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:*

## *«Моделирование облаков»*

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

Руководитель курсовой работы _____ (Подпись, дата) Кузнецова О. В. (И. О. Фамилия)

*2023 г.*

# СОДЕРЖАНИЕ

# ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В настоящей расчетно-пояснительной записке применяют следующие сокращения и обозначения.

ПО — программное обеспечение

ЭВМ — электронная вычислительная машина

# ВВЕДЕНИЕ

Компьютерная графика представляет собой совокупность методов и способов преобразования информации в графическое представление при помощи ЭВМ.

Наличие облаков сильно влияет на восприятие изображения, делая его более реалистичным. Поэтому задача визуализации реалистичных облаков чаще всего встречается при разработке компьютерных игр и в кинематографе [1—3].

Целью данной работы является реализация программного обеспечения, которое предоставляет возможность визуализировать облачное небо. При этом созданная программа должна позволить изменять параметры, влияющие на внешний вид неба.

Чтобы достигнуть поставленной цели, требуется решить следующие задачи:

— проанализировать предметную область визуализации облаков и описать существующие методы их представления;

— разработать ПО, позволяющее визуализировать облака;

— выбрать инструменты для реализации разработанного ПО;

— исследовать затраты реализации по времени выполнения.

# 1 Аналитический раздел

В данном разделе будет рассмотрена предметная область визуализации облаков.

## 1.1 Модель облаков

Облака можно представить с помощью оптической модели с рассеиванием и поглощением [2—5]. Оптическая модель описывает как объем излучает, отражает, рассеивает и поглощает свет [6].

Для визуализации облаков, необходимо рассчитать энергию светового пучка, проходящую через облачный объем. Поскольку распространяющийся луч подвергается поглощению, необходимо рассмотреть приближение закона Бугера—Ламберта—Бера [4; 6]. Энергия светового пучка, обладающего начальной энергией $c$, и прошедшего от точки $x_0$ до $x$ внутри облака, вычисляется по формуле (1.1).

$$c' = c \cdot e^{-\int_{x_0}^{x} k(u)\,du}, \tag{1.1}$$

где $c'$ — итоговая яркость пучка, $k(u)$ — коэффициент поглощения в точке пространства, а интеграл $\tau(d_1, d_2) = -\int_{d_1}^{d_2} k(u)\,du$ называют оптической глубиной [2—4; 6].

Коэффициент прозрачности в точке $u$ определяется по формуле (1.2).

$$A_u = 1 - e^{-k(u)} \tag{1.2}$$

Считается, что коэффициент поглощения $k(u)$ пропорционален оптической плотности облака в точке $u$ [2—4; 6]. Тогда для того чтобы задать форму облака, необходимо задать поле плотности облака в пространстве.

## 1.2 Методы представления облаков

Методы представления облаков должны определять пространственное распределение плотности облаков на сцене.

Были рассмотрены следующие методы представления облаков:

— система частиц;

— неявное представление;

— объемы, ограниченные поверхностью (англ. Surface-Bounded Volumes).

### 1.2.1 Система частиц

Система частиц — используемый в компьютерной графике способ представления объектов, не имеющих четких геометрических границ [7]. Облако можно представить системой частиц [8]. Каждая частица в системе задается своим положением в пространстве. Пусть $x$ — некоторая точка в пространстве, $R$ — заранее заданный радиус восприимчивости к частицам, $f(x)$ — функция распределения плотности от расстояния до частицы. Тогда алгоритм определения плотности в точке $x$ будет состоять из следующих шагов.

1) Найти все частицы, расстояние до которых от точки $x$ не больше заданного радиуса $R$.

2) Вычислить расстояния от заданной точки $x$ до отобранных частиц.

3) Для каждого вычисленного расстояния найти значений функции $f(x)$.

4) Просуммировать полученные значения функций.

Заметим, что выбор частиц в переделах заданного радиуса $R$ требует полного перебора всех частиц. Введение иерархических структур позволяет снизить количество частиц, которые необходимо рассмотреть для отбора требуемых.

**Воксельные октодеревья**

Октодерево — древовидная структура данных, в которой у каждого внутреннего узла ровно восемь потомков [9]. Каждый узел октодерева задает некоторый объем пространства, а каждый потомок этого узла описывает определенную октанту данного пространства. В родительском узле хранятся частицы, попавшие на границы октантов, а в дочернем узле частицы, принадлежащие октанту, который сопоставляется с данным дочерним узлом. Т. о. для отбора частиц в радиусе $R$ достаточно рассмотреть, только частицы, принадлежащие определенному узлу дерева [5].

**Двоичное разбиение пространства**

Двоичное разбиение пространства является методом рекурсивного разбиения пространства на выпуклые множества гиперплоскостями [10]. Каждая

гиперплоскость сопоставляется с узлом дерева, что позволяет построить двочиное дерево. Списки частиц хранятся в листьях такого дерева. Выполняя поиск по такому дереву можно найти частицы, которые необходимо рассмотреть для отбора.

**Иерархия ограничивающих объёмов**

В данном методе группы частиц окружаются оболочками и выстраивается иерархия в виде древовидной структуры данных (двоичное BVH-дерево). По аналогии с двоичным разбиением пространства, для отбора частиц необходимо рассмотреть только те частицы, которые заключены в оболочку, которая сопоставляется с листовым узлом BVH-дерева [5].

### 1.2.2 Объемы, ограниченные поверхностью

Облачный объем представляется окружающей его оболочкой, обычно заданной с помощью полигональной сетки. Полигональная сетка — это совокупность вершин, ребер и граней, которые определяют форму многогранного объекта в пространстве [11].

Поскольку информация о внутренней структуре облака отсутствует, то предполагается, что объем однородный. Тогда плотность в точке $x$ определяется по следующим шагам [5].

1) Проверить попала ли точка в какой-нибудь из ограниченных, заданных объемов.

2) Если точка принадлежит объему, то плотность в этой точке, равна заданной плотности объема.

3) Если точка не принадлежит ни одному объему, то плотность в этой точке равна 0.

### 1.2.3 Неявное представление

Распространенным способом представления поля плотности облаков является использование процедурных методов [2; 3]. В то время как общая форма обычно задается простыми геометрическими объектами, такими как сферы или эллипсоиды, внутренняя структура с высоким разрешением генерируется процедурно [5].

Для процедурной генерации могут использоваться функции шума, такие как: шум Перлина, шум Ворлея.

**Шум Перлина** — это градиентный шум, состоящий из набора псевдослучайных единичных векторов (направлений градиента), расположенных в определенных точках пространства и интерполированных функцией сглаживания, значений между этими точками [12].

**Шум Ворлея** — это шум полученный инверсией шума Вороного. Для генерации шума Вороного необходимо создать регулярную сетку некоторого размера и разместить в центре каждой ячейки точку. Далее используя хеш-функцию, каждая точка смещается в пределах ячейки. Теперь, чтобы рассчитать шум для любого пикселя, нужно определить в какую ячейку он попадает и какие ячейки ему смежны. Далее проходя по всем ячейкам, определяется расстояние между пикселем и смещенными точками, значение шума Вороного — минимальное из расстояний [13].

В итоге, для определения плотности в некоторой точке пространства, необходимо вычислить значение функции шума в данной точке. При этом, можно вычислить значения функции в некотором объеме заранее и хранить эти вычисления в памяти. Данный метод можно использовать в комбинации с полигональной сеткой [2; 3; 5; 14].

## Вывод

В таблице 1.1 представлено сравнение способов представления облаков. По каждому параметру составлен рейтинг: 1 — лучший способ, 3 — худший.

Сравнение проводится для случая, когда необходимо заполнить небо облаками. Трудоемкость вычисления плотности оценивалась без учета предварительных вычислений.

В таблице 1.1 введены следующие обозначения:

— СЧ — система частиц с использованием октодеревьев;

— ООП — объемы, ограниченные поверхностью;

— НП — неявное представление, с предварительно вычисленными значениями функций шума.

В результате для решения поставленной задачи был выбран метод неяв-

Таблица 1.1 – Сравнение методов представления облаков

| Характеристика | СЧ | ООП | НП |
|---|---|---|---|
| Реалистичность изображения | 1 | 3 | 1 |
| Трудоемкость вычисления плотности в точке | 3 | 2 | 1 |
| Требования по памяти | 2 | 1 | 3 |

ного представления. Такой выбор обусловлен сравнительно низкой трудоемкостью.

## 1.3 Анализ методов визуализации облаков

Для визуализации облаков по их пространственному представлению, необходимо провести вычисление цвета каждого пикселя в буфере кадра. Во всех описанных алгоритмах применятся смешивание цветов. Допустим у нас есть массив из $n$ элементов, содержащий значения цвета и прозрачности в некоторой точке. Тогда смешение цвета вычисляется по формуле (1.3) итеративно в порядке от последнего к первому, изменяя $i$ от $n-1$ до 0.

$$C'_i = C_i + (1 - A_i)C'_{i+1}, \tag{1.3}$$

где $C'_i$ — новое значение цвета, $C_i$ и $A_i$ — цвет и прозрачность на $i$-м шаге, $C'_{i+1}$ — значение смешанного цвета на предыдущем шаге. Начальным значением считают $C'_n = 0$.

Формула (1.3) может быть изменена для прохода по массиву от первого к последнему, т. е. $i$ от 1 до $n$.

$$C'_i = C'_{i-1} + (1 - A'_{i-1})C_i \tag{1.4}$$

$$A'_i = A'_{i-1} + (1 - A'_{i-1})A_i \tag{1.5}$$

Начальными значениями в этом случае считают $C'_0 = 0$ и $A'_0 = 0$ [6].

### 1.3.1 Алгоритм Сдвиг—Деформации

Алгоритм Сдвиг—Деформации состоит из следующих шагов [15].

1) Сформировать массив срезов объема плоскостями, параллельными оси координат $z$.

2) Каждый срез из массива сдвинуть перпендикулярно оси $z$, так чтобы луч, распространяющийся в объеме был параллелен оси $z$ (данный шаг называется сдвигом).

3) Смешать пиксели текстур срезов в некотором временном буфере, в порядке от последнего к первому.

4) Спроецировать временный буфер на буфер кадра (данный шаг называется деформацией).

### 1.3.2 Разбрызгивание текстур

Метод разбрызгивания текстур применяется чаще всего к системам частиц, но в общем случае, под частицой, в данном контексте, подразумевается точка пространства с заданной плотностью. Частицы могут быть визуализированы с помощью текстурированного четырехугольника, представляющего проекцию частицы на плоскость, также называемую «пятном» или «отпечатком» (от англ. splat и footprint соответственно).

Алгоритм разбрызгивания состоит из следующих шагов, применяемых к каждому пикселю буфера кадра [5; 8].

1) Найти частицы, пересекаемые лучом, исходящим от наблюдателя.

2) Смешать проекции текстур этих частиц в буфере кадра, в порядке от последней к первой.

### 1.3.3 Ray Marching

Алгоритм Ray Marching состоит из следующих шагов [2; 3; 5].

1) Для каждого пикселя буфера выпускается луч от наблюдателя.

2) Разбить луч на равные интервалы.

3) Выполнить итеративное смешение цветов на этих интервалах.

### 1.3.4 Выбор метода визуализации

В таблице 1.2 представлено сравнение алгоритмов визуализации облаков. По каждому параметру составлен рейтинг: 1 — лучший способ, 3 — худший.

Сравнение проводится для случая, когда необходимо заполнить небо облаками, перспектива отсутствует.

В таблице 1.2 введены следующие обозначения:

— СД — алгоритм сдвиг—деформации;

— РТ — алгоритм разбрызгивания текстур;

— RM — алгоритм Ray Marching.

Таблица 1.2 – Сравнение алгоритмов визуализации облаков

| Характеристика | СД | РТ | RM |
|---|---|---|---|
| Визуализация облаков сложных форм | 3 | 2 | 1 |
| Трудоемкость построения изображения | 1 | 3 | 2 |
| Требования по памяти | 3 | 1 | 1 |

В результате был выбран алгоритм Ray Marching, т. к. данный алгоритм подходит для визуализации облаков любой формы.

## 1.4 Модель освещения

Солнце считается точечным источником света. Угол падения света считается одинаковым для всех точек пространства в пределах сцены.

Световая энергия луча, распространяющегося в облачном объеме, может быть поглощена или отражена некоторой частицей. При этом облака являются анизатропной средой, т. е. отраженные фотоны распространяются в разных направлениях неравномерно [2—5].

Поглощение луча описывается законом Бугера—Ламберта—Бера, и рассчитывается по формуле (1.6).

$$BL = e^{-d} \tag{1.6}$$

где $d$ – накопленная плотность.

Для аппроксимации неравномерности распространения отраженного света используют фазовую функцию [2—5]. Фазовая функция описывает вероятность отражения фотона от рассеивающего объекта под определенным

углом. В качестве фазовой функции для облаков используется функция Хеньи — Гринштейна (1.7).

$$p(\theta) = \frac{1}{4\pi} \frac{1 - g^2}{(1 + g^2 - 2g\cos\theta)^{3/2}}, \tag{1.7}$$

где $g$ — варьируемый параметр, причем $-1 \leq g \leq 1$, а $\theta$ — угол между падающим и отраженным лучами [5].

Из распределения отраженного света фазовой функцией видно, что луч, после отражения будет распространяться вероятнее всего в направлении близком к исходному. Таким образом, чтобы направление луча изменилось на 180 градусов необходим достаточно длинный путь внутри облака, что приводит к большой накопленной плотности, в результате чего по формуле 1.6 энергия луча будет близка к нулю. В результате чего, наблюдатель, выпускающий луч в направлении близком к направлению падения света, будет видеть нереалистично темные облака [2]. Для того чтобы этого не происходило вводится поправка по формуле (1.8).

$$PSE = 1 - e^{-2d} \tag{1.8}$$

Для расчета энергии луча, испущенного солнцем, в некоторой точке пространства, необходимо учесть путь этого луча внутри облака до рассматриваемой точки. Для этого плотность $d$, вычисляется на нескольких точках, выбранных на луче. В итоге, энергия света, в некоторой точке пространства вычисляется по формуле (1.9).

$$E = 2 \cdot e^{-d} \cdot (1 - 2e^{-2d}) \cdot \frac{1}{4\pi} \frac{1 - g^2}{(1 + g^2 - 2g\cos\theta)^{3/2}} \tag{1.9}$$

где $g$ – некоторая константа в функции Хеньи-Гринштейна, а $\theta$ – угол между лучами взгляда и падения света.

## Вывод

В данном разделе был проведен анализ способов хранения поля плотности облаков, методов визуализации и модели освещения, которые возможно использовать для решения поставленных задач. В итоге была выбрана связка из неявного представления облаков с алгоритмом Ray Marching, так как

такой подход позволяет достичь высокой реалистичности, а также точности построенного изображения.

# 2 Конструкторский раздел

В данном разделе представлены требования к программному обеспечению, а также схемы алгоритмов, выбранных для решения поставленной задачи.

## 2.1 Требования к программному обеспечению

Программа должна предоставлять доступ к функционалу:

— задать направление взгляда камеры;

— изменение положения источника света;

— конфигурация облачного неба с помощью загрузки погодной карты;

— варьирование параметров облачного неба.

К программе предъявляются следующие требования:

— время отклика программы должно быть менее 50 мс [2];

— программа должна корректно реагировать на любые действия пользователя.

## 2.2 Разработка алгоритмов

Алгоритм Ray Marching применяется для визуализации облачного неба.

Для повышения эффективности алгоритма Ray Marching необходимо снизить количество шагов вне облака. Для этого применяются объемлющие оболочки, в данном случае, такими оболочками будут являться сферы. Когда луч пересекает оболочку, необходимо сделать $N$ шагов, на каждом из которых вычисляется плотность в точке пространства и ее освещенность, используя эти данные итеративно вычисляется значение цвета пикселя.

Как показано на рисунке 2.1, атмосфера будет моделироваться с помощью двух концентрических сфер, между которыми и будет происходить генерация облаков.

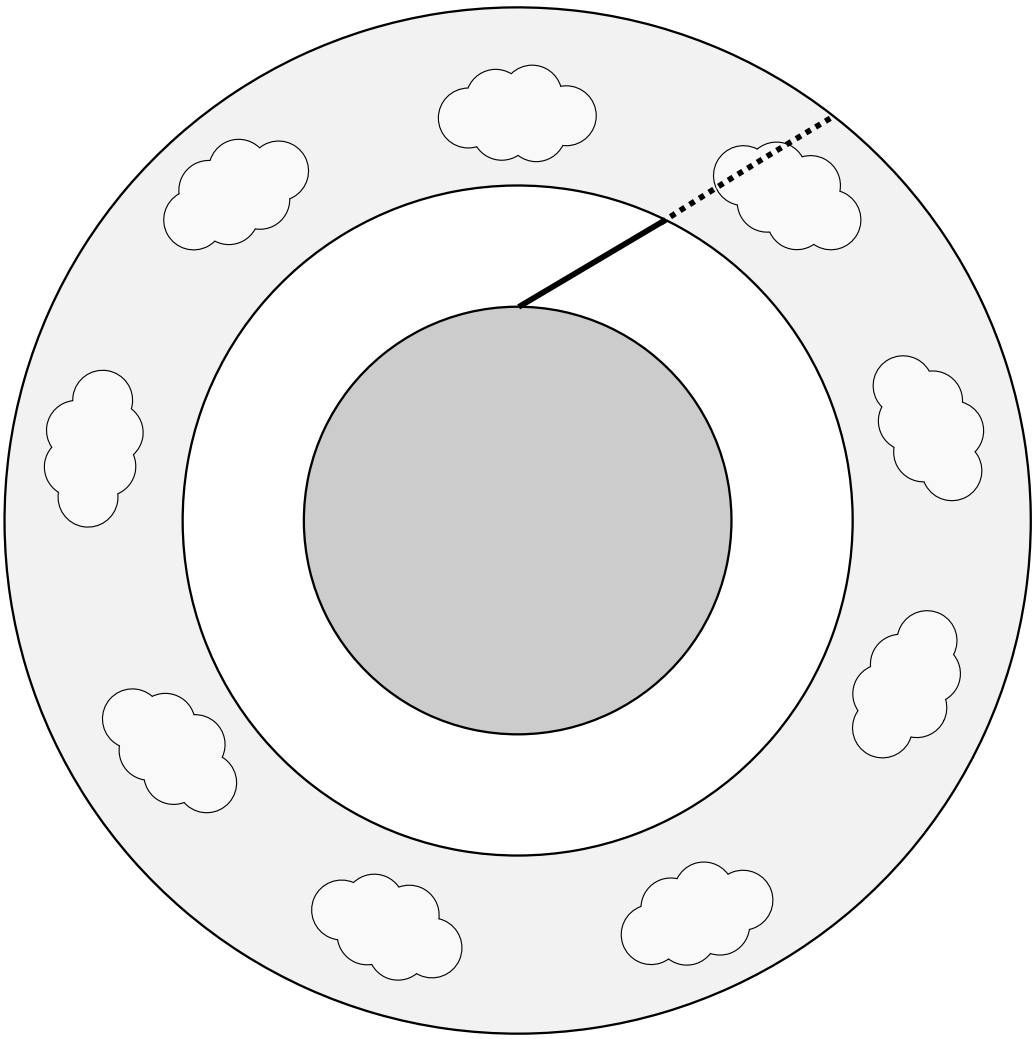

Рисунок 2.1 – Схема атмосферы и положения наблюдателя

### 2.2.1 Пересечение луча со сферой

Уравнение луча запишем следующим образом:

$$P = S + t\vec{D}, t \geq 0 \tag{2.1}$$

где $S$ - точка, откуда луч испускается, а $\vec{D}$ - направление луча.

Пусть сфера задается своим центром $C$ и радиусом $r$. Если луч пересекает сферу, тогда точка $P$ - лежит на поверхности, запишем это следующим образом:

$$\|P - C\| = r \tag{2.2}$$

Перепишем (2.2), используя скалярное произведение:

$$\sqrt{\langle P - C, P - C \rangle} = r \tag{2.3}$$

Подставим в (2.3) уравнение луча (2.1):

$$\sqrt{\langle S + t\vec{D} - C, S + t\vec{D} - C\rangle} = r \tag{2.4}$$

Обозначим $S - C = \vec{SC}$ и раскроем скалярное произведение, возведя обе части уравнения (2.4) в квадрат:

$$t^2\langle\vec{D}, \vec{D}\rangle + 2t\langle\vec{SC}, \vec{D}\rangle + \langle\vec{SC}, \vec{SC}\rangle - r^2 = 0 \tag{2.5}$$

Решая квадратное уравнение (2.5) находим точки пересечения луча с объемлющей оболочкой.

### 2.2.2 Вычисление плотности облаков в атмосфере

Как было описано в пункте 1.2.3, для представления поля плотности облаков будем хранить две объемные текстуры. Первая (основная) текстура отвечает за низкочастотный шум, имеет размер $128 \times 128 \times 128$. Вторая (вспомогательная) текстура отвечает за высокочастотный шум, имеет размер $32 \times 32 \times 32$. В таблице 2.1 показаны какие шумы используются для формирования текстур.

Таблица 2.1 – Таблица шумов.

| Текстура | Шумы | | | |
|---|---|---|---|---|
| Основная | П-В (НЧ) | В (НЧ) | В (СЧ) | В (ВЧ) |
| Вспомогательная | В (НЧ) | В (СЧ) | В (ВЧ) | - |

Примечание: П-В – шум Перлина-Ворлея, В – шум Ворлея, НЧ – низкая частота, СЧ – средняя частота, ВЧ – высокая частота.

В итоге, чтобы рассчитать плотность в некоторой точке, необходимо получить срезы шумов из текстуры и составить из них fBM (англ. fractal Brownian motion). FBM представляет собой сумму ряда октав шума, каждая из которых имеет более высокую частоту и более низкую амплитуду.

Основная текстура применяется для формирования общей грубой формы облака и может использоваться для определения факта попадания луча в облако. Вспомогательная текстура используется для добавления деталей на поверхности облака.

Для возможности формировать облачное небо используются погодные карты. Погодная карта является двумерной текстурой, и состоит из трех кана-

лов RGB, где в R канале хранится процент покрытия неба облаками, G хранит тип облаков, B канал может быть использован для хранения вероятности возникновения дождя, но в данном алгоритме он не будет использоваться.

Тип облаков будем задавать с помощью функции, которая рассеивает облако в зависимости от его высоты.

В итоге плотность в данной точке вычисляется с помощью базовой плотности, полученной из текстуры, затухания, полученного из функции типа облака и R канала погодной карты.

### 2.2.3 Общий алгоритм построения изображения

Общий алгоритм построения изображения показан на рисунке 2.2.

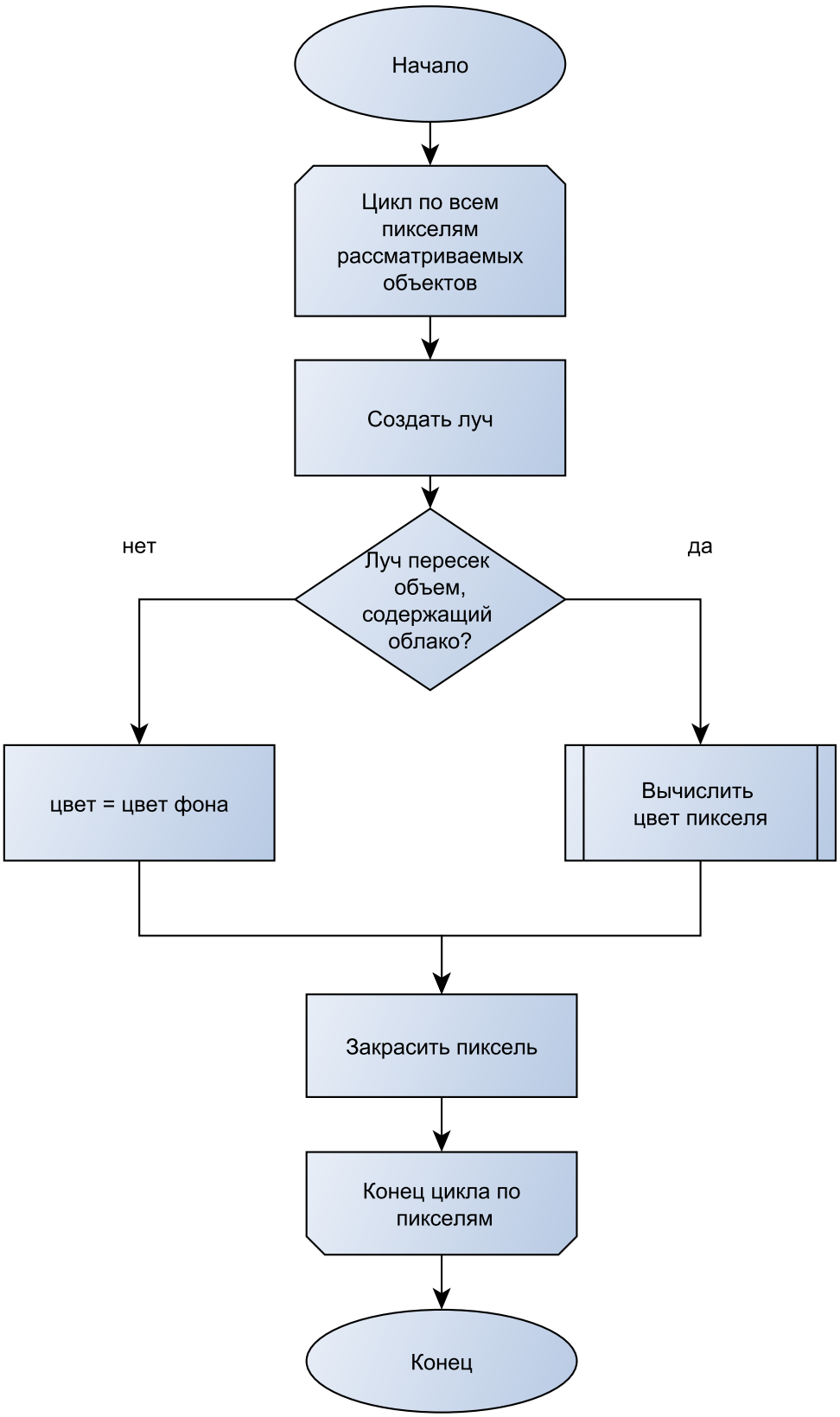


Рисунок 2.2 – Схема визуализации облаков с помощью алгоритма Ray Marching

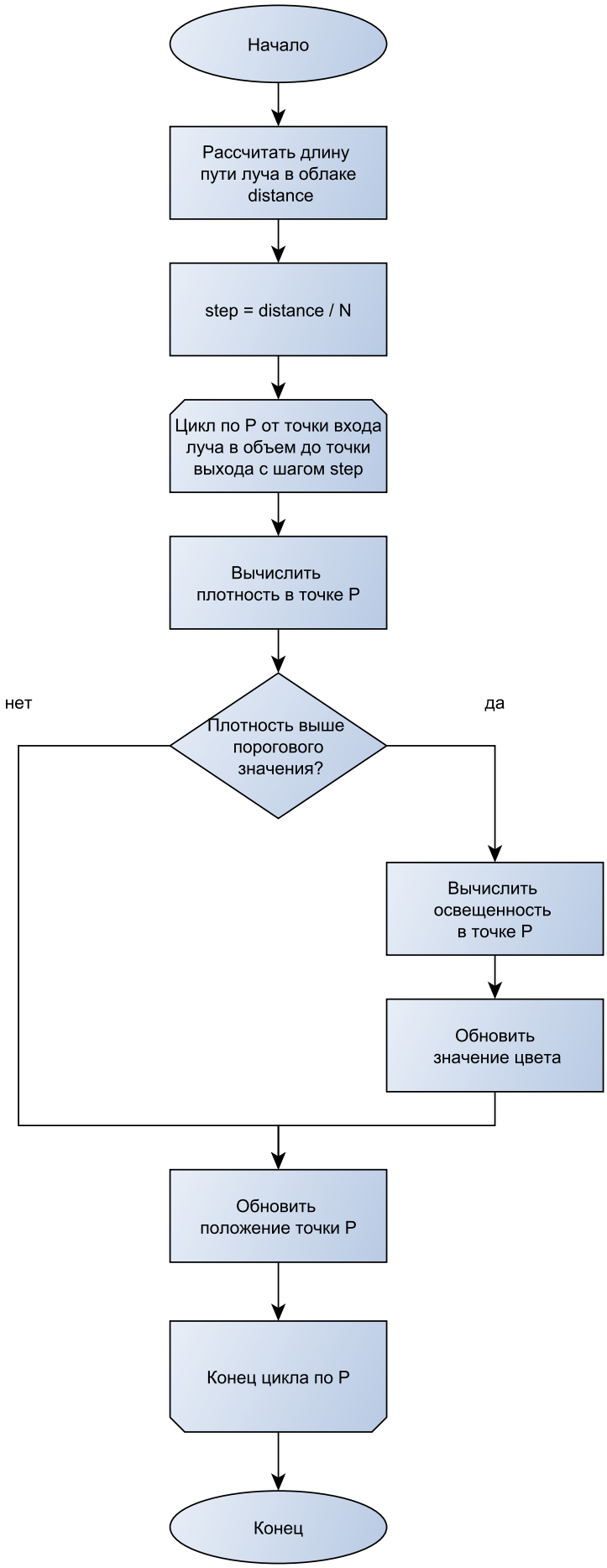


Рисунок 2.3 – Схема расчета цвета пикселя

## 2.3 Выбор используемых типов и структур данных

В данной работе используются следующие типы и структуры данных:

1) источник света – задается вектором направления и интенсивностью;
2) облака – задаются с помощью объемных текстур, погодной карты, функций рассеивания;
3) текстура – задается с помощью двумерных и трехмерных массивов, состоящих из цветов;
4) цвет – хранит три или четыре составляющие RGB или RGBA модели цвета соответственно;
5) математические абстракции:
   - точка – хранит координаты x, y, z;
   - вектор – хранит направление по x, y, z.

## Вывод

В данном разделе были представлены требования к разрабатываемому программному обеспечению и разработана схема разрабатываемого алгоритма. Так же, были описаны структуры данных, которые будут использоваться при реализации программного обеспечения.

# 3 Технологический раздел

В данном разделе будут представлены средства разработки программного обеспечения, детали реализации и процесс сборки разрабатываемого программного обеспечения.

## 3.1 Выбор средств реализации

В качестве языка программирования для разработки программного обеспечения был выбран язык *GLSL* [16]. Данный выбор обусловлен следующим:

— поддержка языком всех структур данных, выбранных в результате проектирования;

— возможность реализовать все алгоритмы, выбранные в результате проектирования.

Для создания окна был выбран язык *Python* [17], поскольку данный язык позволяет работать с библиотекой *ModernGL* [18].

## 3.2 Реализация алгоритмов

На листингах **??** – **??**, приведенных ниже, описаны основные алгоритмы используемые для визуализации облаков на языке шейдеров GLSL.

Листинг 3.1 – Реализация функции определяющей плотность в точке

```
float cloudGetHeight(vec3 position, vec2 cloudMinMax){
    return (position.y - cloudMinMax.x) / (cloudMinMax.y -
       cloudMinMax.x);
}

float saturate(float height){
    height -= 0.4;
    height *= 100;
    float v = 2 * 2 * 2 / (height * height  + 2 * 2);
    v *= 100;

    return clamp(v, 0, 1);
}

float remap(float value, float minValue, float maxValue, float
   newMinValue, float newMaxValue)
{
    return newMinValue+(value-minValue) /
        (maxValue-minValue)*(newMaxValue-newMinValue);
}

float cloudSampleDensity(vec3 position, vec2 cloudMinMax)
{
    position.xz+=vec2(0.2)*u_time;
    float base = texture(u_lfNoise, position / 48).r;
    float height = cloudGetHeight(position, cloudMinMax);

    float coverage = texture(u_weatherMap, position.xz / 480).r;
    float coff = saturate(height);

    base *= coff;

    float baseCloudWithCoverage = remap(base, 1-coverage, 1, 0,
       1);

    baseCloudWithCoverage *= coverage;

    float hfFBM = texture(u_hfNoise, position / 48).r;
    float hfNoiseModifier = mix(hfFBM, 1 - hfFBM, clamp(height *
       10, 0, 1));
```

```
37
38      float finalCloud = remap(baseCloudWithCoverage,
            hfNoiseModifier * 0.2, 1, 0, 1);
39
40
41      return max(finalCloud, 0);
42  }
```

Листинг 3.2 – Реализация алгоритма Ray Marching

```
const vec3 EXTINCTION_MULT = vec3(0.8, 0.8, 1.0);
const float CLOUD_LIGHT_MULTIPLIER = 50.0;

vec4 mainMarching(vec3 ro, vec3 viewDir, vec3 sunDir, vec3
   sunColor, vec3 ambientColor)
{
    vec2 t = rsi(ro, viewDir, minCloud);
    vec3 position = ro + viewDir * t.y;

    vec2 t2 = rsi(ro, viewDir, maxCloud);
    vec3 position2 = ro + viewDir * t2.y;

    float avrStep = (maxCloud - minCloud) / 64;

    vec2 cloudMinMax;
    cloudMinMax.x = position.y;
    cloudMinMax.y = position2.y;

    vec3 iPos = position;

    float mu = dot(viewDir, sunDir);

    vec3 transmittance = vec3(1);
    vec3 scattering = vec3(0);

    vec3 sunLight = sunColor * CLOUD_LIGHT_MULTIPLIER;
    vec3 ambient = vec3(AMBIENT_STRENGTH * sunColor) * u_ambient;

    for (int i = 0; i < 128; ++i){
        if (length(iPos) > maxCloud) break;

        float density = cloudSampleDensity(iPos, cloudMinMax);

        if (density > 0.01){
            vec3 luminance = ambient + sunLight *
               calculateLightEnergy(iPos, sunDir, mu,
               cloudMinMax);
            vec3 ttransmittance = exp(-density * avrStep *
               EXTINCTION_MULT * u_attenuation);
            vec3 integScatt = density * (luminance - luminance *
```

```
                ttransmittance) / density;
37
38              scattering += transmittance * integScatt;
39              transmittance *= ttransmittance;
40          if (length(transmittance) <= 0.01) {
41                  transmittance = vec3(0.0);
42                  break;
43              }
44          }
45          iPos += viewDir * avrStep;
46      }
47
48      transmittance = saturate3(transmittance);
49      vec3 color = ambientColor.xyz * transmittance + scattering;
50
51      return vec4(color, 1);
52  }
```

Листинг 3.3 – Реализация модели освещения

```
float HenyeyGreenstein(float g, float mu) {
    float gg = g * g;
    return (1.0 / (4.0 * PI))  * ((1.0 - gg) / pow(1.0 + gg -
       2.0 * g * mu, 1.5));
}

float cloudSampleDirectDensity(vec3 position, vec3 sunDir, vec2
   cloudMinMax)
{
    float avrStep=(cloudMinMax.y - cloudMinMax.x)*0.01;
    float sumDensity=0.0;

    for(int i=0;i<4;i++)
    {
        float step=avrStep;

        if (i==3)
            step=step*6.0;

        position+=sunDir*step;
        float density=cloudSampleDensity(position,
           cloudMinMax)*step;
        sumDensity+=density;
    }
    return sumDensity;
}

vec3 calculateLightEnergy(vec3 position, vec3 sunDir, float mu,
   vec2 cloudMinMax) {

    float density = cloudSampleDirectDensity(position, sunDir,
       cloudMinMax)* u_attenuation2;
    vec3 beersLaw = exp(-density * EXTINCTION_MULT * a) *
       HenyeyGreenstein(u_eccentrisy2, mu);
    vec3 powder = 1.0 - exp(-density * 2.0 * EXTINCTION_MULT);

    return beersLaw * mix(2.0 * powder, vec3(1.0), remap(mu,
       -1.0, 1.0, 0.0, 1.0));
}
```

## 3.3 Вывод

В данном разделе были представлены средства разработки программного обеспечения и детали реализации. В итоге был получен рендер следующего кадра:

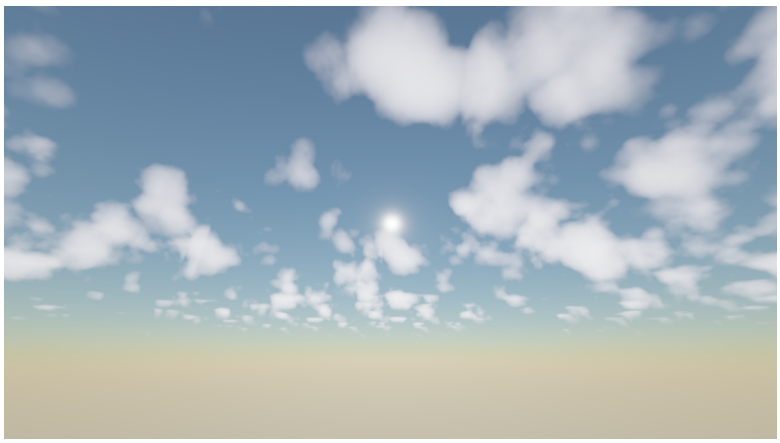

Рисунок 3.1 – Кадр, облачного неба, созданный с помощью описанных алгоритмов

# 4 Исследовательский раздел

# ЗАКЛЮЧЕНИЕ

В процессе выполнения поставленных задач были рассмотрены различные алгоритмы визуализации облаков и способы их представления. Изучены явления, происходящие в облаках, и на их основе разработана модель освещения.

Подробно были изучены алгоритм Ray Marching и неявное представление облаков. Было спроектировано программное обеспечение, позволяющее визуализировать облачное небо.

На основе проделанной работы было разработано программное обеспечение, которое строит кадр, содержащий облачное небо, приемлемого качества. В дальнейшем данное программное обеспечение можно модифицировать, добавляя интерфейс и подбирая параметры облаков.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ


1. Oz: the great and volumetric. — 2013. — Режим доступа: https://www.researchgate.net/publication/262309690_Oz_the_great_and_volumetric (дата обращения: 16.07.2023).
2. The Real-Time volumetric cloudscape of Horizon Zero Dawn. — 2015. — Режим доступа: http://killzone.dl.playstation.net/killzone/horizonzerodawn/presentations/Siggraph15_Schneider_Real-Time_Volumetric_Cloudscapes_of_Horizon_Zero_Dawn.pdf (дата обращения: 16.07.2023).
3. Physically Based Sky, Atmosphere and Cloud Rendering in Frostbite. — 2016. — Режим доступа: https://media.contentapi.ea.com/content/dam/eacom/frostbite/files/s2016-pbs-frostbite-sky-clouds-new.pdf (дата обращения: 16.07.2023).
4. A Survey on Participating Media Rendering Techniques. — 2005. — Режим доступа: https://www.researchgate.net/publication/47407939_A_Survey_on_Participating_Media_Rendering_Techniques (дата обращения: 23.07.2023).
5. A Survey of Cloud Lighting and Rendering Techniques. — 2019. — Режим доступа: https://dspace5.zcu.cz/bitstream/11025/1082/1/Hufnagel.pdf (дата обращения: 18.07.2023).
6. *Engel K.* Real-Time Volume Graphics // ACM SIGGRAPH. — 2004.
7. КОМПОНЕНТЫ И КЛАССИФИКАЦИЯ ЧАСТИЦ И ИХ СИСТЕМ В КОМПЬЮТЕРНОЙ ГРАФИКЕ. — 2016. — Режим доступа: https://www.elibrary.ru/item.asp?id=26847523 (дата обращения: 13.07.2023).
8. A simple, efficient method for realistic animation of clouds. — 2000. — Режим доступа: https://www.researchgate.net/publication/220721744_A_simple_efficient_method_for_realistic_animation_of_clouds (дата обращения: 13.07.2023).
9. Octree C++ Class TemplaternGL. — 2022. — Режим доступа: http://nomis80.org/code/octree.html (дата обращения: 05.11.2023).
10. *А. М. А.* Параллельный алгоритм поиска ближайшей точки в радиусе // Машиностроение и компьютерные технологии. — 2013. — № 11.

11. *О.М. Р.* МЕТОДЫ УСТРАНЕНИЯ ШУМА НА ПОЛИГОНАЛЬНЫХ СЕТКАХ // Теория и практика современной науки. — 2021. — № 1.

12. *Качурин А.В. И. В.* ОСНОВЫ ФОРМИРОВАНИЯ ВИЗУАЛЬНОЙ КАРТИНЫ ШУМА ПЕРЛИНА ПРИ ПРОГРАММИРОВАНИИ НА ЯЗЫКЕ PYTHON // E-Scio. — 2021. — №8.

13. A Cellular Texture Basis Function. — 1996. — Режим доступа: `https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.95.412&rep=rep1&type=pdf` (дата обращения: 26.07.2023).

14. Interactive multiple anisotropic scattering in clouds. — 2008. — Режим доступа: `https://inria.hal.science/inria-00333007/document` (дата обращения: 28.07.2023).

15. *Philip Lacroute M. L.* Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation // ACM Computer Graphics. — 1994.

16. Core Language (GLSL). — 2021. — Режим доступа: `https://www.khronos.org/opengl/wiki/Core_Language_(GLSL)` (дата обращения: 06.11.2023).

17. Python. — 2001. — Режим доступа: `https://www.python.org/` (дата обращения: 20.08.2023).

18. ModernGL. — 2022. — Режим доступа: `https://moderngl.readthedocs.io/en/latest/index.html` (дата обращения: 20.08.2023).