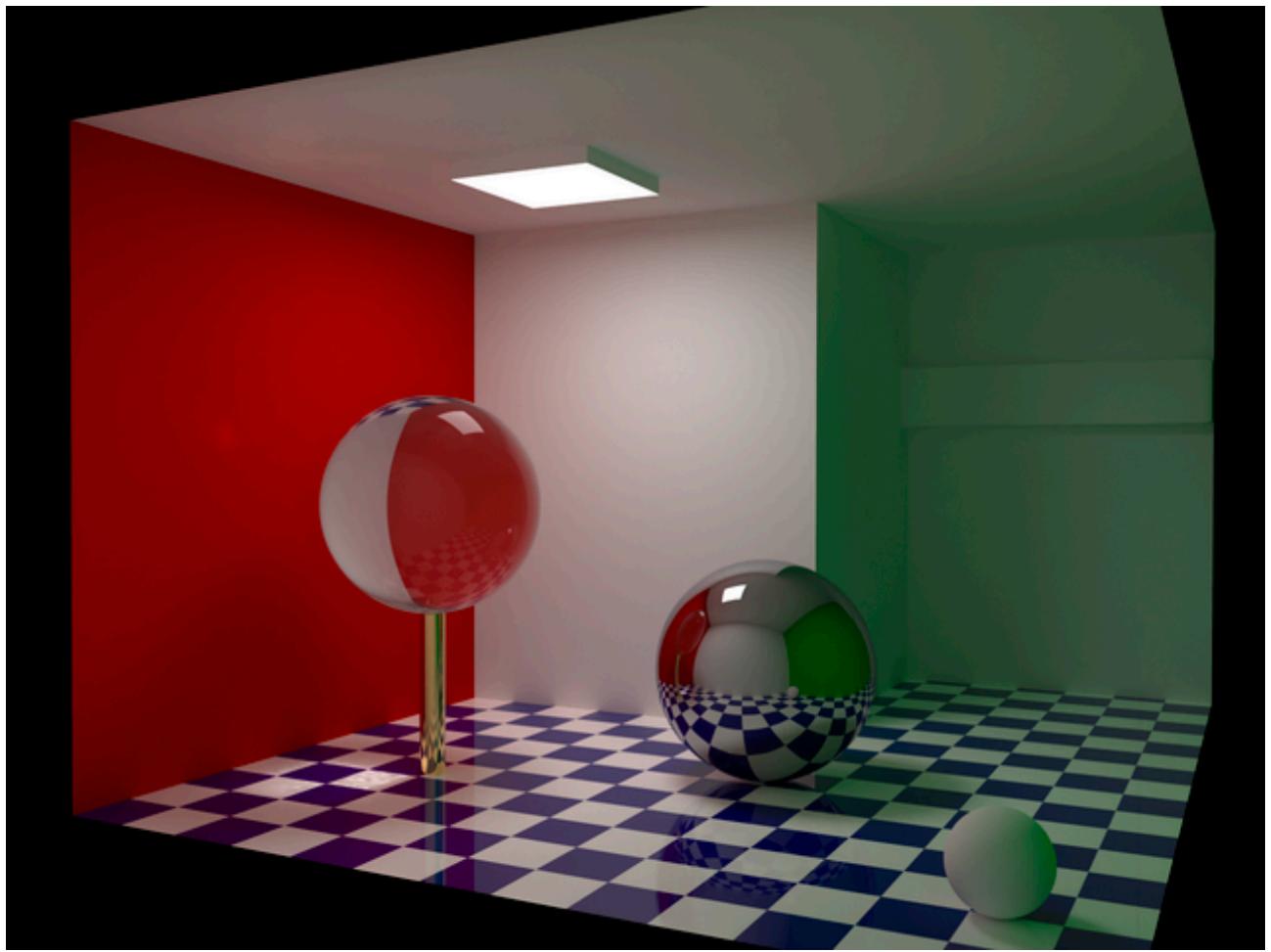


Рисунок 1.6 – Изображение, построенное с использованием глобальной модели освещения

1.4 Закраска

Закраска – это процесс определения цвета каждого пикселя на основе его освещенности и свойств поверхности объекта. Существуют различные подходы к закраске, каждый из которых даёт разные результаты в зависимости от того, какие эффекты освещенности учитываются.

Простая закраска (или плоская закраска) – это самый базовый способ закраски объектов. В этом методе для каждого полигона (или поверхности) используется один фиксированный цвет, который зависит от угла падения света на этот полигон.

Закраска Гуро – это метод, в котором цвета интерполируются между вершинами полигона. Для каждой вершины вычисляется цвет с учётом освещенности (диффузного, зеркального и окружающего света), а затем

этот цвет интерполируется по всей поверхности полигона, создавая плавный переход. Закраска Гуро позволяет получить более гладкие и реалистичные изображения по сравнению с простой закраской.

Закраска Фонга — это более сложный метод, в котором цвет рассчитывается для каждого пикселя, а не только для вершин. В отличие от закраски Гуро, где цвет интерполируется по поверхности, в закраске Фонга используется нормаль к каждой точке на поверхности для вычисления угла между нормалью и направлением света. Этот метод позволяет точно моделировать освещенность, учитывая не только диффузное и зеркальное отражение, но и создание бликов на поверхности.

Закраска Фонга подходит для построения реалистичных изображений объектов с гладкими гранями, так как она учитывает нормали в каждой точке поверхности, что позволяет точно моделировать освещенность и плавные переходы между цветами. Однако этот метод плохо подходит для объектов с перпендикулярными гранями (например, кубов или объектов с резкими углами), поскольку в таких случаях интерполяция нормалей может привести к визуальным артефактам и неестественным переходам цвета, из-за чего объекты с резкими углами выглядят неприродно сглаженными.

1.5 Представление объектов сцены

Объекты сцены в 3D-графике часто представляют собой набор полигональных поверхностей, которые используются для аппроксимации сложных форм. Одним из наиболее распространённых типов полигонов является треугольник, который может быть задан тремя вершинами. Именно три вершины необходимо для того, чтобы задать плоскость.

Для того чтобы правильно вычислять освещенность и отображать световые эффекты на поверхности, каждому полигону требуется задавать вектор нормали, который используется для определения угла падения света и позволяет вычислять такие параметры, как диффузное и зеркальное отражение. Так как плоскость имеет две нормали, то необходимо задавать одну из нормалей вручную.

1.6 Сравнение алгоритмов удаления невидимых поверхностей и линий

В таблице 1.1. представлены результаты сравнения алгоритмов удаления невидимых поверхностей и линий по различным параметрам, где N - количество объектов на сцене, $SIZE$ - количество пикселей на экране, Р - алгоритм Робертса, Z - алгоритм Z-буффер, Т - алгоритм трассировки лучей.

Таблица 1.1 – Сравнение алгоритмов

Параметр	Р	Z	Т
Временная сложность	$O(N^2)$	$O(SIZE \cdot N)$	$O(SIZE \cdot N)$
Подготовка данных	Разбиение на выпуклые объекты	–	–
Дополнительная Память	–	+	–
Преимущества	Простота реализации	Эффективен для 3D сцен	Высокая реалистичность
Параллельные вычисления	–	–	+

1.7 Генерация горного ландшафта

Шум Перлина был предложен Кеном Перлином в 1983 году для генерации процедурных текстур и ландшафтов в компьютерной графике. Ландшафт выглядит естественным благодаря наличию в нем случайных элементов: выступов, неровностей. Шум Перлина добавляет эти случайные элементы в создаваемое изображение.

На генерируемой карте выбирается сетка, в узлах которой нужно сгенерировать векторы градиентов функции изменения высоты. Все векторы имеют единичную длину и случайное направление, как показано в формуле 1.1 [3], где $\theta \in [0, 2\pi]$ — случайный угол.

$$g(x_i) = \cos \theta, \quad g(y_i) = \sin \theta \quad (1.1)$$

В пределах каждого промежутка сетки выбираются точки (x_p, y_p) , для

которых рассчитывается значение шума.

Для каждой точки внутри ячейки сетки вычисляется смещение относительно ближайших узлов, как показано в формуле 1.2 [3], где x_{int} и y_{int} — целочисленные координаты ближайшего узла (например, левого верхнего), а x_p и y_p — координаты точки внутри ячейки, для которой вычисляется шум.

$$u = x_p - x_{\text{int}}, \quad v = y_p - y_{\text{int}} \quad (1.2)$$

Для каждой из четырех ближайших точек сетки рассчитывается скалярное произведение между градиентом узла и вектором смещения, как показано в формуле 1.3 [3], где g_x и g_y — компоненты градиента узла, а d_x и d_y — компоненты вектора смещения до точки.

$$\text{dot}(g, d) = g_x \cdot d_x + g_y \cdot d_y \quad (1.3)$$

где g_x и g_y — компоненты градиента узла, а d_x и d_y — компоненты вектора смещения до точки.

Для получения итогового значения шума в точке используется линейная интерполяция, как показано в формуле 1.4 [3], где a и b — значения шума из двух ближайших узлов, а t — дробная часть координаты, сглаженная функцией *fade*.

$$\text{lerp}(a, b, t) = a + t \cdot (b - a) \quad (1.4)$$

Для плавного перехода между точками применяется функция сглаживания (*fade function*), как показано в формуле 1.5 [3].

$$\text{fade}(t) = 6t^5 - 15t^4 + 10t^3 \quad (1.5)$$

Эта функция позволяет избежать резких изменений значений шума между узлами сетки.

На рисунке 1.7 показана визуализация шума Перлина. На рисунке 1.8 показана 3D визуализация шума Перлина.

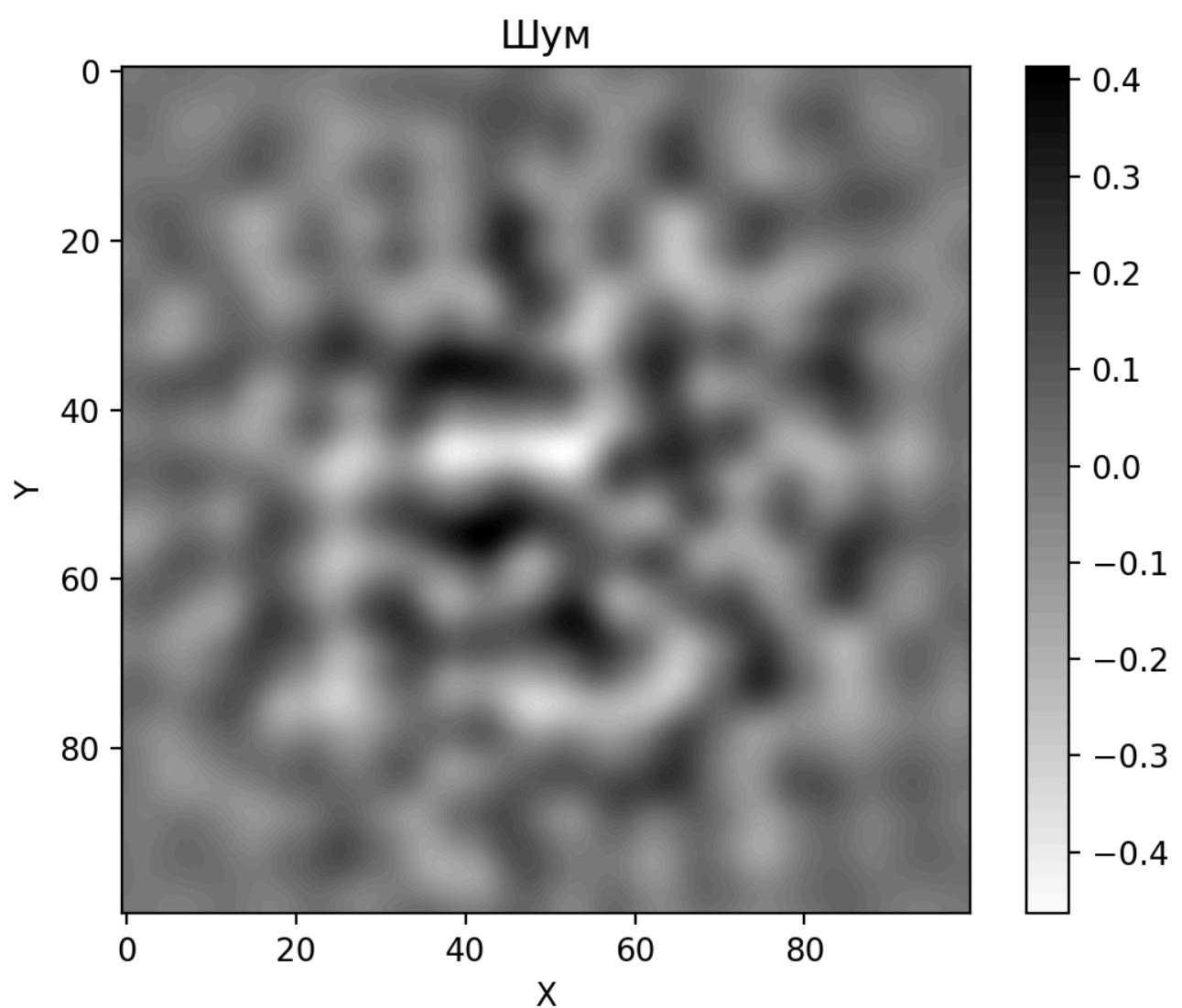


Рисунок 1.7 – Шум перлина

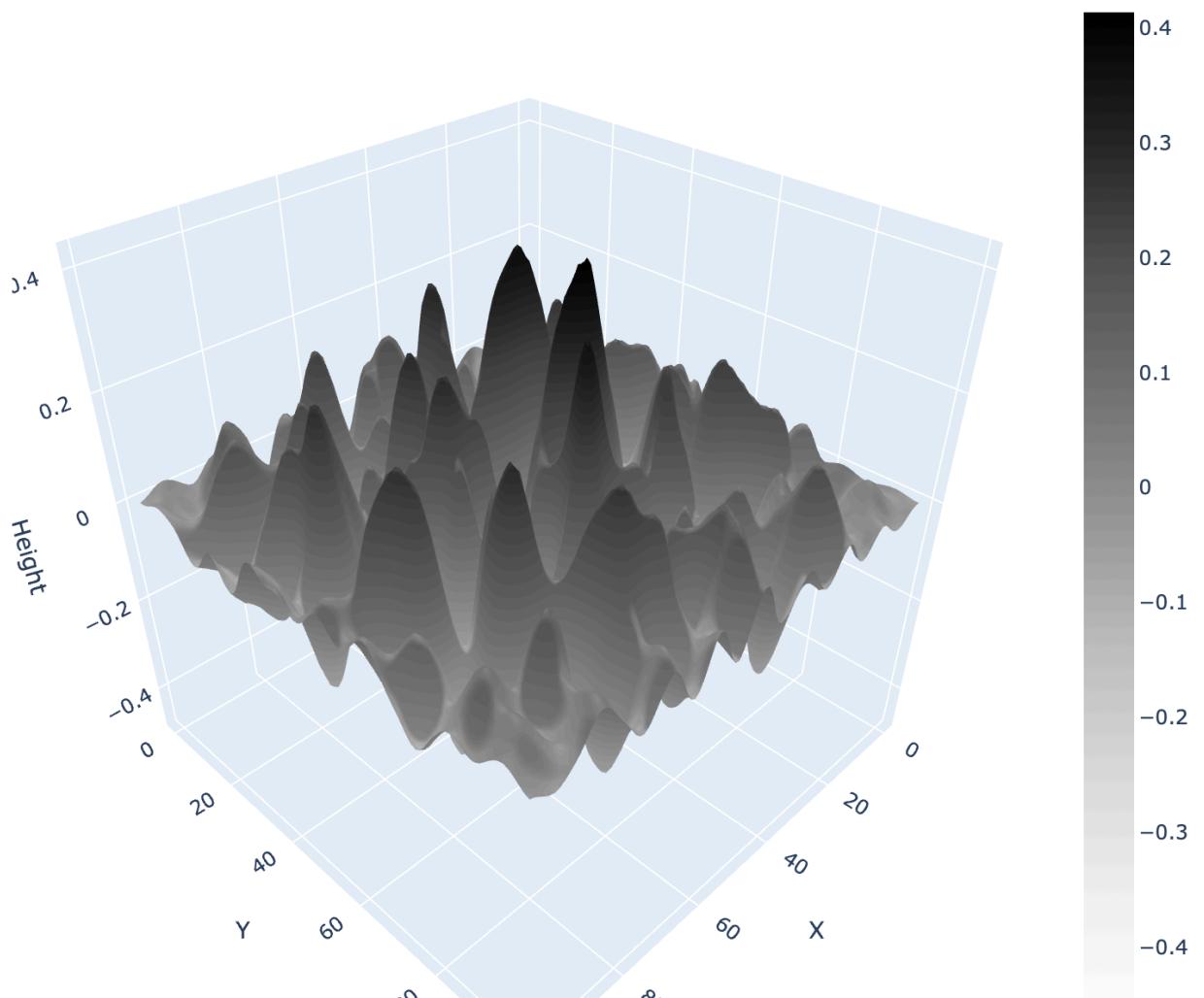


Рисунок 1.8 – 3D визуализация шума

1.8 Формализация задачи

На рисунке 1.9 представлена схема задачи построения реалистичного изображения воздушного шара на фоне горного ландшафта в формате idef0.

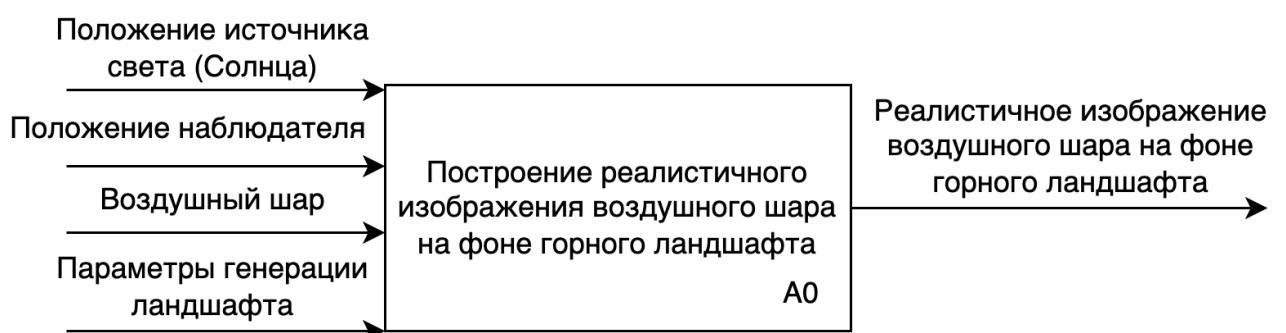


Рисунок 1.9 – Схема задачи построения реалистичного изображения воздушного шара на фоне горного ландшафта в формате idef0

1.9 Вывод

Можно сделать вывод, что для решения задачи построения реалистичного изображения воздушного шара на фоне горного ландшафта, использование алгоритма трассировки лучей будет обеспечивать большую реалистичность изображения. Также алгоритм трассировки лучей поддается распараллеливанию, в отличие от алгоритма, использующего Z-буффер. Поэтому было решено использовать алгоритм трассировки лучей.

Ввиду того, что глобальная модель освещения требует больших вычислительных затрат, было решено использовать локальную модель освещения. Для закраски был выбран алгоритм Фонга, так как почти все объекты сцены (воздушный шар, ландшафт) не имеют острых углов и являются гладкими объектами. Для генерации ландшафта был выбран алгоритм, использующий шум Перлина. Для представления объектов было решено использовать полигональную сетку, состоящую из треугольных полигонов с заданной нормалью.

2 Конструкторская часть

2.1 Алгоритм построения изображения

На рисунке 2.1 изображено представление рендера изображения в виде диаграммы iDef0.

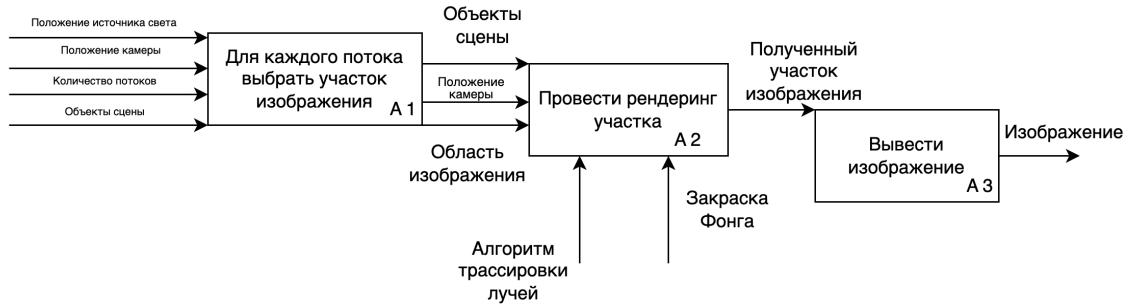


Рисунок 2.1 – Схема рендера изображения

На рисунке 2.2 изображено представление алгоритма, использующего трассировку лучей, в виде блок-схемы. На рисунке 2.3 изображено представление алгоритма, осуществляющего закраску Фонга, в виде блок-схемы.

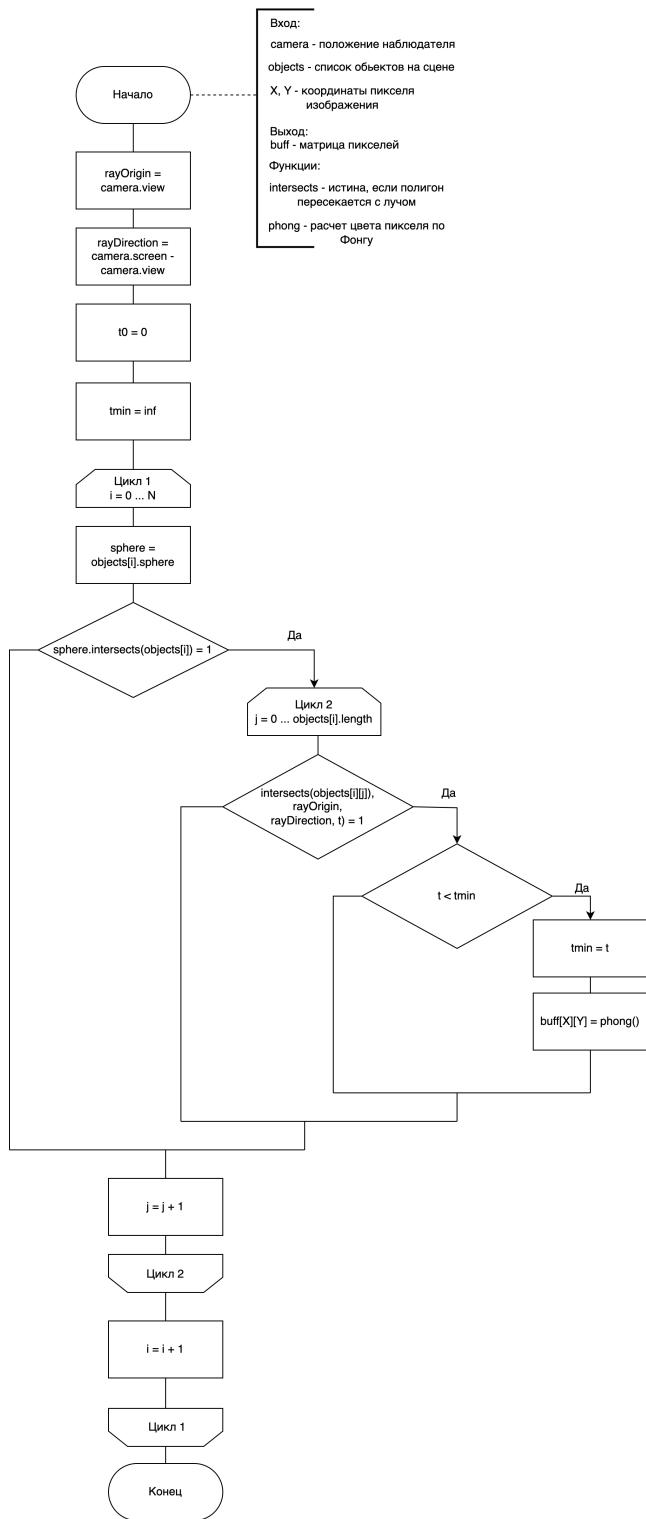


Рисунок 2.2 – Схема

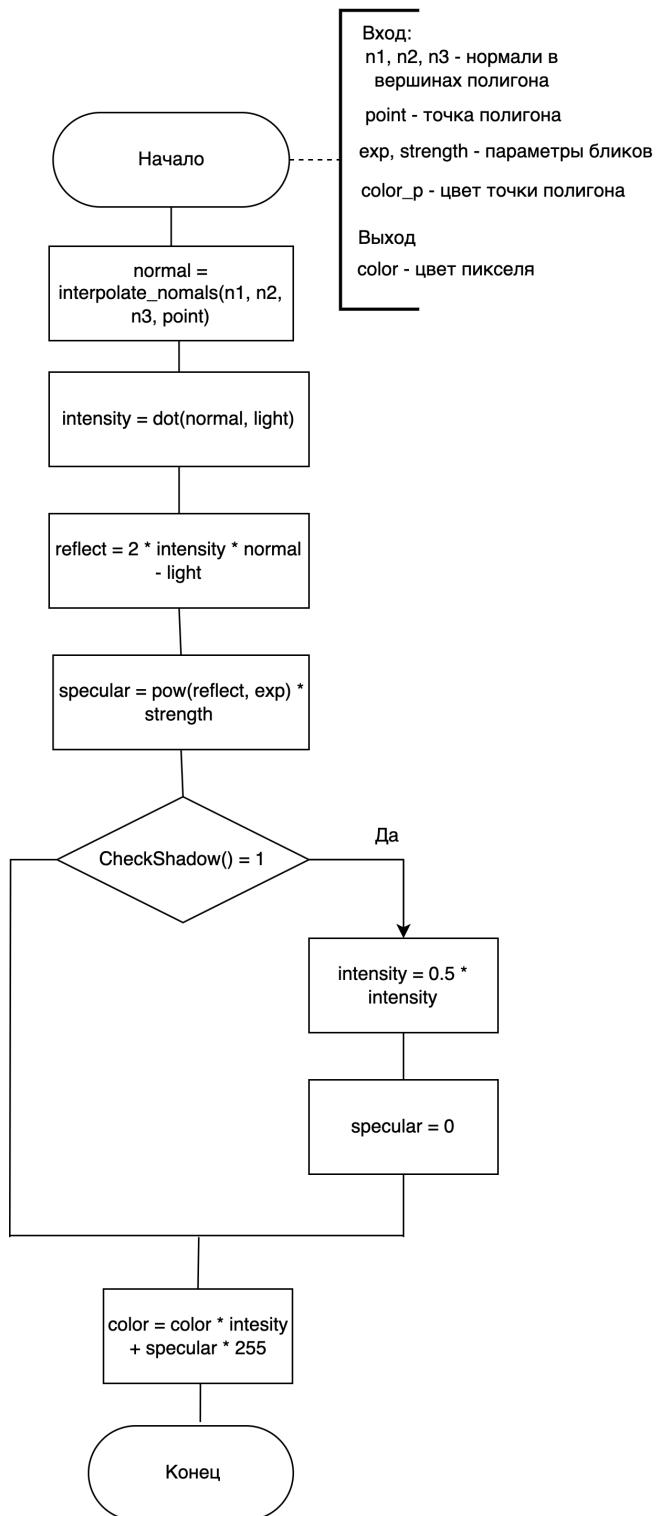


Рисунок 2.3 – Схема

2.2 Математические основы алгоритмов

Алгоритм трассировки лучей основан на поиске пересечения луча с полигоном. Для этого используется параметрическое представление луча и

треугольника. Параметрическое уравнение луча представлено в формуле 2.1 [4], где O - начальная точка луча, D - направление луча.

$$P(t) = O + t \cdot D \quad (2.1)$$

Параметрическое уравнение треугольника представлено в формуле 2.2 [4], где V_0, V_1, V_2 - вершины треугольника.

$$P(u, v) = (1 - u - v) \cdot V_0 + u \cdot V_1 + v \cdot V_2 \quad (2.2)$$

Для нахождения пересечения луча с треугольником необходимо решить систему уравнений, которая выражает пересечение двух параметрических уравнений: луча и треугольника. Для этого используется метод матричного детерминанта для вычисления коэффициентов t, γ и β , которые определяют положение точки пересечения на луче и в пределах треугольника.

Для вычисления пересечения вычисляется детерминант матрицы, которая зависит от разности координат вершин треугольника и направлений луча. Если детерминант слишком мал, то пересечения нет, так как луч и плоскость треугольника не пересекаются.

Детерминант A вычисляется по формуле 2.3 [4], где:

$$1. \ a = v_0[X] - v_1[X],$$

$$2. \ b = v_0[Y] - v_1[Y],$$

$$3. \ c = v_0[Z] - v_1[Z],$$

$$4. \ d = v_0[X] - v_2[X],$$

$$5. \ e = v_0[Y] - v_2[Y],$$

$$6. \ f = v_0[Z] - v_2[Z],$$

$$7. \ g = \text{rayDir}[X],$$

$$8. \ h = \text{rayDir}[Y],$$

$$9. \ i = \text{rayDir}[Z].$$

$$\det(A) = \begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = a(ei - fh) - b(di - fg) + c(dh - eg) \quad (2.3)$$

Если детерминант матрицы $\det(A)$ не равен нулю, то вычисляется параметр t , который определяет расстояние от начала луча до точки пересечения по формуле 2.4 [4], где:

1. $j = v_0[X] - \text{rayOrigin}[X]$,
2. $k = v_0[Y] - \text{rayOrigin}[Y]$,
3. $l = v_0[Z] - \text{rayOrigin}[Z]$.

$$t = -\frac{(f \cdot (a \cdot k - j \cdot b) + e \cdot (j \cdot c - a \cdot l) + d \cdot (b \cdot l - k \cdot c))}{\det(A)} \quad (2.4)$$

2.3 Вывод

В данном разделе были представлены схемы алгоритмов рендеринга изображения и математические основы алгоритма поиска пересечения луча с полигоном.

3 Технологическая часть

3.1 Средства реализации

В качестве языка программирования для реализации программного обеспечения был выбран язык программирования C++ [5], так как он позволяет реализовать все выбранные алгоритмы и имеет поддержку нативных потоков.

Для визуализации изображения и реализации интерфейса был выбран фреймворк QT5 [6] так как в нем есть поддержка вывода изображений.

3.2 Диаграмма классов

На рисунке 3.1 представлена диаграмма классов для разрабатываемого ПО в формате UML.

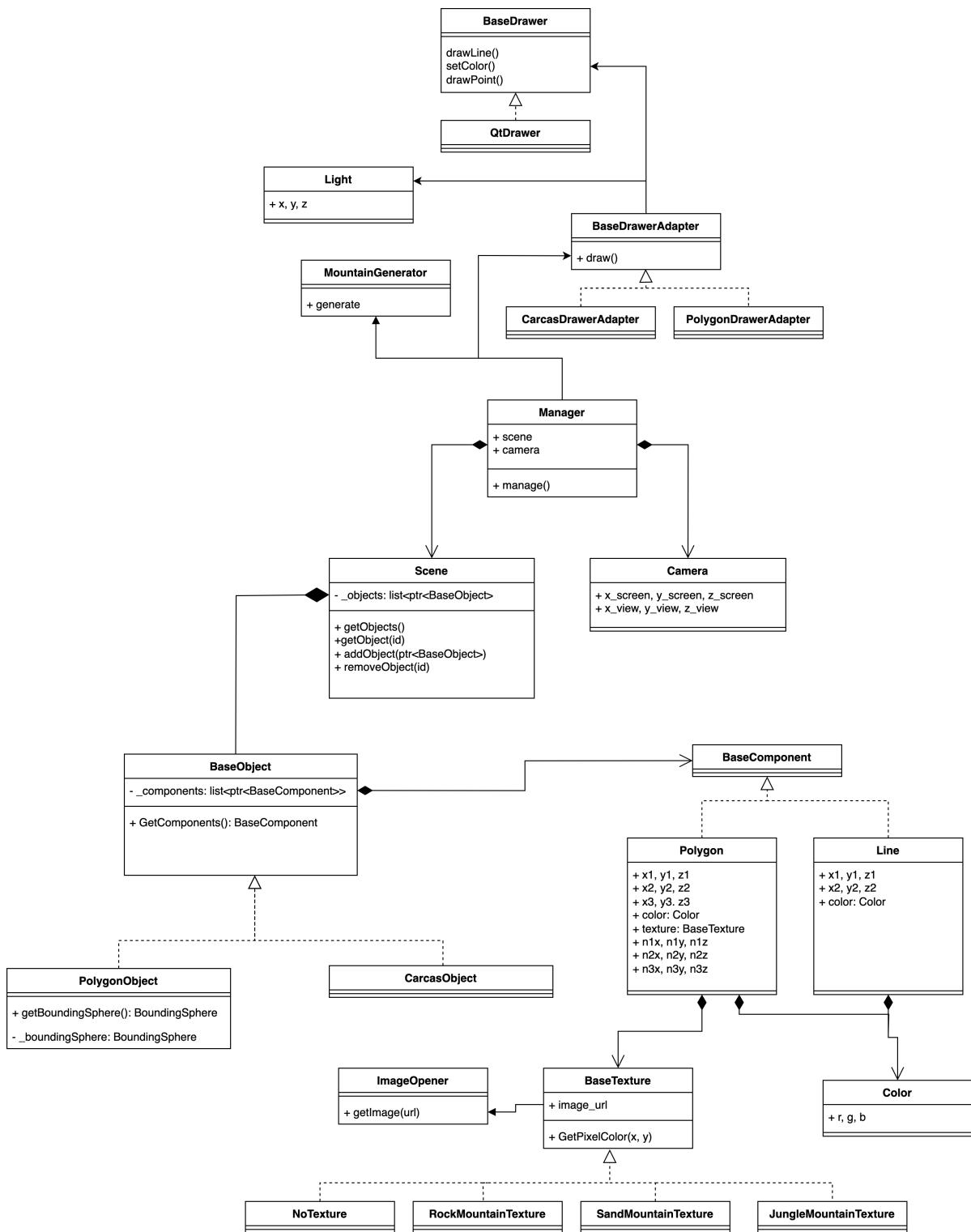


Рисунок 3.1 – Диаграмма классов в формате UML

3.3 Пример работы программы

На рисунках 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8 представлены примеры работы программы.

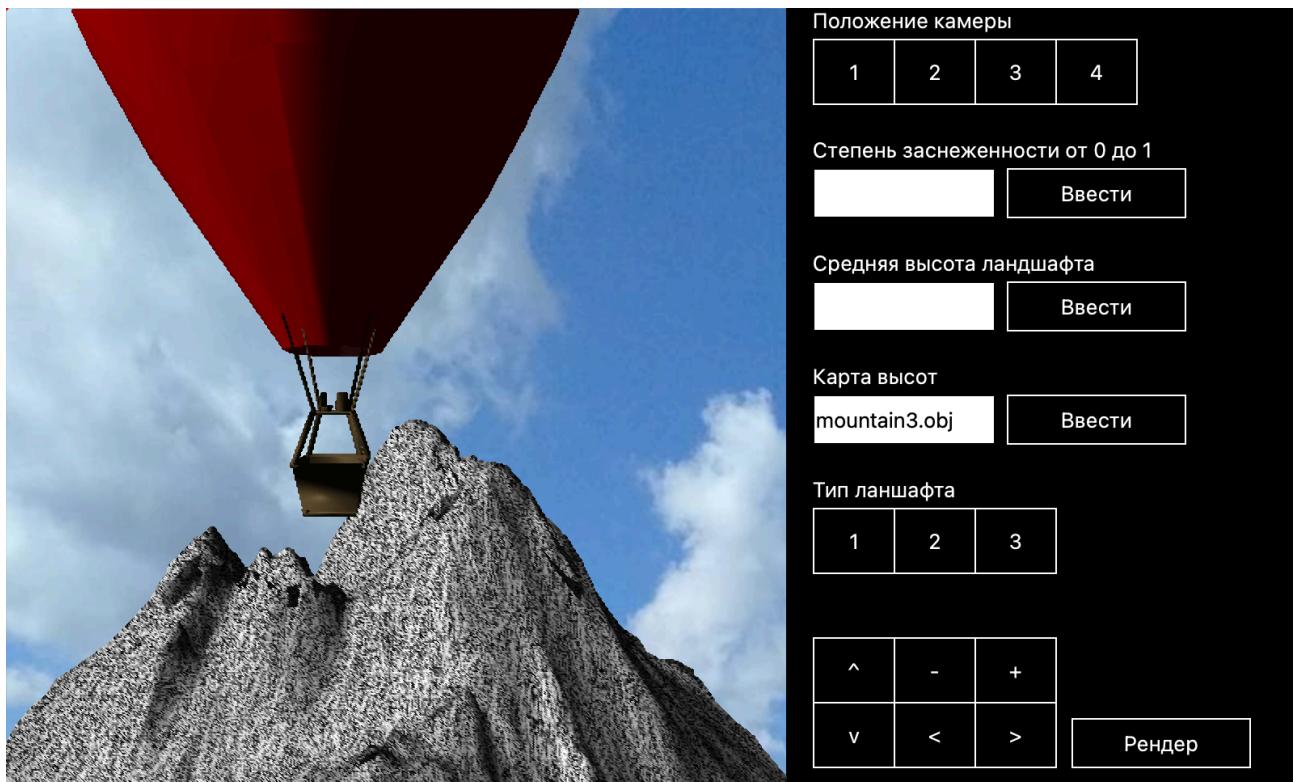


Рисунок 3.2 – Интерфейс программы



Рисунок 3.3 – Пример созданного изображения снежного ландшафта с видом из корзины



Рисунок 3.4 – Пример созданного изображения тропического ландшафта



Рисунок 3.5 – Пример созданного изображения тропического ландшафта с видом из корзины



Рисунок 3.6 – Пример созданного изображения песчаного ландшафта



Рисунок 3.7 – Пример созданного изображения песчаного ландшафта с видом из корзины

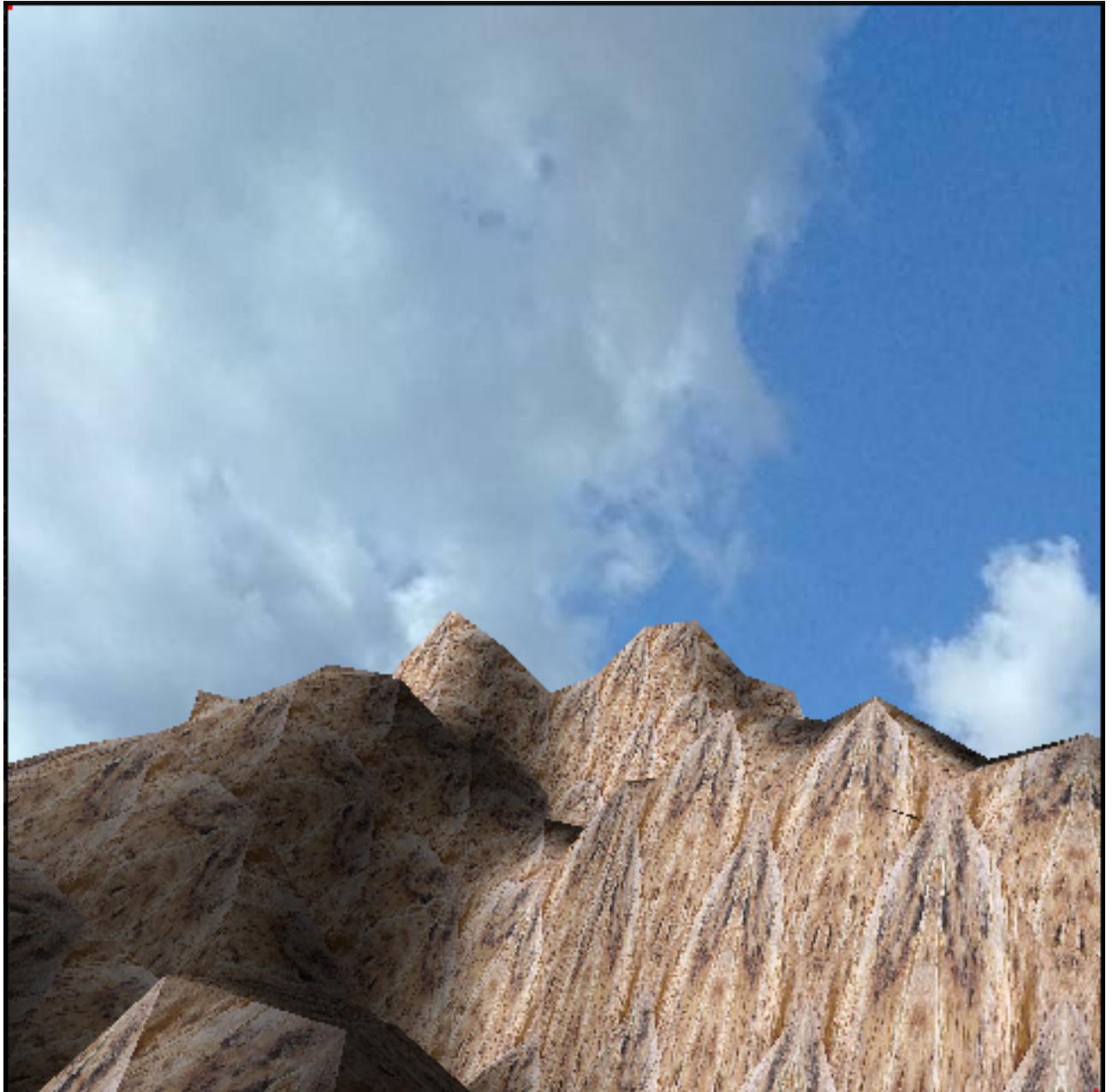


Рисунок 3.8 – Пример созданного изображения песчаного ландшафта без воздушного шара

4 Исследовательская часть

В данной части описано проведённое исследование и его результаты, а также технические характеристики устройства, на котором проводились замеры.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры:

1. операционная система Ubuntu 22.04.045 LTS [7];
2. процессор Intel Core i5-10400F 2.9 ГГц [8];
3. 16 ГБ оперативной памяти.

4.2 Зависимость времени создания изображения от количества потоков

Для проведения замеров времени использовалась функция `std::chrono` [9]. В таблице 4.1 представлена зависимость времени выполнения программы от количества используемых потоков. Для каждого значения количества потоков указано время выполнения программы в секундах.

На рисунке 4.1 представлен график зависимости времени выполнения программы от количества потоков.

Таблица 4.1 – Зависимость времени выполнения от количества потоков

Количество потоков	Время выполнения (секунды)
1	17.8488
2	13.6123
3	10.2385
4	8.41588
5	7.18214
6	6.24993
7	5.488
8	5.1341
9	4.8422
10	4.5506
15	4.05306
20	3.95047
25	3.8543
30	3.78371
31	3.7
32	3.5786
33	3.96616
34	4.24772
35	4.89
40	4.959
45	5.24695

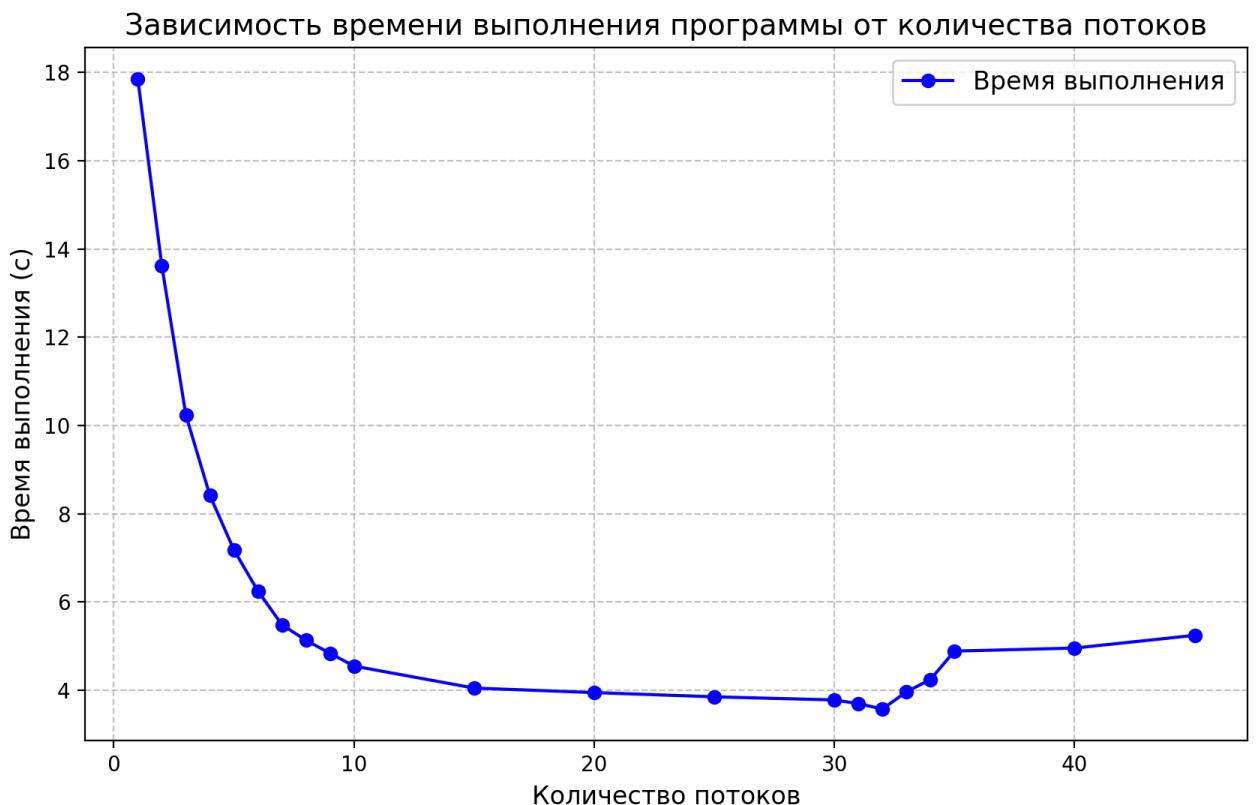


Рисунок 4.1 – График зависимости времени выполнения программы от количества потоков

4.3 Вывод

Из этого раздела можно сделать вывод, что с увеличением количества потоков время выполнения программы сначала значительно сокращается, что связано с эффективным использованием многопоточности. Однако, начиная с определенного количества потоков (около 32), время выполнения перестает уменьшаться, а при дальнейшем увеличении даже возрастает. Это может быть связано с накладными расходами на синхронизацию потоков, ограничениями аппаратного обеспечения или снижением эффективности при распределении нагрузки между потоками.

ЗАКЛЮЧЕНИЕ

В рамках данной работы была решена задача визуализации сцены, включающей воздушный шар и горный ландшафт, с использованием современных методов компьютерной графики.

Для достижения поставленной цели была разработана программа для визуализации горного ландшафта с высоты полета воздушного шара. Также было проведено исследование зависимости времени работы программы от количества потоков, что позволило выявить возможности оптимизации вычислений.

Полученные результаты демонстрируют, что предложенные подходы и алгоритмы успешно справляются с задачей визуализации горного ландшафта и воздушного шара. Выполненный анализ времени работы программы подтверждает эффективность использования параллельных вычислений для повышения производительности.

СПИСОК ИСТОЧНИКОВ

ИСПОЛЬЗОВАННЫХ

1. Гамбетта Габриел. Рейтрайсинг и растеризация. 1-е изд. Санкт-Петербург: Питер, 2022.
2. Pharr Matt, Jakob Wenzel, Humphreys Greg. Physically Based Rendering: From Theory to Implementation. 3-е изд. San Francisco: Morgan Kaufmann, 2016.
3. Akenine-Möller Tomas, Haines Eric, Hoffman Naty. Real-Time Rendering. 4-е изд. Boca Raton: A K Peters/CRC Press, 2018.
4. Shirley Peter. Fundamentals of Computer Graphics. 5-е изд. CRC Press, 2021.
5. Briggs Will. C++20 for Lazy Programmers: Quick, Easy, and Fun C++ for Beginners. Apress, 2020.
6. Eng Lee Zhi. Hands-On GUI Programming with C++ and Qt5. 1-е изд. Birmingham, UK: Packt Publishing, 2018.
7. Ubuntu Documentation. 2024. Дата обращения: 13 декабря 2024. URL: <https://docs.ubuntu.com/>.
8. Intel Core i5-10400F Processor. 2024. Дата обращения: 13 декабря 2024. URL: <https://www.intel.com/content/www/us/en/products/sku/199278/intel-core-i510400f-processor-12m-cache-up-to-4-30-ghz/specifications.html>.
9. std::chrono - C++ Reference. 2024. Дата обращения: 13 декабря 2024. URL: <https://en.cppreference.com/w/cpp/chrono>.