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;

— спроектировать 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]. Данная модель может быть описана функцией (1.1).

$$L(x, \vec{\omega}) = \underbrace{\rho(x_0, x) L(x_0, \vec{\omega})}_{L_{ri}(x, \vec{\omega})} + L_m(x, \vec{\omega}) \tag{1.1}$$

где $L(x, \vec{\omega})$ — яркость луча света в точке x, излучаемого в направлении $\vec{\omega}$; x_0 — точка в пространстве откуда был испущен луч света; $\rho(x_1, x_2)$ — функция, описывающая поглощение луча при распространении его от точки x_1 до точки x_2. Итоговая яркость луча формируется из двух составляющих: $L_{ri}(x, \vec{\omega})$ — яркости света, формируемой исходным лучом; $L_m(x, \vec{\omega})$ — яркости света, формируемой явлениями внутри среды, например излучением [4].

Поскольку распространяющийся в облаках луч подвергается поглощению, рассматривают закон Бугера—Ламберта—Бера [4; 6]. Интенсивность светового пучка, обладающего начальной интенсивностью I_0, и прошедшего от точки x_0 до x внутри облака, вычисляется по формуле (1.2).

$$I = I_0 \cdot e^{-\int_{x_0}^{x} k(u)\, du}, \tag{1.2}$$

где I — итоговая интенсивность пучка, $k(u)$ — показатель ослабления, а интеграл $\tau(x_1, x_2) = \int_{x_1}^{x_2} k(u)\, du$ называют оптической толщиной [2—4; 6].

Коэффициент прозрачности в точке u определяется по формуле (1.3).

$$A_u = 1 - e^{-k(u)} \tag{1.3}$$

Поскольку значение $k(u)$ зависит от плотности вещества, то для того чтобы задать форму облака, необходимо задать распределение плотности в пространстве [2—4; 6].

1.2 Методы представления облаков

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

Были рассмотрены следующие методы представления облаков:

— система частиц;

— процедурное;

— 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]. 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.

Особенности метода представления:

— для определения плотности необходимо выполнить полный перебор;

— возможно создавать облака любой формы;

— качество результат и количество вычислений зависят от количества 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.

Особенности данного метода:

— возможно задать облака любой формы;

— для определения плотности в точке, необходимо рассмотреть все облака, чтобы понять пересек ли луч одно из них;

— предположение, что облака однородны внутри снижает их реалистичность.

1.2.3 Процедурное представление

Распространенным способом представления распределения плотности облаков является использование процедурных методов [2; 3]. В данном подходе форма облака может задаваться простыми геометрическими объектами, такими как сферы или эллипсоиды, а внутренняя структура с высоким разрешением генерируется процедурно [5].

Для процедурной генерации могут использоваться функции шума, такие как: шум Перлина, шум Ворлея [2; 3].

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 инвертированием шума Вороного. Для генерации шума Вороного необходимо создать регулярную сетку некоторого размера и разместить в центре каждой ячейки точку. Далее каждая точка смещается в пределах ячейки случайным образом. 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 вычисленных 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 [2; 3; 5; 14].

1.2.4 Сравнение методов представление облаков

Сравнение методов будет производится для представления облака параллелепипедной формы, размером $n \times n \times n$ вокселей. Для всех алгоритмов будет рассмотрена худшая ситуация.

Для задания неоднородного распределения плотности внутри облака предполагается использование частиц с радиусом в пределах одного вокселя, тогда в объеме $n \times n \times n$ будет размещено n^3 частиц. Т. о. для определения плотности в некоторой точке необходимо рассмотреть n^3 частиц, а для их хранения необходимо $n^3 \cdot C$ байт памяти, где C — объем памяти в байтах, занимаемый одной частицей.

В свою очередь, для объемов ограниченных поверхностью (в качестве поверхности используется полигональная сетка), для того чтобы задать такое же облако необходимо хранить в памяти восемь верши и двенадцать полигонов, а для определения попадания точки в облако необходимо проверить принадлежность этой точки ко всем облакам на сцене, в данном случае, принадлежность одному облаку. Т. о. плотность в точке вычисляется за одно действие и не зависит от размера облака, а объем занимаемой памяти равен $8 \cdot C + 12 \cdot D$, где C — объем памяти в байтах, занимаемый одной вершиной, а D — объем памяти, занимаемый одним полигоном.

Для процедурного представления облака, может быть задана также объемлющая оболочка с помощью полигональной сетки, для хранения которой необходимо $8 \cdot C + 12 \cdot D$ байт памяти, где C — объем памяти в байтах, занимаемый одной вершиной, а D — объем памяти, занимаемый одним полигоном. При этом неоднородность плотности внутри облака, будет задаваться с помощью предварительно процедурно вычисленной текстуры, размер которой $n \times n \times n$, а объем занимаемой памяти $n^3 \cdot E$, где E — объем памяти в байтах, занимаемый одним вокселем текстуры. Размер текстуры меньше, поскольку

она дублируется в пространстве, чтобы занять весь объем. Т. о. суммарные требования по памяти $8 \cdot C + 12 \cdot D + n^3 \cdot E$, а для определения плотности в точке, необходимо определить попала ли точка в облако и получить значение плотности из текстуры, т. е. так же одно действие.

В результате, с учетом того, что $D < C < E$. Получаем, что для представления облака заданной формы и размера, большие памяти будет требоваться для представления облака с помощью процедурной генерации, затем с помощью системы частиц и после с помощью объемлющих оболочек. При этом вычисление плотности в точке для процедурной генерации и для объемлющих оболочек одинаково по трудозатратам и меньше, чем для системы частиц.

Вывод

В результате для решения поставленной задачи был выбран метод процедурного представления, поскольку данный метод позволяет получить плотность в точке пространства за наименьшее количество действий и, в отличии от метода объемлющих оболочек, позволяет учитывать неоднородность внутреннего объема облака.

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.

Имеется массив из n элементов, содержащий значения цвета и прозрачности в некоторой точке. Тогда смешение цвета вычисляется по формуле (1.4) итеративно в порядке от последнего к первому, изменяя i от $n-1$ до 0.

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

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 (1.4) может быть изменена для прохода по массиву от первого

к последнему, т. е. i от 1 до n.

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

$$A'_i = A'_{i-1} + (1 - A'_{i-1})A_i \tag{1.6}$$

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).

Особенность алгоритма заключается в том, что выбор большого шага среза приведет к меньшему количеству вычислений, но при этом возможна потеря резких форм облака.

1.3.2 Разбрызгивание текстур

Алгоритм разбрызгивания текстур применяется для визуализации системы частиц, но в общем случае, под частицой, в данном контексте, подразумевается точка пространства с заданной плотностью [5]. Частицы могут быть визуализированы с помощью текстурированного четырехугольника, представляющего проекцию частицы на плоскость, также называемую «пятном» или «отпечатком» (от англ. splat и footprint соответственно) [5; 8].

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

1) Найти частицы, пересекаемые лучом, исходящим от наблюдателя и приходящим в некоторый пиксель буфера кадра.

2) Смешать проекции текстур этих частиц в буфере кадра, в порядке возрастания расстояния от наблюдателя.

Особенность алгоритма состоит в том, что необходимо найти частицы, отпечаток которых попадает на выбранный пиксель и при этом хранить такие частицы нужно будет в порядке возрастания удаленности от наблюдателя.

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.

Особенность алгоритма заключается в том, что за счет итеративного смешивания цветов нет необходимости в сохранении порядка интервалов, а итоговый цвет можно вычислять по ходу распространения луча.

1.4 Сравнение алгоритмов визуализации

Сравнение алгоритмов будет проводится для облака, описанного в пункте 1.2.4. Сравнение производится по худшему случаю.

Для алгоритма сдвиг деформации важен выбор шага, который влияет на точность визуализации. В дальнейшем будет считаться, что шаг среза равен одному. Тогда для данного алгоритма необходимо будет сформировать n срезов размера $n \times n$ и один дополнительный буфер размера $n \times n$, для хранения которых требуется $(n + 1) \cdot (n^2 \cdot A)$, где A — объем памяти, занимаемый одной ячейкой среза. Для сдвига необходимо изменить координаты каждого пикселя в каждом срезе, т. о. необходимо выполнить порядка $n \cdot n^2$ операций [15]. После сдвига производится смешивание текстур во временном буфере, т. е. производится еще порядка $n \cdot n^2$ операций и далее полученный буфер проецируется на буфер кадра за порядка n^2 операций. В итоге для

визуализации одного облака алгоритмом Сдвиг—Деформации необходимо порядка $O(n^3)$ операций.

В алгоритме разбрызгивания текстур, будет считаться, что каждый луч, испущенный наблюдателем, пересек n частиц. Тогда для каждого луча требуется дополнительно $n \cdot B$ байт памяти, где B — объем памяти в байтах, занимаемый одной частицей. В итоге, для каждого пикселя буфера кадра вычисляются проекции частиц, т. о. для визуализации облака необходимо выполнить порядка $N \cdot M \cdot n \cdot \alpha$ операций, где размер буфера кадра $N \times M$, α — количество операций, требуемое для определения текстур частиц, которые пересекаются с лучом наблюдателя. α зависит от способа представления частиц, в худшем случае, эта операция требует полного просмотра частиц, т. е. в худшем случае визуализация выполняется за порядка $N \cdot M \cdot n \cdot n^3$ операций.

В алгоритме Ray Marching, испущенный луч хранит результирующий цвет некоторого пикселя в буфере кадра, т. е. требуется D байт памяти, где D — объем памяти, требуемый для хранения цвета. Для визуализации облака требуется порядка $N \cdot M \cdot \beta$ операций, где размер буфера кадра $N \times M$, β — количество операций, требуемое для определения цвета в некоторой точке луча. В худшем случае, если для представления облаков используется система частиц, то визуализация выполняется за порядка $N \cdot M \cdot n^2$ операций.

В результате, поскольку обычно размеры буфера кадра велики, то минимальное количество операций необходимо для алгоритма Сдвиг—Деформации, затем для алгоритма Ray Marching, и больше всего операций требуется для визуализации облака с помощью системы частиц. При этом алгоритм Ray Marching требует меньше памяти, чем алгоритм Сдвиг—Деформации.

Вывод

В результате для визуализации облаков был выбран алгоритм Ray Marching, поскольку данный алгоритм требует меньше всего памяти и при этом обладает приемлемой оценкой количества операций.

1.5 Модель освещения

Поскольку солнце сильно удалено от Земли, то лучи распространяются параллельно, т. е. 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].

Закон Бугера—Ламберта—Бера (1.2) описывает явление затухания луча при его распространении в облаках, приближенно интеграл будет рассчитываться как сумма, тогда закон можно записать следующим образом (1.7).

$$BL = e^{-d} \tag{1.7}$$

где d — накопленная оптическая толщина.

Для аппроксимации явления неравномерного 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 (1.8).

$$p(\theta) = \frac{1}{4\pi} \frac{1 - g^2}{(1 + g^2 - 2g\cos\theta)^{3/2}}, \tag{1.8}$$

где g — варьируемый параметр, причем $-1 \leq g \leq 1$, а θ — 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 1.7 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 (1.9).

$$PSE = 1 - e^{-2d} \tag{1.9}$$

Для расчета энергии луча, испущенного солнцем, в некоторой точке

пространства, необходимо учесть путь этого луча внутри облака до рассматриваемой точки. Для этого накопленная оптическая толщина d, вычисляется на нескольких точках, выбранных на луче. В итоге, энергия света, в некоторой точке пространства вычисляется по формуле (1.10).

$$E = 2 \cdot e^{-d} \cdot (1 - 2e^{-2d}) \cdot \frac{1}{4\pi} \frac{1 - g^2}{(1 + g^2 - 2g\cos\theta)^{3/2}} \tag{1.10}$$

где g — некоторая константа в функции Хеньи-Гринштейна, а θ — угол между лучами взгляда и падения света.

1.6 Постановка задачи

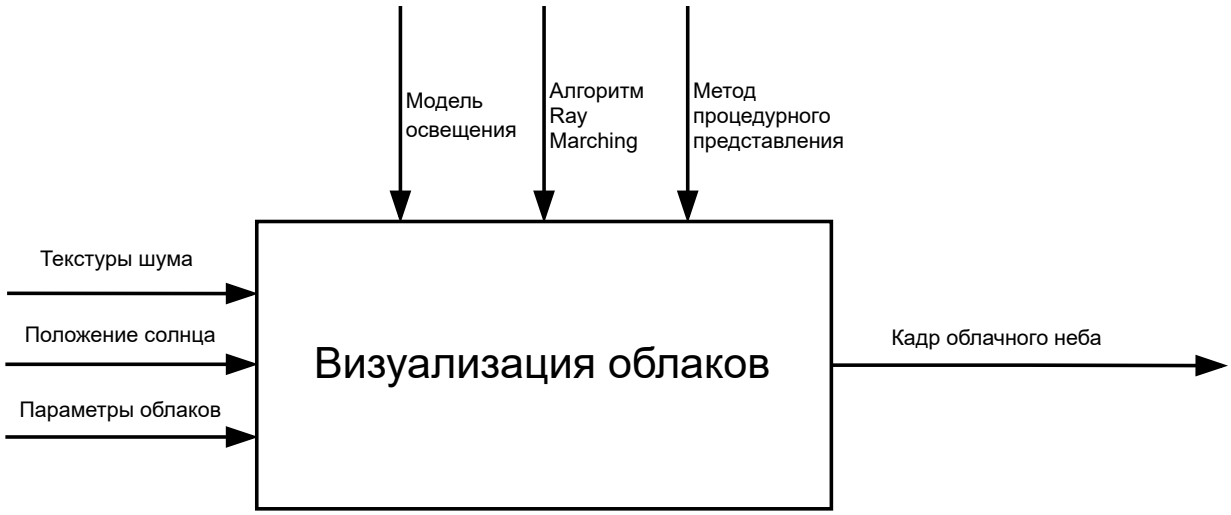

Рисунок 1.1 – Диграмма IDEF0 постановки задачи нулевого уровня

Вывод

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

В таблицах 1.1 и 1.2 представлено сравнение методов представления облаков и алгоритмов визуализации. По каждому параметру составлен рейтинг: 1 — лучший способ, 3 — худший.

Таблица 1.1 – Сравнение методов представления облаков

Характеристика	СЧ	ООП	П
Учет неоднородности	Да	Нет	Да
Трудоемкость вычисления плотности в точке	2	1	1
Требования по памяти	2	1	3

Примечание: СЧ — система частиц; ООП — объемы, ограниченные поверхностью; П — процедурное представление, с предварительно вычисленными значениями функций шума.

Таблица 1.2 – Сравнение алгоритмов визуализации облаков

Характеристика	СД	РТ	RM
Трудоемкость построения изображения	1	3	2
Требования по памяти	3	2	1

Примечание: СД — алгоритм сдвиг—деформации; РТ — алгоритм разбрызгивания текстур; RM — алгоритм Ray Marching.

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 обладать следующей функциональностью:

— задать направление взгляда камеры;

— изменение направление лучей света;

— варьирование параметров облачного неба.

Программа должна корректно реагировать на любые действия пользователя.

2.2 Разработка алгоритмов

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.

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

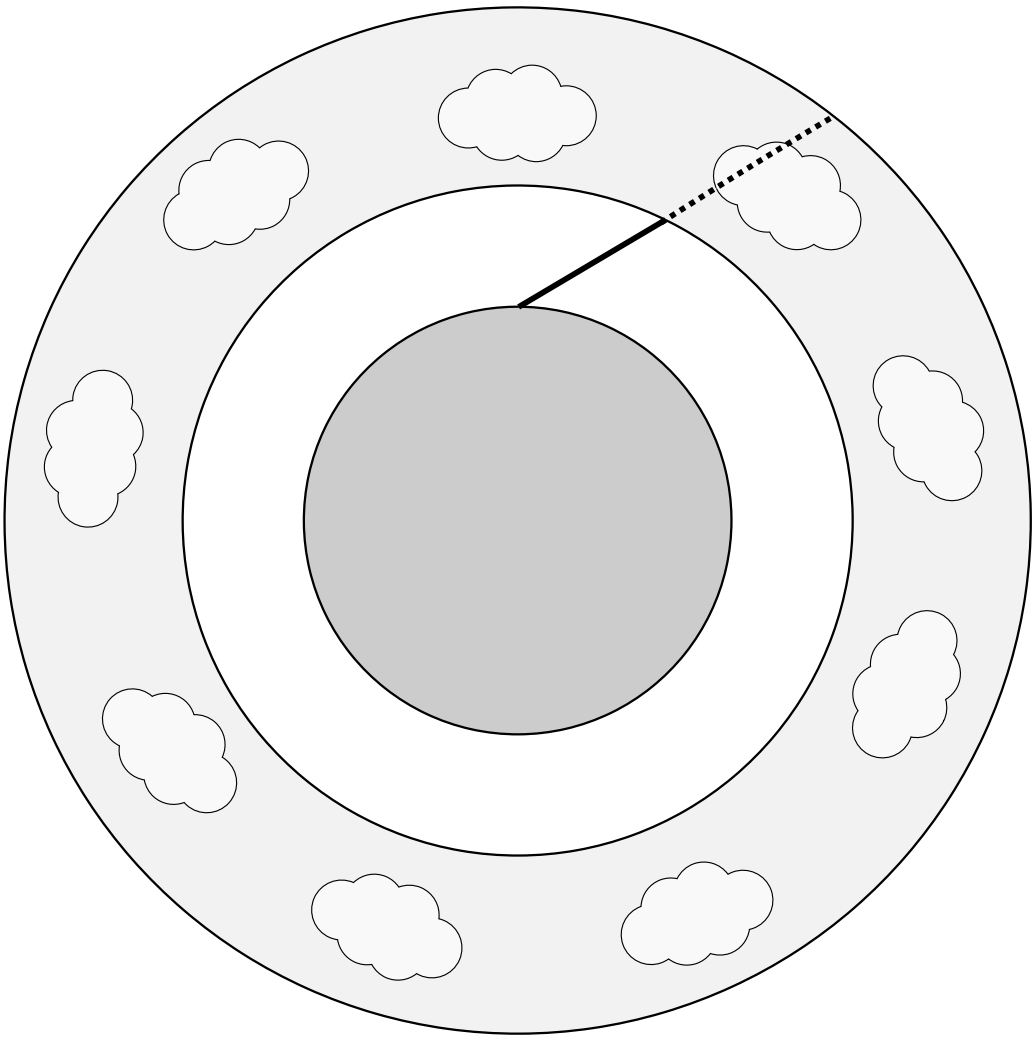

Рисунок 2.1 – Схема атмосферы и положения наблюдателя

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:

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

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

2.2.2 Вычисление плотности облаков в атмосфере

Для вычисления плотности внутри облака будут использоваться две 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.

Для возможности формировать облачное небо используются погодные карты. Погодная карта является двумерной текстурой, и состоит из четырех каналов RGBA, где в R канале хранится коэффициент покрытия неба облаками на малой высоте, G хранит покрытие облаками на большой высоте, B канал хранит максимальную высоту облаков, A канал хранит коэффициент плотности облаков.

2.2.3 Общий алгоритм построения изображения

Общий алгоритм построения изображения показан на рисунке 2.2. Алгоритм вычисления плотности в точке изображен на рисунке 2.3. Алгоритм вычисления яркости в точке изображен на рисунке 2.4.

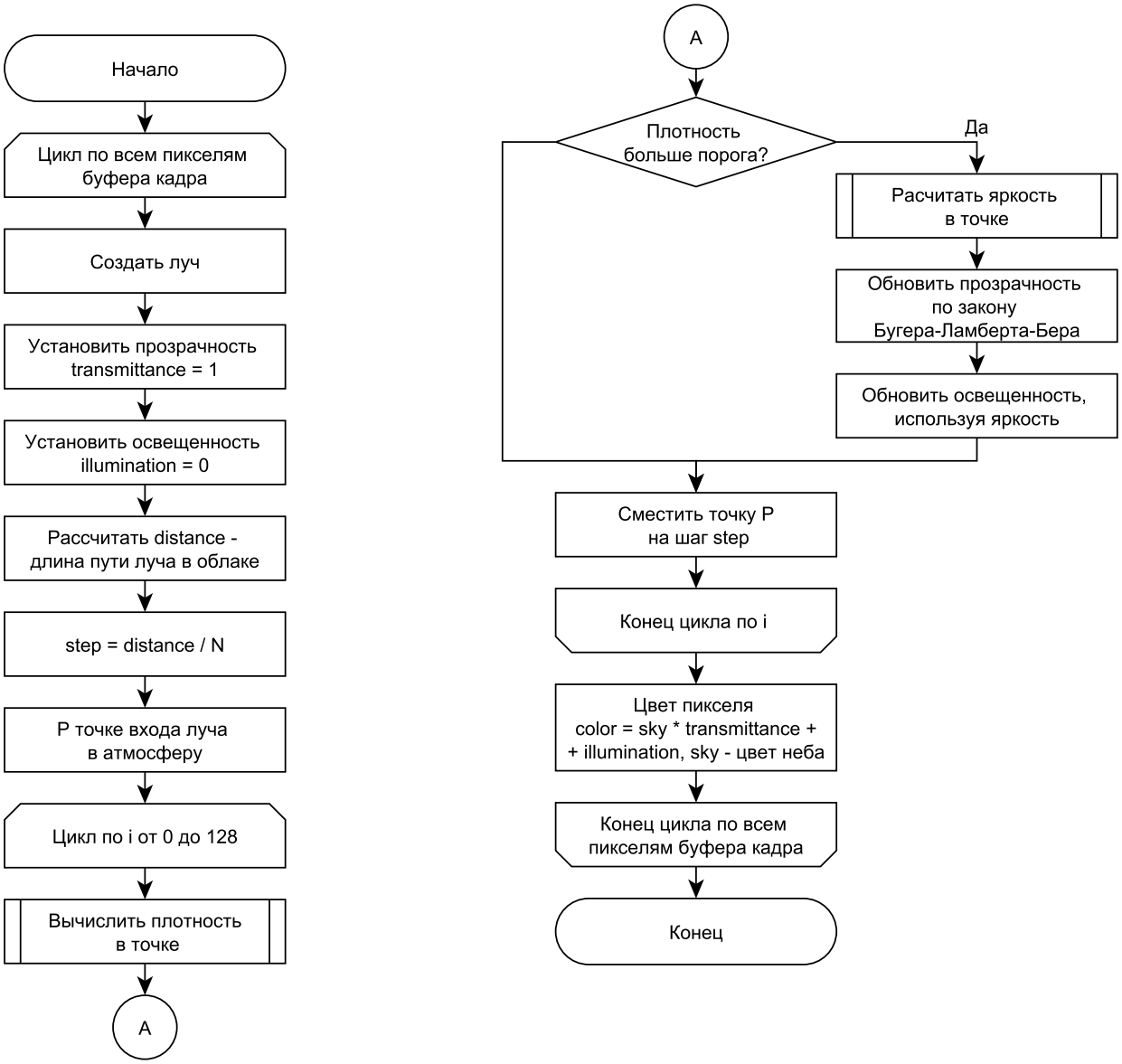

Рисунок 2.2 – Схема визуализации облаков с помощью алгоритма Ray Marching

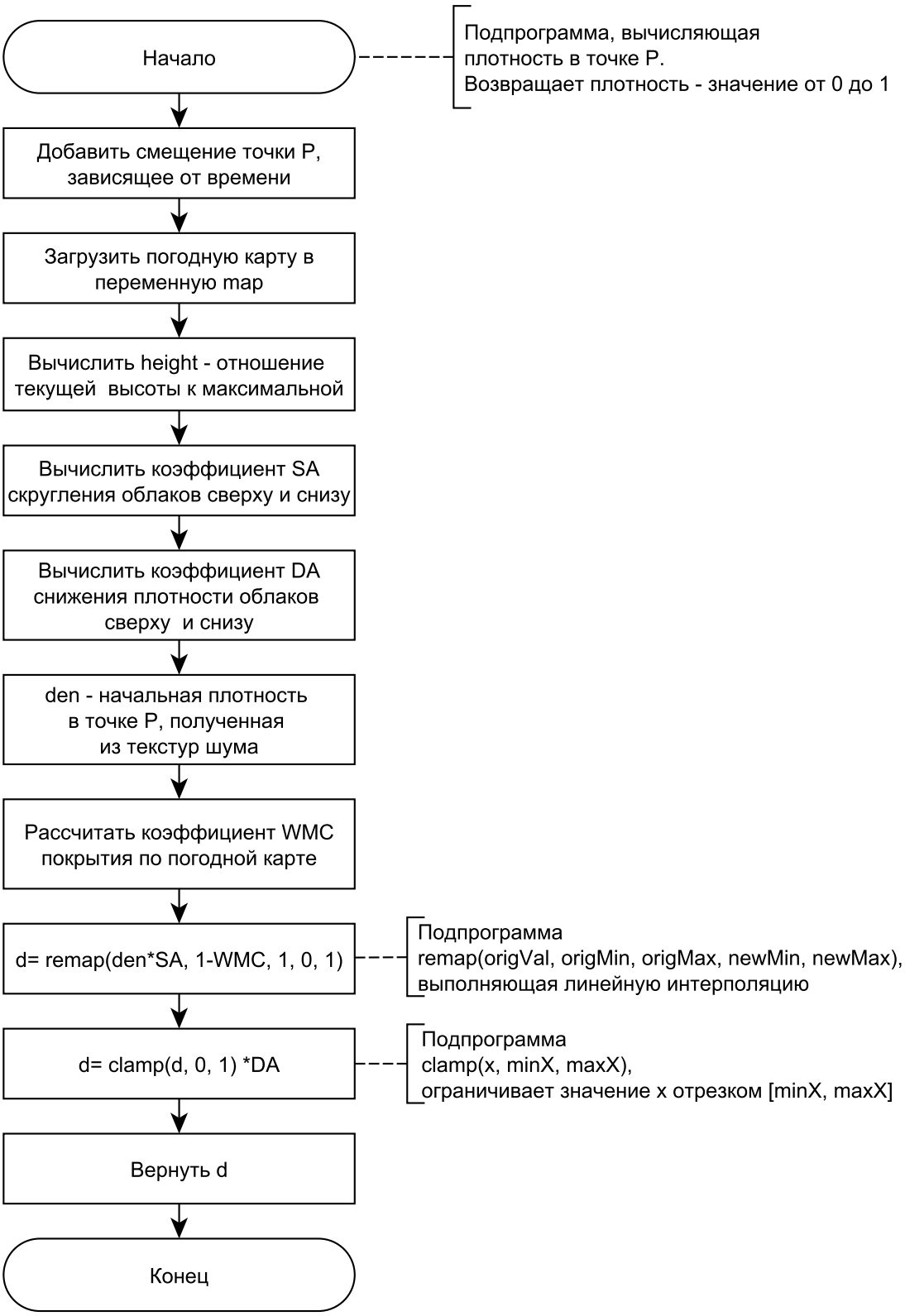

Рисунок 2.3 – Схема вычисления плотности в точке

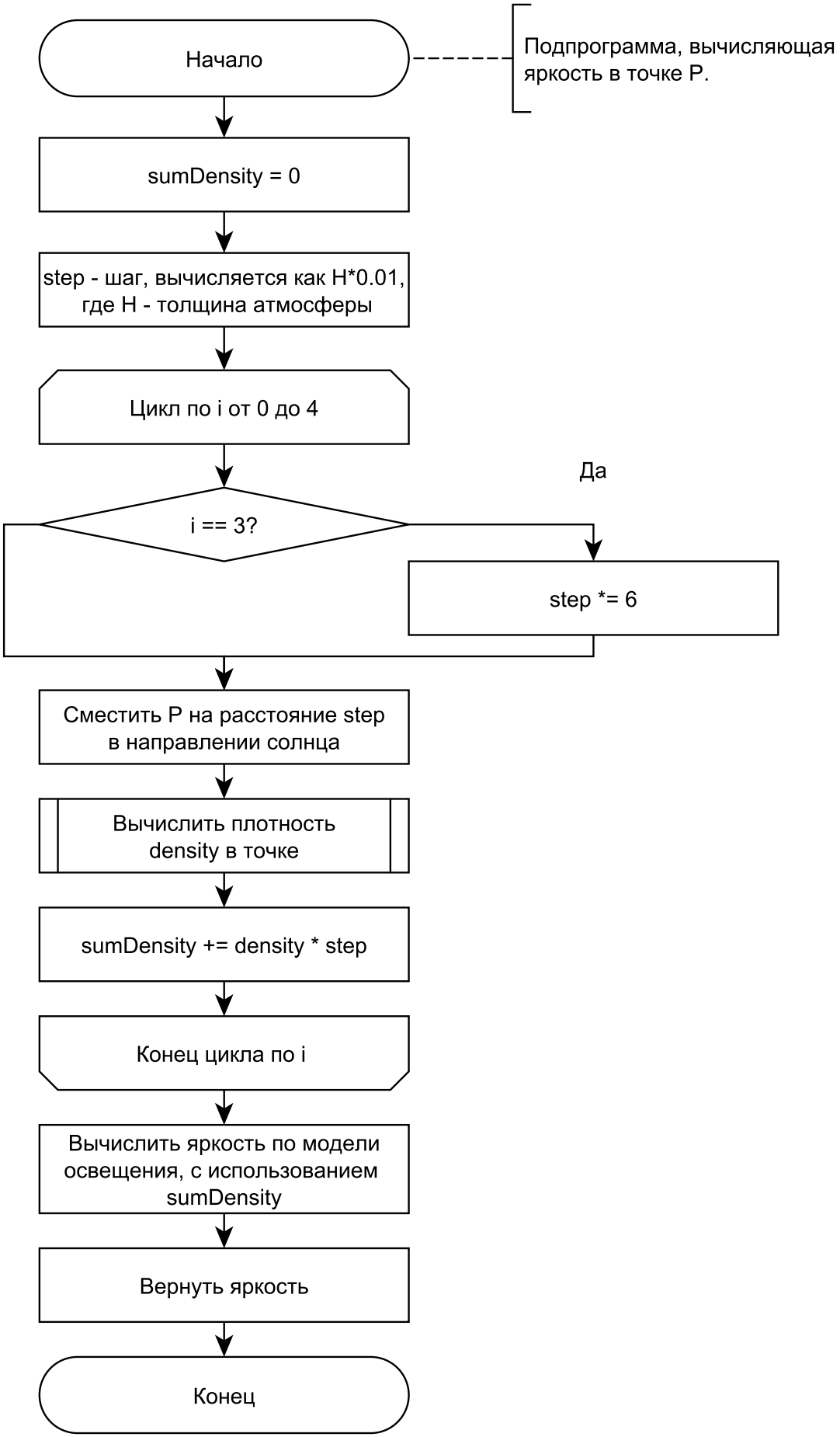

Рисунок 2.4 – Схема вычисления яркости в точке

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:

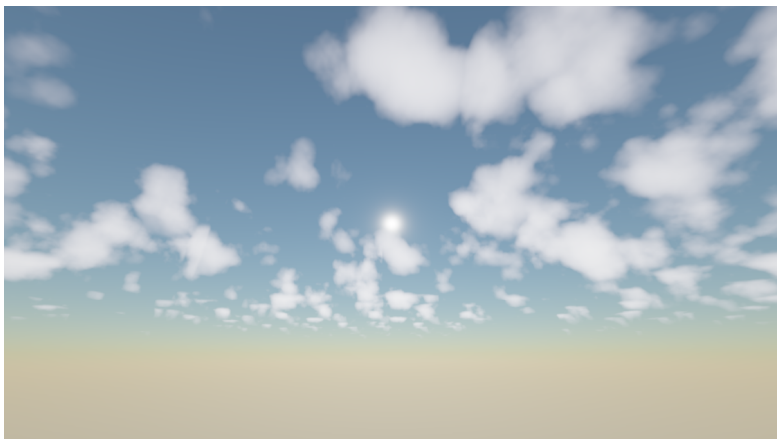

Рисунок 3.1 – Кадр, облачного неба, созданный с помощью описанных алгоритмов

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).