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

КУРСОВАЯ РАБОТА

НА ТЕМУ:

Визуализация ландшафтной сцены с облаками

Студент ИУ7-54Б (группа) _____ (подпись, дата) Т. А. Асадуллин (И.О. Фамилия)

Руководитель курсового проекта _____ (подпись, дата) К. А. Кивва (И.О. Фамилия)

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ **5**

1 Аналитический раздел **6**

1.1 Формализация задачи и объектов 6

1.2 Оптическая модель облаков 7

1.2.1 Закон Бугера — Ламберта — Бера 7

1.2.2 Фазовая функция Хеньи – Гринстейна 7

1.3 Алгоритмы визуализации облаков 7

1.3.1 Жидкостная симуляция 8

1.3.2 Воксельная генерация 9

1.3.3 Визуализация на основе трассировки лучей 9

1.4 Способы представления ландшафта 11

1.4.1 Аппроксимация примитивами 11

1.4.2 Аппроксимация неявными кривыми 11

1.5 Шумы для процедурной генерации 11

1.5.1 Шум Перлина 12

1.5.2 Шум Ворли — Вороного 12

1.6 Модель освещения 13

1.6.1 Модель Фонга 13

1.6.2 Модель Ламберта 14

1.7 Учёт теней от облаков на поверхности 15

1.8 Удаление невидимых линий и поверхностей 15

1.8.1 Алгоритм Робертса 15

1.8.2 Алгоритм Варнока 15

1.8.3 Алгоритм Вейлера-Азертона 15

1.8.4 Алгоритм художника 16
1.8.5 Метод обратной трассировки лучей 16
1.8.6 Алгоритм, использующий Z-буфер 16
1.9 Методы закраски 17
1.9.1 Простая закраска 17
1.9.2 Закраска методом Гуро 17
1.9.3 Закраска методом Фонга 17

2 Конструкторский раздел 19
2.1 Функциональная диаграмма 19
2.2 Схемы алгоритмов 20
2.3 Диаграмма классов 24

3 Технологический раздел 26
3.1 Реализация алгоритмов 26
3.2 Модульное тестирование 31
3.3 Интерфейс 31

4 Исследовательский раздел 35
4.1 Цель исследования 35
4.2 Исследование 35

ЗАКЛЮЧЕНИЕ 38

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 39

Приложение А 41

ВВЕДЕНИЕ

Погода оказывает значительное влияние на общее эмоциональное состояние человека [1]. Это важно, например, для компьютерных игр, где множество факторов задействовано для создания нужного настроения в сценах, представленных игроку. Облака на небе – один из наиболее значимых факторов, задающих настроение и формирующих пейзаж в компьютерных играх [1].

В итоге динамическая визуализация реалистичных облаков стала чрезвычайно востребованной для таких приложений, как компьютерные игры с открытым миром, системы имитации полетов и среды виртуальной реальности [1, 2].

Цель работы – разработка программного обеспечения для визуализации динамической ландшафтной сцены с облаками.

Для достижения поставленной цели необходимо решить следующие задачи:

— проанализировать предметную область: рассмотреть известные подходы и алгоритмы для генерации и визуализации облаков и ландшафта;

— описать оптическую модель облаков;

— проанализировать и выбрать алгоритмы решения основных задач компьютерной графики: удаления невидимых линий и поверхностей, учёта теней и освещения;

— спроектировать программное обеспечение, позволяющее визуализировать динамическую ландшафтную сцену с облаками;

— выбрать средства реализации этого программного обеспечения и создать его;

— провести исследование разработанного программного обеспечения.

1 Аналитический раздел

В аналитическом разделе формализованы задачи и объекты сцены, определены геометрические и оптические характеристики объектов сцены. Также проанализированы и описаны алгоритмы, используемые для визуализации и генерации ландшафтной сцены с облаками. Установлены допустимые диапазоны и ограничения, накладываемые на входные данные.

1.1 Формализация задачи и объектов

Объектами сцены являются:

1) облака;
   - высота, на которой находятся облака;
   - ширина: горизонтальное пространство, которое облака занимают на небе;
   - скорость движения облаков в горизонтальной плоскости;
   - кучность: степень сгущённости и концентрации облаков, что влияет на их внешний вид и отбрасываемую тень.
   - плотность: определяет, сколько солнечного света облака могут заблокировать, что влияет на освещённость ландшафта.
2) ландшафт;
   - рельеф: равнина — земная поверхность без гор и значительных холмов [3];
   - материалы и текстуры: характеристики поверхности, такие как цвет и диффузное отражение.
3) бесконечно удаленный источник света (солнце);
   - расположение: определяется азимутальным и зенитным углом на небесной сфере от 0° до 180°.
   - интенсивность: яркость, с которой будет освещена сцена.
4) наблюдатель (камера).
   - расположение: координаты и угол обзора камеры, позволяющие наблюдать сцену с разных ракурсов;
   - поле зрения: угол обзора, влияющий на ширину сцены.

1.2 Оптическая модель облаков

1.2.1 Закон Бугера — Ламберта — Бера

Для облаков некоторая часть света рассеивается от направления распространения, а еще большее количество поглощается каплями воды и молекулами озона, но остается часть, которая продолжает движение без изменений [2, 4].

Закон Бугера—Ламберта—Бера определяет ослабление пучка света при поглощении средой.

$$I_l = I_o e^{-k_\lambda l}, \tag{1.1}$$

где I_0 – интенсивность света на входе в вещество, l – расстояние, прошедшее светом, k_λ – показатель поглощения среды.

1.2.2 Фазовая функция Хеньи – Гринстейна

Облака представляют собой анизотропную среду (среда, где физические свойства: показатели преломления, скорость распространения и пр. – различаются в различных направлениях внутри этой среды) из-за того, что облака представляют собой капли жидкой воды и кристаллов ледяного льда [1]. Для описания этого используют фазовую функцию (индикатриса) Хеньи – Гринстейна [2, 4]

Фазовая функция Хеньи – Гринстейна определяет угловое распределение интенсивности для наблюдателя:

$$P(g, \theta) = \frac{1 - g^2}{(1 + g^2 - 2g\cos\theta)^{3/2}} \tag{1.2}$$

где θ – угол между направлением распространения света и взгляда наблюдателя и g – параметр асимметрии, который описывает среднее значение косинуса угла рассеяния.

1.3 Алгоритмы визуализации облаков

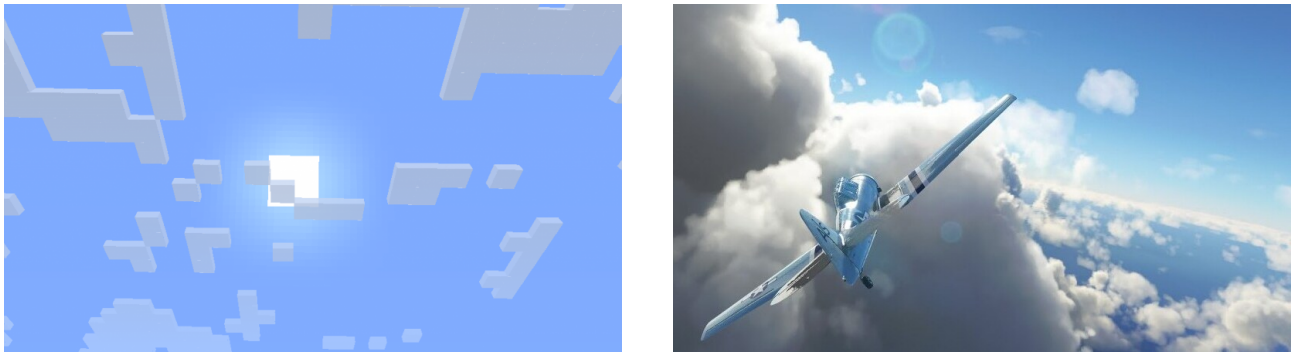
Существует несколько подходов к реализации облаков [2, 4, 5]:

— геометрический: облака представляют собой, например, набор тре-

угольников, сфер или прямоугольников. Геометрический подход к созданию облаков имеет смысл для создания изображения в определённой стилистике (Рисунок 1.1 (а)) [5];

— двумерная текстура: простой и малозатратный подход, но такая статичная картинка имеет смысл только как дальнеплановые статичные изображения, через которые, например, нельзя пролететь сквозь. К тому же, такие облака не могут отбрасывать тени [5];

— объемные *(volumetric)*: динамические облака, с которыми можно взаимодействовать и которые способны отбрасывать тени (Рисунок 1.1 (б)) [4, 6]. Именно поэтому такие облака будут реализованы в данной работе.



а б

Рисунок 1.1 — (а) геометрические облака в компьютерной игре *Minecraft*; (б) объёмные облака в компьютерном авиасимуляторе *Microsoft Flight Simulator*

Исходя из требований к алгоритму, выдвигаемых в современной игровой индустрии [2,4,5], условия, которые будут использованы в данной работе:

— облака должны быть объемные;

— облака должны генерироваться процедурно;

— алгоритм должен быть быстродействующим.

1.3.1 Жидкостная симуляция

Использование жидкостной симуляции для создания объемных облаков: создать простые объекты (сферы, шары), вокселизировать их и рассматривать их как жидкость, получая похожие на объемные облака фигуры [4]. Современные физические модели облаков, основаны на решении *уравнений*

Навье-Стокса [2], что влечет за самой следующие недостатки [2]:

— алгоритм медленный;

— сложность контроля генерации;

— сложность реализации.

Из-за этих недостатков не используется в индустрии компьютерной графики, например в игровой [5].

1.3.2 Воксельная генерация

Алгоритм заключается в генерации ограничивающего параллелепипеда (bounding box), состоящего из вокселей, хранящих информацию о цвете [4]. Преимущества:

— хорошо сочетается с алгоритмом построением теней

Недостатки:

— высокие затраты памяти;

— сложность обработки большого количества вокселей в реальном времени;

— необходимость оптимизаций для обработки больших объемов.

Из-за этих недостатков так же не используется в индустрии компьютерной графики, например в игровой [5].

1.3.3 Визуализация на основе трассировки лучей

В алгоритме также используется ограничивающий параллелепипед, у которого визуализируются пиксели, соответствующие его видимым граням. Вместо вычисления каждого вокселя, алгоритм из точки наблюдателя для пикселя, соответствующему видимой грани параллелепипеда высчитывает его итоговый цвет [2, 7].

Для получения итогового цвета используется трёхмерная текстура сгенерированного шума [4].

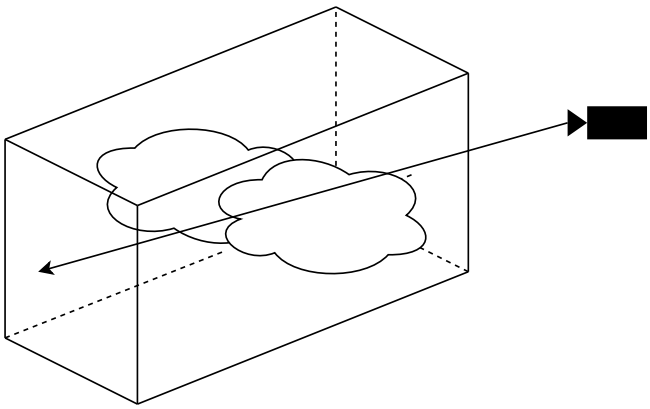


Рисунок 1.2 — Генерация на основе обратной трассировки лучей

Так для каждого луча вычисляется накопленная из шума плотность (Рисунок 1.3 демонстрирует трассу луча). Накопленная плотность преобразуется в интенсивность пикселя по закону Бугера 1.1.

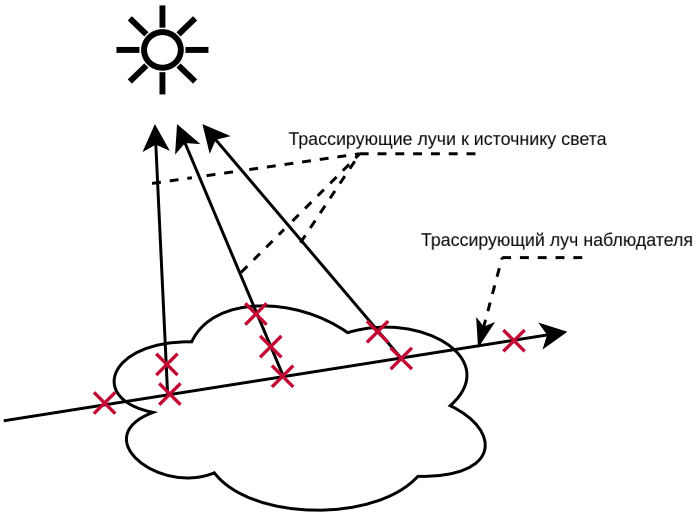


Рисунок 1.3 — Получение плотности облака лучом. Крестики – точки, в которых значения текстуры облака используются для вычисления плотности.

Визуализация на основе обратной трассировки лучей показывает преимущество перед воксельной и жидкостной генерациях, так как обрабатывает пиксели, соответствующие видимым граням ограничивающего параллелепипеда, что снижает вычислительные затраты и экономит память, что необходимо при формировании динамического изображения. Алгоритм рекомендован к использованию на практике [1, 2, 4].

1.4 Способы представления ландшафта

1.4.1 Аппроксимация примитивами

Для представления сгенерированного ландшафта используются примитивы, и наиболее распространенным и простым вариантом являются треугольники [8].

Также используется высотная карта – это двумерная текстура, где каждое значение соответствует высоте точки на поверхности.

Формируется сетка, состоящая из вершин, соединенных ребрами. Каждая вершина соответствует точке в высотной карте, а ребра образуют треугольники.

1.4.2 Аппроксимация неявными кривыми

Неявная кривая – это функция $F(x, y) = 0$, где равенство не выражает ни решение x от переменной y, ни наоборот. В частности, используются кривые, использующие тригонометрические функции. Тогда рельефом будет служить кривая, построенная по этой функции [9].

Подбор коэффициентов для получения такой кривой делает этот способ менее гибким и сложно контролируемым, чем использование предыдущего метода с использованием двумерной текстуры.

1.5 Шумы для процедурной генерации

Для моделирования объемных облаков используются трёхмерные текстуры, для создания форм, напоминающих облака [1]. Аналогичные текстуры используются для получения рельефа.

В работе рассматриваются два шума: Перлина и Ворли — Вороного. Существуют и другие шумы, однако использование конкретного шума для получения облаков и рельефа не является критичным и часто зависит только от визуальных и эстетических предпочтений. В данной работе для рельефа выбран шум Перлина, а для облаков – Ворли — Вороного.

1.5.1 Шум Перлина

Шум Перлина (Рисунок 1.5 (а)) – это пример градиентного шума [8]. Он не содержит полностью случайных значений в каждой точке, а представляет собой «волны». На рисунке 1.4 представлен пример такой «волны», чьи значения постепенно увеличиваются и уменьшаются. Ввиду этого, такой шум часто используется для создания рельефа.

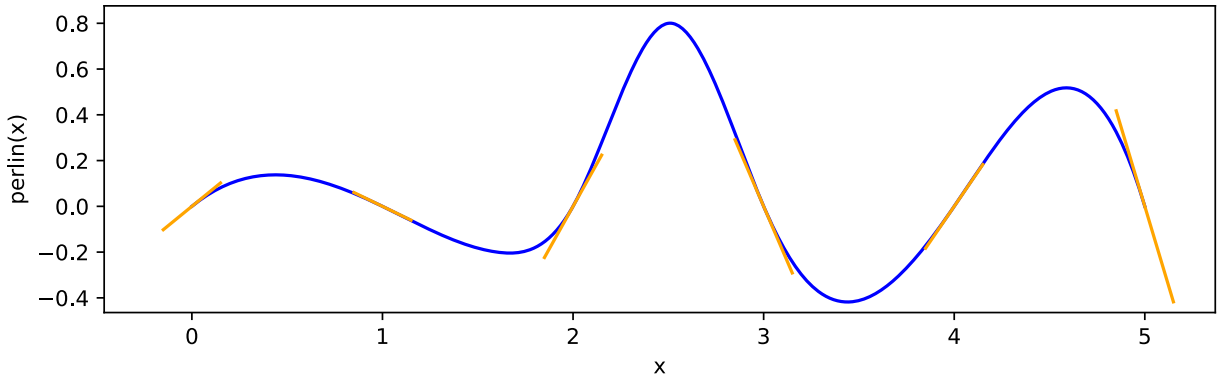


Рисунок 1.4 — Шум Перлина

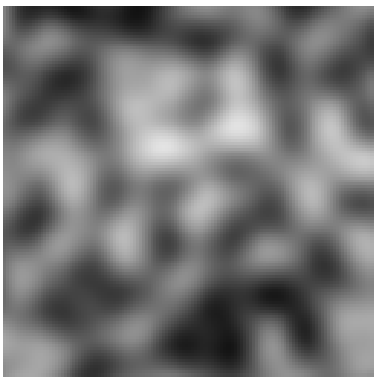
Для получения функции этого шума $p(x) : \mathbb{R}^n \to \mathbb{R}^n \in [0,1]^n$ необходим набор векторов из псевдослучайных чисел $g_u \in \mathbb{R}^n$ и значений $u \in \mathbb{Z}^n$ и, учитывая, что $p'(u) = g_u$ и $p(u) = 0$, остальные значения получаются путём интерполяции.

1.5.2 Шум Ворли — Вороного

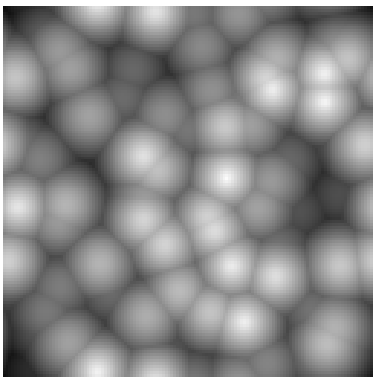
Шум Ворли — Вороного (Рисунок 1.5 (б)) позволяет получить круглые, пузыреподобные формы, напоминающие облачные структуры [4, 10].

Для получения функции этого шума $w(x) : \mathbb{R}^n \to \mathbb{R}^n \in [0,1]^n$: необходим набор из m векторов псевдослучайных чисел $g_i \in \mathbb{R}^n, i \in 1, 2, \ldots, m$.

Тогда $w(x) = \min\limits_{i \in 1,2,\ldots,m} (\rho(x, g_i))$, где $\rho(x, y)$ – расстояние между x и y. Далее значения $w(x)$ нормализуются по максимальному значению.



а



б

Рисунок 1.5 — (а) шум Перлина; (б) шум Ворли

1.6 Модель освещения

1.6.1 Модель Фонга

Интенсивность зеркального отраженного света (рисунок 1.6) зависит от угла падения, длины волны падающего света и свойств вещества [11]. Зеркальное отражение – направленное. Угол отражения от идеально отражающей поверхности равен углу падения. Это значит, что угол наблюдения совпадает с углом отражения. При неидеальных поверхностях, количество света достигшего наблюдения зависит от пространственного распределения отраженного света. У гладких поверхностей распределенное узкое, у шероховатых – более широкое. Благодаря зеркальному отражению появляются световые блики [11].

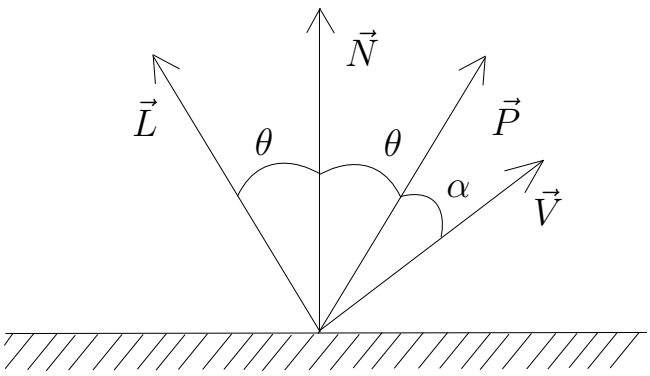


Рисунок 1.6 — Зеркальное отражение: $\vec{V}$ — вектор наблюдения, $\vec{P}$ — отраженный луч, $\vec{L}$ — падающий луч и $\vec{N}$ нормаль к поверхности.

В моделе Фонга используют эмпирическую форму [11]:

$$I = I_l w(i, \lambda) \cos^n(\theta), \tag{1.3}$$

где I – интенсивность отраженного света, I_l – интенсивность точечного источника, w(i, λ) – кривая отражения, представляющее отношение зеркального отраженного света, к падающему как функцию угла падения i и длины волны λ, n – степень, аппроксимирующая пространственное распределение зеркального света.

1.6.2 Модель Ламберта

Диффузное отражения происходит, когда свет проникает под поверхность объекта, поглощается, а затем вновь испускается (Рисунок 1.7) [11].

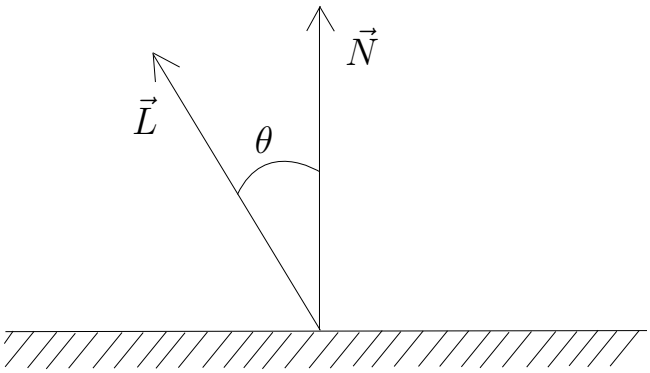


Рисунок 1.7 — Диффузное отражение: $\vec{L}$ — падающий луч и $\vec{N}$ нормаль к поверхности.

Не зависит от положения наблюдателя, так как рассеивается равномерно по всем направлениям [11]. Интенсивность отраженного света вычисляется по закону косинусов Ламберта:

$$I = I_l k_d \cos(\theta), \tag{1.4}$$

где I – интенсивность отраженного света, I_l – интенсивность точечного источника, k_d – коэффициент диффузного отражения, θ – угол между направлением света и нормалью к поверхности.

Поверхность с ламбертовым диффузным отражением выглядит мато-

вой и блеклой [11], что имеет смысл для изображения травяного рельефа. Так как рельеф – незеркальная поверхность, в работе используется модель Ламберта.

1.7 Учёт теней от облаков на поверхности

Тени от облаков зависят только от положения источника света и не зависят от положения наблюдателя [11]. Для объемных облаков при построении их теней аналогично используется обратная трассировка лучей: при движении луча от поверхности к облакам определяется суммарная плотность облаков по пути, чтобы вычислить, сколько света блокируется [6].

Таким образом, учитывая закон Бугера–Ламберта–Бера 1.1 и закон Ламберта 1.4, итоговая формула для расчета интенсивности света на поверхности:

$$I_s = I_0 I_l k_d \cos(\theta) e^{-k_\lambda l}. \tag{1.5}$$

1.8 Удаление невидимых линий и поверхностей

1.8.1 Алгоритм Робертса

Алгоритм Робертса — метод работающий в объектном пространстве. Алгоритм прежде всего удаляет из каждого тела те ребра или грани, которые экранируются самим телом. Затем каждое из видимых ребер каждого тела сравнивается с каждым из оставшихся тел для определения того, какая его часть или части, если таковые есть, экранируются этими телами [11].

1.8.2 Алгоритм Варнока

Алгоритм работает в пространстве изображения [11]. Алгоритм разбивает окно, чтобы определить, является ли его содержимое достаточно простым для отрисовки, если нет – рекурсивно разбивает его дальше [11].

1.8.3 Алгоритм Вейлера-Азертона

Оптимизация алгоритма Варнока в отношении числа выполняемых разбиений за счёт перехода от прямоугольных разбиений к разбиениям вдоль границ многоугольников. Работает в пространстве изображений [11].

1.8.4 Алгоритм художника

Используется для простых элементов сцены, например для многоугольников [11]. Алгоритм аналогичен тому способу, которым художник создает картину. Сначала художник рисует фон, затем предметы, лежащие на среднем расстоянии, и, наконец, передний план. Тем самым художник решает задачу об удалении невидимых поверхностей, или задачу видимости, путем построения картины в порядке обратного приоритета [11].

1.8.5 Метод обратной трассировки лучей

В этом алгоритме предполагается, что сцена уже преобразована в пространство изображения [11].

Алгоритм работает за счет отслеживания (трассировки) лучей от наблюдателя к объекту. Трассировка прекращается, как только луч пересекает поверхность видимого объекта.

1.8.6 Алгоритм, использующий Z-буфер

Z-буфер – хранит значения глубины z для каждого пикселя [11].

Этапы работы алгоритма

1) Инициализация Z-буфера минимально возможным значением z.
2) Преобразование каждого многоугольника сцены в растровую форму.
3) Для каждого пикселя (x, y), принадлежащего многоугольнику, вычисляется его глубина $Z(x, y)$.
4) Выполняется сравнение глубины $Z(x, y)$ с текущим значением глубины $Z_{\text{буф}}(x, y)$, хранящимся в Z-буфере. Если $Z(x, y) > Z_{\text{буф}}(x, y)$, то в буфер кадра записывается атрибут пикселя (цвет) и значение $Z_{\text{буф}}(x, y)$ заменяется на $Z(x, y)$.

Так как для визуализации облаков используется алгоритм обратной трассировки лучей, работающий непосредственно с пикселями, а не в объектном пространстве, следует выбрать алгоритм, работающий в пространстве изображений.

Ввиду простоты реализации из рассмотренных был выбран алгоритм,

использующий Z-буфер.

1.9 Методы закраски

1.9.1 Простая закраска

Вся грань закрашивается единственным цветом, интенсивность которого рассчитывается по закону Ламберта 1.4. Простая закраска используется при выполнении трех условий [12]:

— источник находится в бесконечности: так как падающие лучи параллельны друг другу – это повлияет на расчёт диффузной составляющей, так как она зависит от угла падения, однако для всех точек угол падения одинаков, значит диффузная составляющая одинакова;

— закрашиваемая грань является реально существующей, а не полученной в результате аппроксимации поверхности;

Так как рельеф, который закрашивается в работе будет получен с помощью аппроксимации, простая модель не будет использоваться в этой работе.

1.9.2 Закраска методом Гуро

При такой закраске определяется интенсивность в вершинах на основе нормали к вершине как среднее нормалей граней, образующих эту вершину, а затем интерполяцией вычисляется интенсивность каждого пикселя многоугольника [11].

1.9.3 Закраска методом Фонга

При такой закраске определяются нормали в вершинах многоугольника, а затем интерполяцией вычисляется нормаль каждого пикселя многоугольника, а затем его интенсивность. При таком методе достигается лучшая аппроксимация поверхности [11]. Именно по этому критерию этот метод выбран для закраски ландшафта.

Вывод

В аналитической части формализованы задачи и объекты сцены. Также проанализированы и описаны алгоритмы, используемые для визуализа-

ции ландшафтной сцены с облаками. Был выбран алгоритм использующий обратную трассировку лучей для визуализации объемных облаков, а также метод представления ландшафта с помощью аппроксимацией примитивами. Для процедурной генерации ландшафта выбран шум Перлина, для облаков – Ворли — Вороного.

Описана оптическая модель облаков и выбраны алгоритмы решения основных задач компьютерной графики. Для удаления невидимых линий и поверхностей выбран алгоритм использующий Z-буфер. Для учёта теней и освещения рассмотрены и выбраны закон ламбертового диффузного отражения и закраска методом Фонга.

2 Конструкторский раздел

В конструкторском разделе спроектировано разрабатываемое программное обеспечение и формально описаны используемые алгоритмы.

2.1 Функциональная диаграмма

На рисунке 2.1 представлена IDEF0-диаграмма визуализации сцены уровня A0.

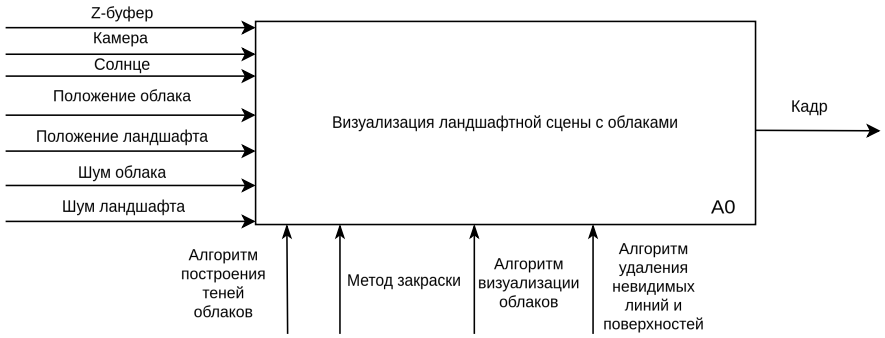


Рисунок 2.1 — IDEF0-диаграмма функциональной декомпозиции уровня A0

На рисунке 2.2 представлена IDEF0-диаграмма визуализации сцены уровня A1.

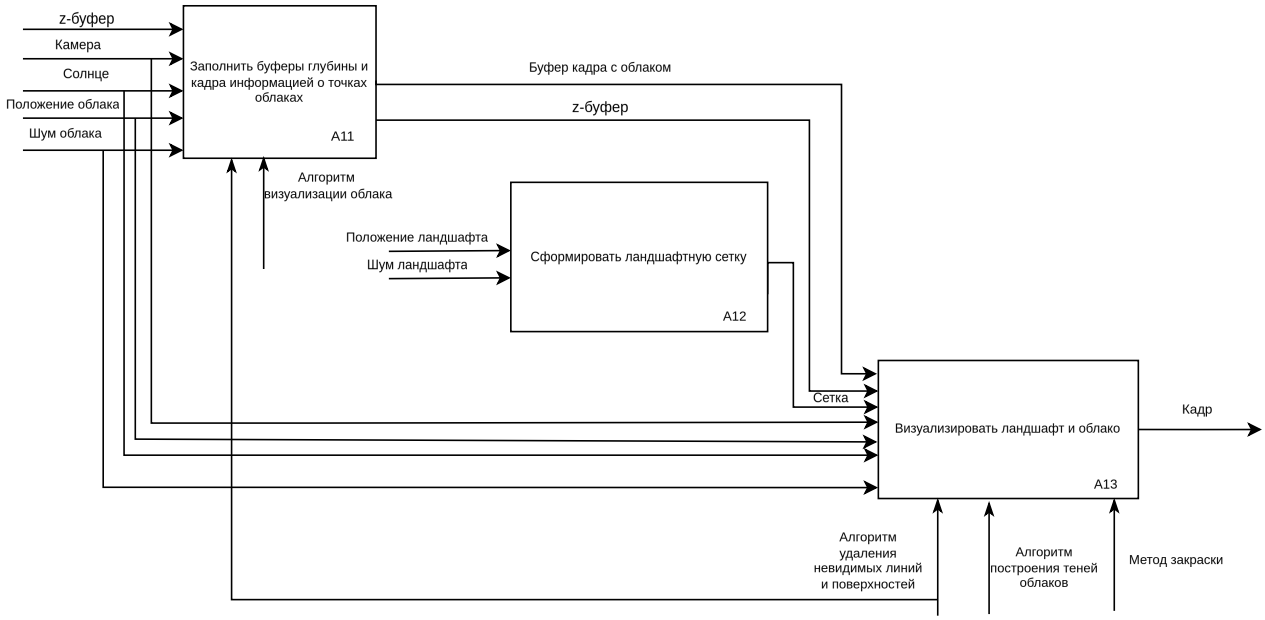


Рисунок 2.2 — IDEF0-диаграмма функциональной декомпозиции уровня A1

2.2 Схемы алгоритмов

На рисунке 2.3 представлена схема алгоритма отрисовки облака.

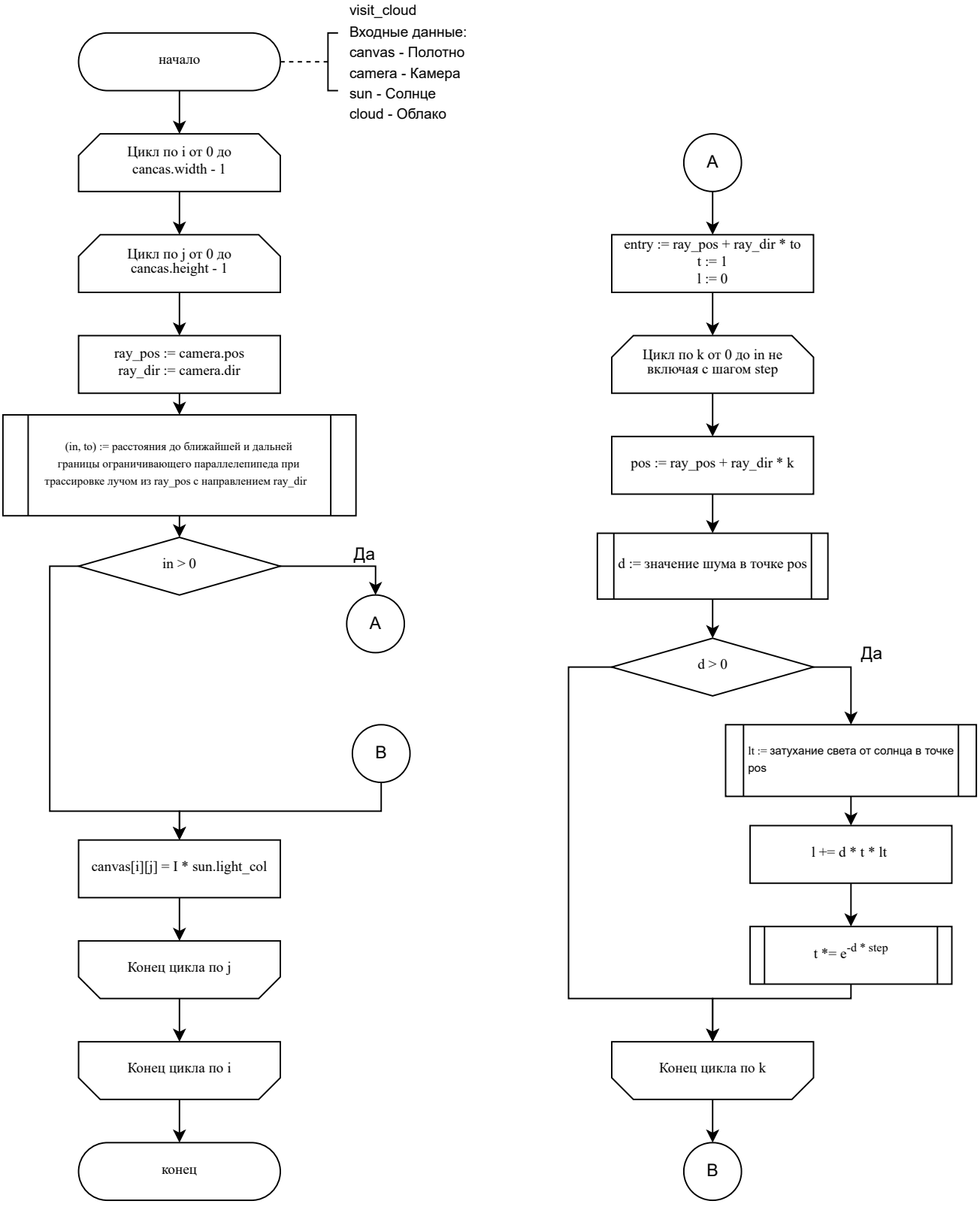


Рисунок 2.3 — Схема алгоритма отрисовки облака

На рисунке 2.4 представлена схема алгоритма определения затухания света от солнца в точке облака.

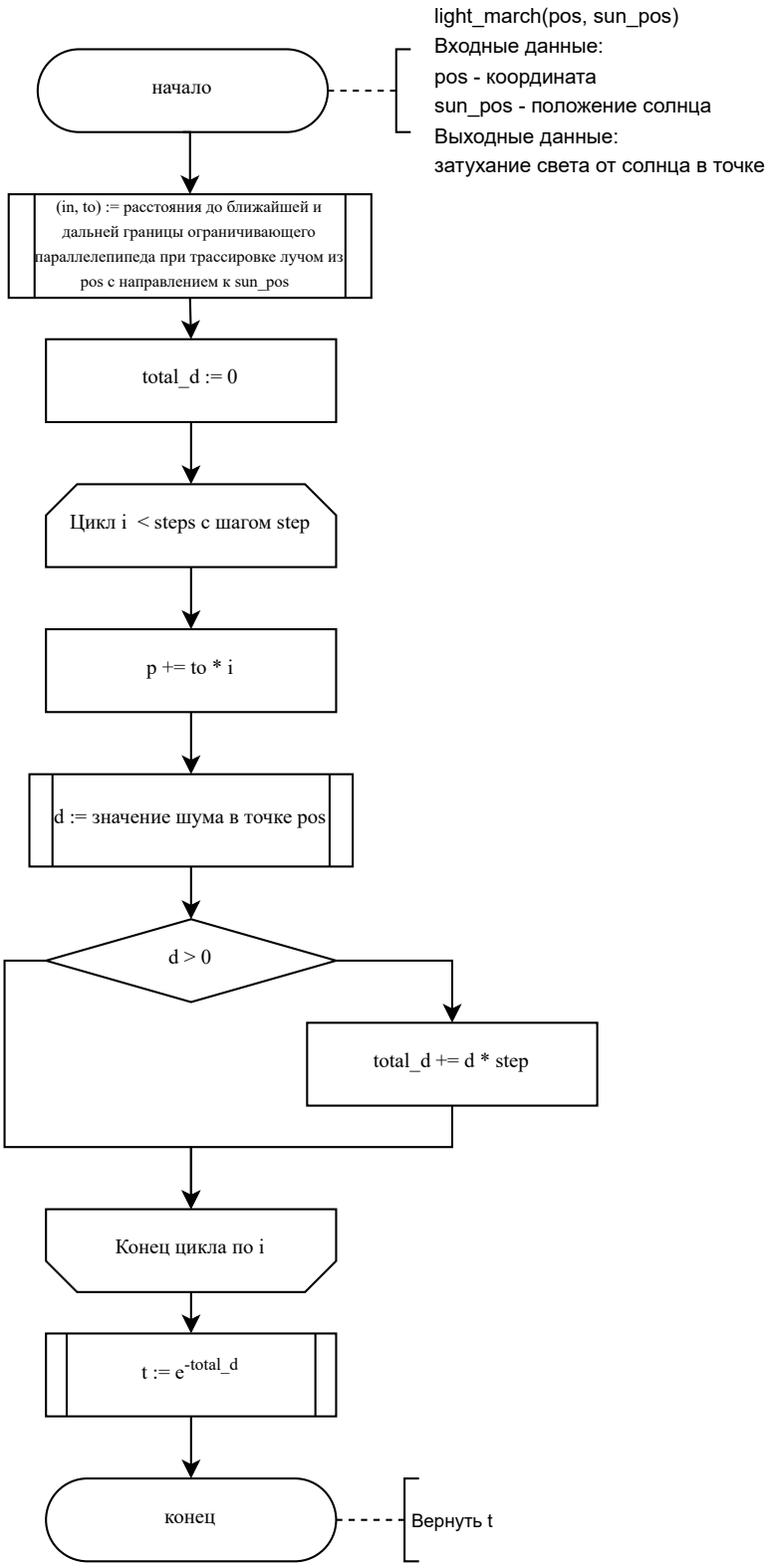


Рисунок 2.4 — Схема алгоритма определения затухания света от солнца в точке облака

На рисунке 2.5 представлена схема алгоритма генерации ландшафтной сетки из треугольников.

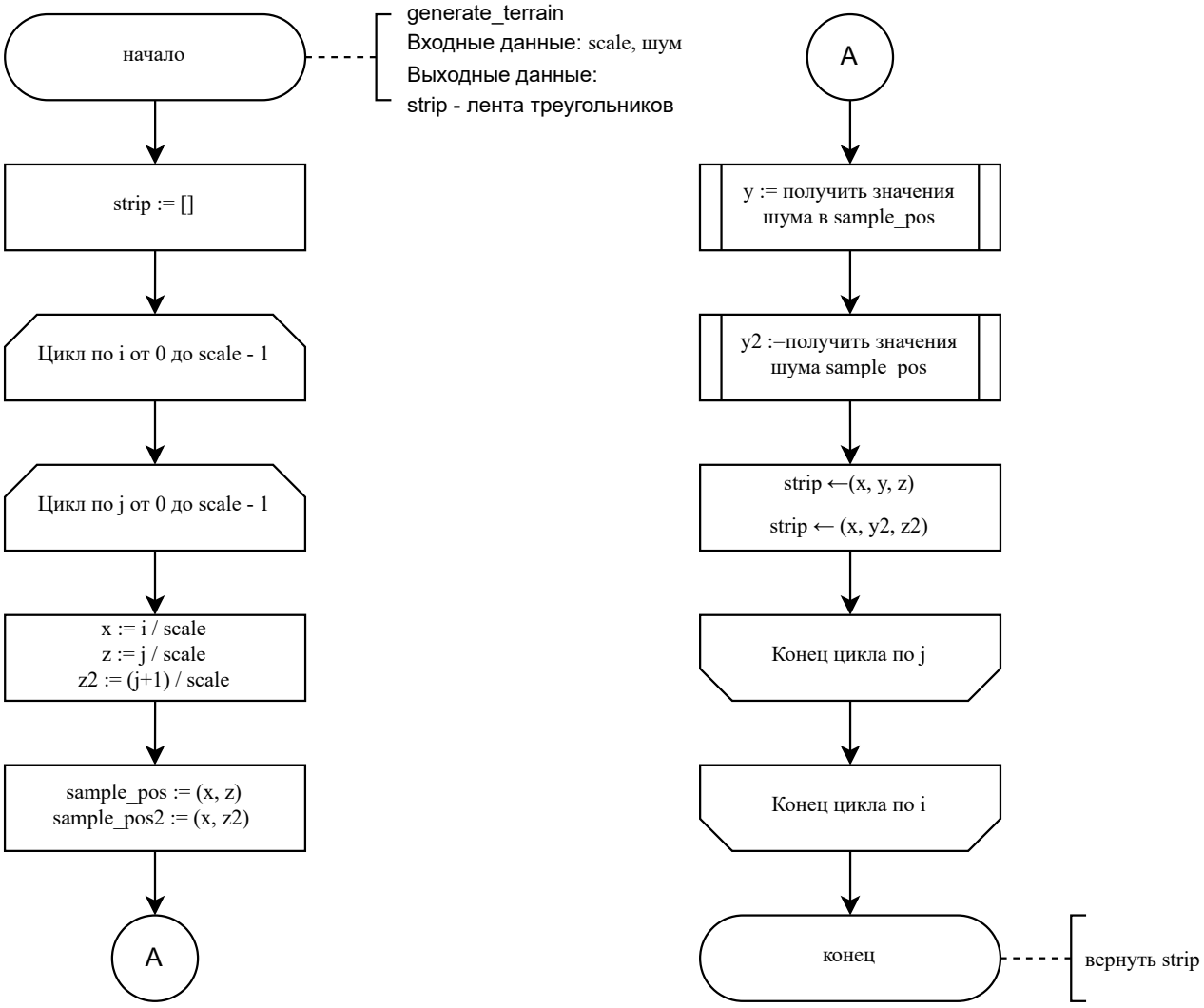


Рисунок 2.5 — Схема алгоритма генерации ландшафтной сетки.

На рисунке 2.6 представлена схема алгоритма отрисовки ландшафта.

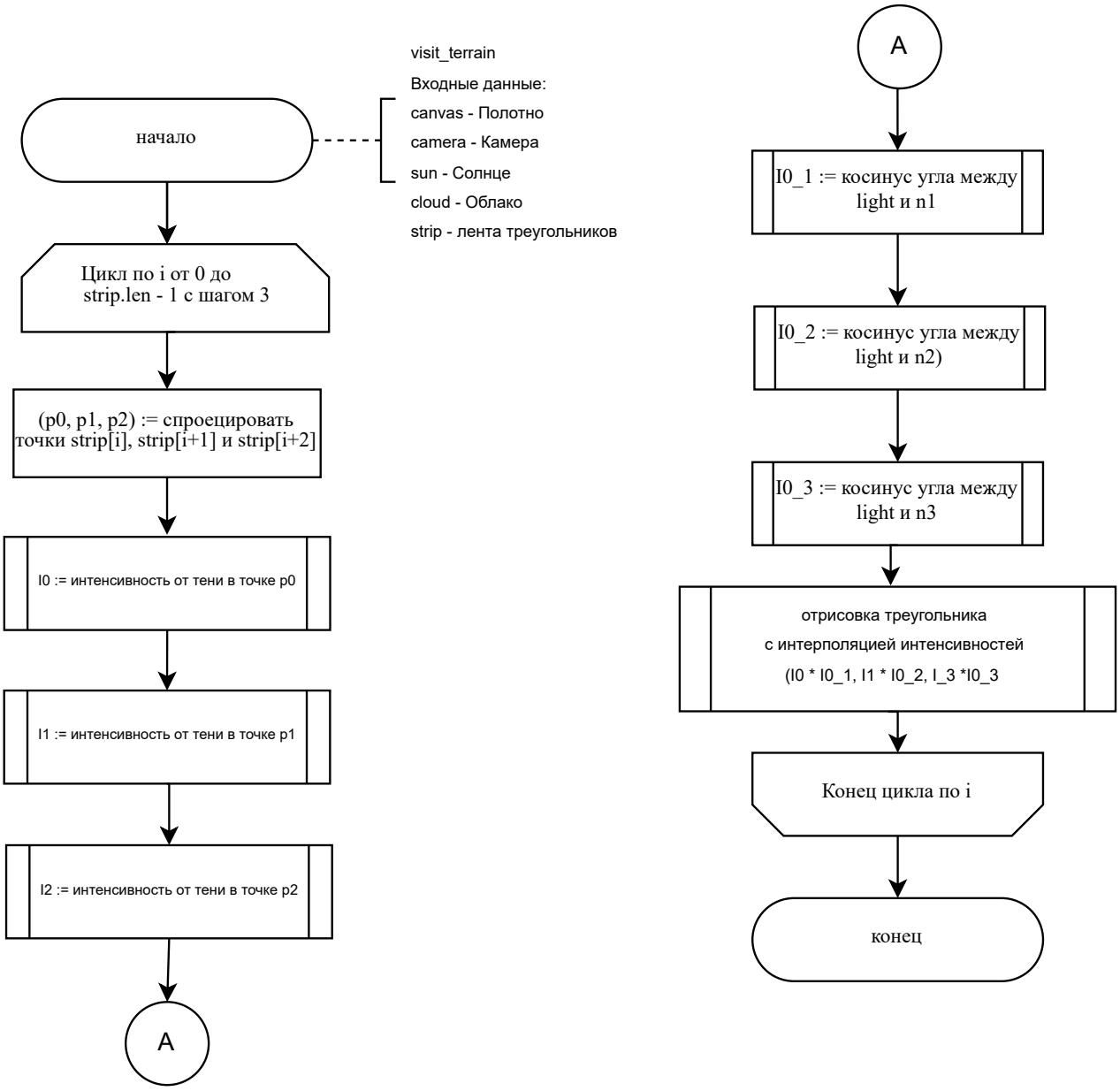


Рисунок 2.6 — Схема алгоритма отрисовки ландшафта.

2.3 Диаграмма классов

На рисунке 2.7 представлена UML-диаграмма классов программного обеспечения.

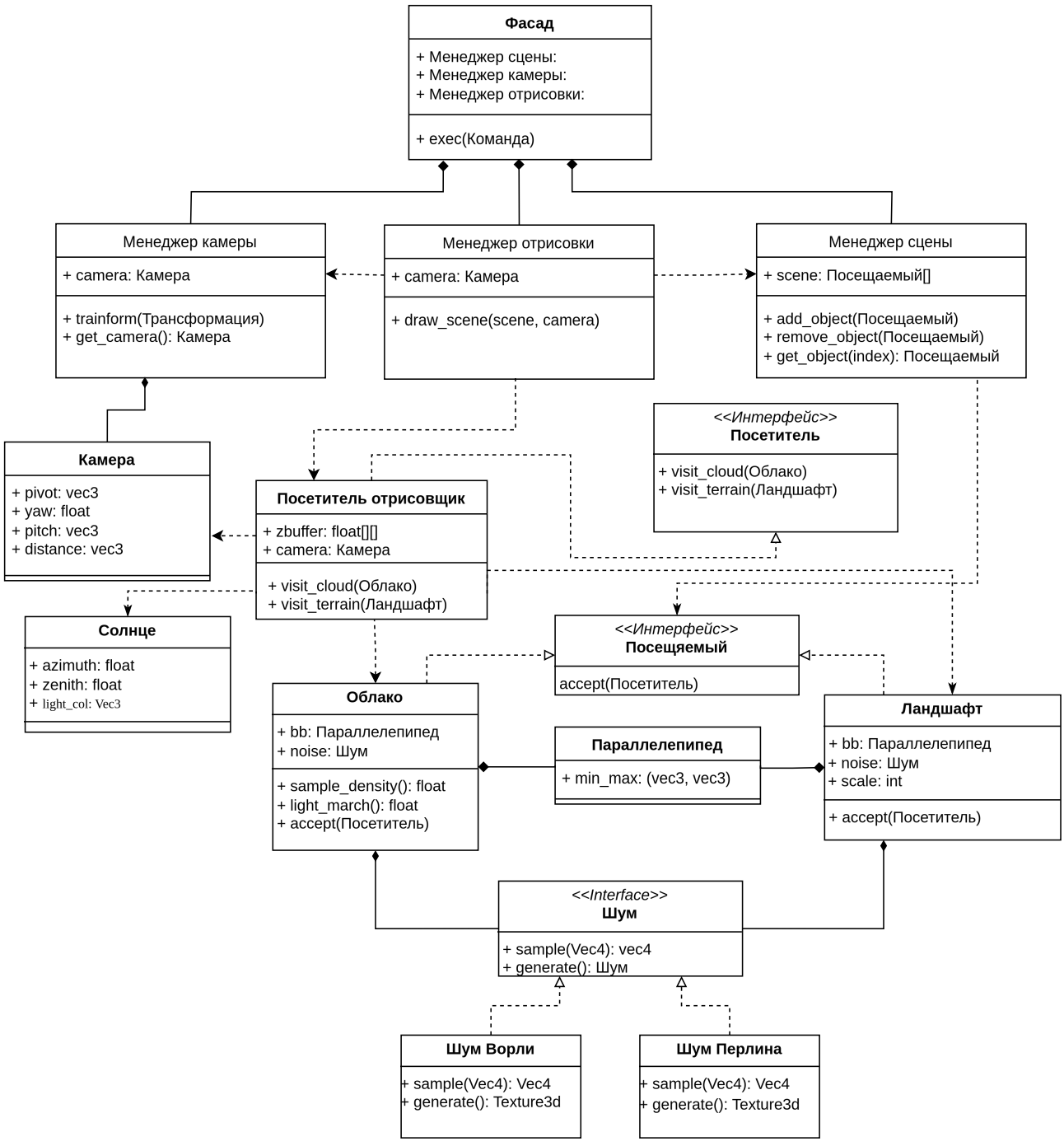


Рисунок 2.7 — UML-диаграмма

Операции производятся командами через фасад, который управляет элементами сцены через менеджеров – это позволит программе не зависеть от интерфейса.

Менеджер сцены управляет её элементами: облако, ландшафт и солнце.

Менеджер камеры управляет камерой, для которой можно задавать точку наблюдения *pivot* и удаление от неё *distance*, рыскание *yaw* и тангаж *pitch*.

Менеджер отрисовки обращается к менеджерам сцены и камерой и передает камеру и элементы сцены в посетителя–отрисовщика для отрисовки последних.

Облака и ландшафт используют шум Перлина или Ворли—Вороного. Для получения значения шума в определённой точке облака используется метод *sample_density*. Облако и ландшафт так же хранят ограничивающий параллелепипед, задаваемый двумя точками *min_max* в пространстве.

Вывод

В разделе спроектировано разрабатываемое программное обеспечение и описана декомпозиция разрабатываемого ПО с формальным описанием используемых алгоритмов.

3 Технологический раздел

В технологическом разделе выбраны и описаны средства реализации программного обеспечения и представлены детали его реализации.

Для реализации программного обеспечения был выбран язык программирования *Rust* [13], так как он позволяет реализовать все алгоритмы, выбранные в результате проектирования и поддерживает все требуемые структуры данных.

Был выбран фреймворк *egui* [14] для реализации интерфейса программного обеспечения, так как в нём присутствуют инструменты для работы с изображениями и разработки интерфейса.

3.1 Реализация алгоритмов

В листинге 3.1 представлен листинг функции отрисовки облака.

Листинг 3.1 — Фунция отрисовки облака

```
fn visit_cloud(&mut self, cloud: &Cloud) {
    use rayon::prelude::*;

    let bb = cloud.bounding_box();

    let (w, h) = self.canvas.size();
    let light_color: Vec4 = cloud.light_color.into();
    let sun_pos = sun.map(|x| x.get_pos()).unwrap_or_default();
    let ray_origin = self.camera.pos();
    img.pixels.par_iter_mut().enumerate().for_each(|(idx, pixel)|
       {
        let i = idx / w + min_tuple.y as usize;
        let j = idx % w + min_tuple.x as usize;

        let ray_dir = (self.camera.egui_to_world(i, j, w, h) -
           ray_origin).normalize();
        let ray_box_info = bb.dst(ray_origin, ray_dir);
        let dst_to_box = ray_box_info.x;
        let dst_inside_box = ray_box_info.y;

        if dst_inside_box <= 0.0 {
            *pixel = Color32::TRANSPARENT
        } else {
            let mut dst_travelled = 0.0;
            let dst_limit = dst_inside_box;
            let step_size = dst_inside_box / cloud.num_steps as f32
               ;
            let mut transmittance = 1.0;
            let mut light_energy = 0.0;
```

```
            let entry_point = ray_origin + dst_to_box * ray_dir;
            let cos_angle = ray_dir.dot(sun_pos);
            let phase = phase(cos_angle.abs(), cloud.phase_params);

            while dst_travelled < dst_limit {
                let ray_pos = entry_point + ray_dir * dst_travelled
                    ;
                let density = cloud.sample_density(ray_pos);
                if density > 0.1 {
                    let light_transmittance = cloud.light_march(
                        ray_pos, sun_pos);
                    light_energy +=
                    density * step_size * transmittance *
                        light_transmittance * phase;
                    transmittance *=
                    beer(density * step_size * cloud.
                        light_absorption_through_cloud);
                    if transmittance < 0.01 {
                        break;
                    }
                }
                dst_travelled += step_size;
            }

            *pixel = if transmittance >= 0.01 {
                Color32::TRANSPARENT
            } else {
                let focused_eye_cos = cos_angle
                .clamp(-std::f32::consts::PI, std::f32::consts::PI)
                .powf(cloud.params.x);
                let sun = hg(focused_eye_cos, 0.995).clamp(-1.0,
                    1.0) * transmittance;

                let cloud_col = light_energy * light_color.xyz();
                let col = cloud_col.clamp(Vec3::ZERO, Vec3::ONE) *
                    (1.0 - sun)
                + light_color.xyz() * sun;
                let (r, g, b) = col.into();
                Color32::from_rgb(
                (r * 255.0) as u8,
                (g * 255.0) as u8,
                (b * 255.0) as u8)
            };
        }
    });
}
```

В листинге 3.2 представлен листинг функции определения затухания света от солнца в точке облака.

Листинг 3.2 — Определение затухания света от солнца в точке облака.

```
pub fn light_march(&self, mut p: Vec3, world_space_light_pos0:
   Vec3) -> f32 {
    let dir_to_light = world_space_light_pos0;
    let dst_inside_box = self.bounding_box().dst(p, dir_to_light).
       y;
    let step_size = dst_inside_box / self.num_steps_light as f32;
    p += dir_to_light * step_size;

    let mut total_density = 0.0;
    let step_size_f32 = step_size;

    for _ in 0..self.num_steps_light {
        let density = self.sample_density(p);
        total_density += density.max(0.0) * step_size_f32;
        p += dir_to_light * step_size_f32;
    }

    let transmittance = beer(total_density * self.
       light_absorption_toward_sun);
    transmittance.lerp(1.0, self.darkness_threshold)
}
```

В листинге 3.3 представлен листинг функции генерации ландшафтной сетки.

Листинг 3.3 — Функция генерации ландшафтной сетки.

```
    pub fn generate_grid(&mut self) {
        use rayon::prelude::*;
        let bb = self.bounding_box;
        let min = bb.min;
        let max = bb.max;
        let scale = self.scale;

        let sample_y = self.noise_weight.x;
        let sample_z = self.noise_weight.y;
        let noise = &self.perlin;

        let vec: Vec<_> = (0..self.scale)
        .into_par_iter()
        .flat_map(|z| {
            (0..=self.scale)
            .into_par_iter()
            .map(move |x| {
                let x_frac = x as f32 / scale as f32;
                let z_frac = z as f32 / scale as f32;
                let next_z_frac = (z + 1) as f32 / scale as f32;
```

```
                let base_x = min.x + (max.x - min.x) * x_frac;
                let base_z = min.z + (max.z - min.z) * z_frac;
                let next_z = min.z + (max.z - min.z) * next_z_frac;

                let sample_pos = bb.min
                + Vec3::new(base_x, sample_y, base_z)
                / Vec3::new(bb.size().x, 1.0, bb.size().z);
                let sample_pos2 = bb.min
                + Vec3::new(base_x, sample_y, next_z)
                / Vec3::new(bb.size().x, 1.0, bb.size().z);

                let worley_height =
                bb.min.y + noise.sample_level(sample_pos).x * (bb.
                   max.y - bb.min.y);

                let worley_height2 =
                bb.min.y + noise.sample_level(sample_pos2).x * (bb.
                   max.y - bb.min.y);

                let vec = Vec3::new(base_x, worley_height, base_z);
                let vec2 = Vec3::new(base_x, worley_height2, next_z
                   );

                vec![vec, vec2]
            })
            .flatten()
        })
        .collect();
    }
```

В листинге 3.4 представлен листинг функции отрисовки ландшафта.

Листинг 3.4 — Функция отрисовки ландшафта

```
fn visit_terrain(&mut self, terrain: &Terrain) {
    use rayon::prelude::*;
    let (width, height) = (1056.0, 900.0);
    let bb = terrain.bounding_box;

    terrain.triangles.par_iter().for_each(|(v, (n0, n1, n2))| {
        let center = v.center();
        let light = (sun_pos - center).normalize();

        let (v0, v1, v2) = v.to_tuple();
        let (p0, p1, p2) = (v0, v1, v2);

        let v0 = self.canvas.transform(v0, self.mvp);
        let v1 = self.canvas.transform(v1, self.mvp);
        let v2 = self.canvas.transform(v2, self.mvp);
        if let (Some(v0), Some(v1), Some(v2)) = (v0, v1, v2) {
            let min_x = v0.x.min(v1.x).min(v2.x) as usize;
            let max_x = v0.x.max(v1.x).max(v2.x) as usize;
            let min_y = v0.y.min(v1.y).min(v2.y) as usize;
```

```
            let max_y = v0.y.max(v1.y).max(v2.y) as usize;

            for y in min_y..=max_y {
                for x in min_x..=max_x {
                    if inside_triangle(Pos2::new(x as f32, y as f32)
                       , v0, v1, v2) && x < width as usize && y <
                       height as usize {
                        let x1 = cloud.light_march(p0, light);
                        let x2 = cloud.light_march(p1, light);
                        let x3 = cloud.light_march(p2, light);

                        let alpha1 = (-(p0 - sun_pos).normalize()).
                           dot(n0.normalize());
                        let alpha2 = (-(p1 - sun_pos).normalize()).
                           dot(n1.normalize());
                        let alpha3 = (-(p2 - sun_pos).normalize()).
                           dot(n2.normalize());

                        let beta = interpolate(Pos2::new(x as f32, y
                            as f32), v0, v1, v2, x1 * alpha1, x2 *
                           alpha2, x3 * alpha3);

                        let dif = terrain.diffuse * beta;
                        let col = terrain.color * dif;
                        let (r, g, b) = (col.x * 255.0, col.y *
                           255.0, col.z * 255.0);
                        img[(y, x)] = Color32::from_rgb(r as u8, g
                           as u8, b as u8);
                    }
                }
            }
        }
    });
}
```

3.2 Модульное тестирование

Было проведено модульное тестирование программного обеспечения с помощью менеджера пакетов *Cargo* [15] языка *Rust* командой *cargo test* [16].

Пример модульного теста представлен в листинге 3.5.

Листинг 3.5 — Модульный тест проверки изменения компонента композита

```
    #[test]
    fn test_get_mut_object() {
        let mut scene = SceneObjects::default();
        let component = Component::Sun(Sun::new(0.0, 0.0));
        scene.add_object("object1", component.clone());

        if let Some(retrieved) = scene.get_mut_object("object1") {
            if let Component::Sun(sun) = retrieved {
                sun.prepend_angle(10.0)
            }
        }

        let modified_component = Component::Sun(Sun::new(0.0,
           10.0));
        assert_eq!(scene.get_object("object1"), Some(&
           modified_component));
    }
```

Была проведена оценка покрытия программы модульными тестами с помощью утилиты *Cargo Tarpaulin* [17]. В качестве метрики успешности покрытия данная утилита использует формулу 3.1.

$$\frac{N}{M} * 100\%, \tag{3.1}$$

где N – покрытые тестами строки кода и M – общее количество строк в программе.

Тестировались критически важные функции. Итоговое покрытие программы составило 19.83%.

3.3 Интерфейс

Поворот камеры осуществляется движением курсора с зажатием левой кнопки мыши, приближение – среднее колесо, перемещение движением курсора с зажатием правой кнопки мыши. На рисунке 3.1 представлен результат визуализации динамической сцены с облаками.

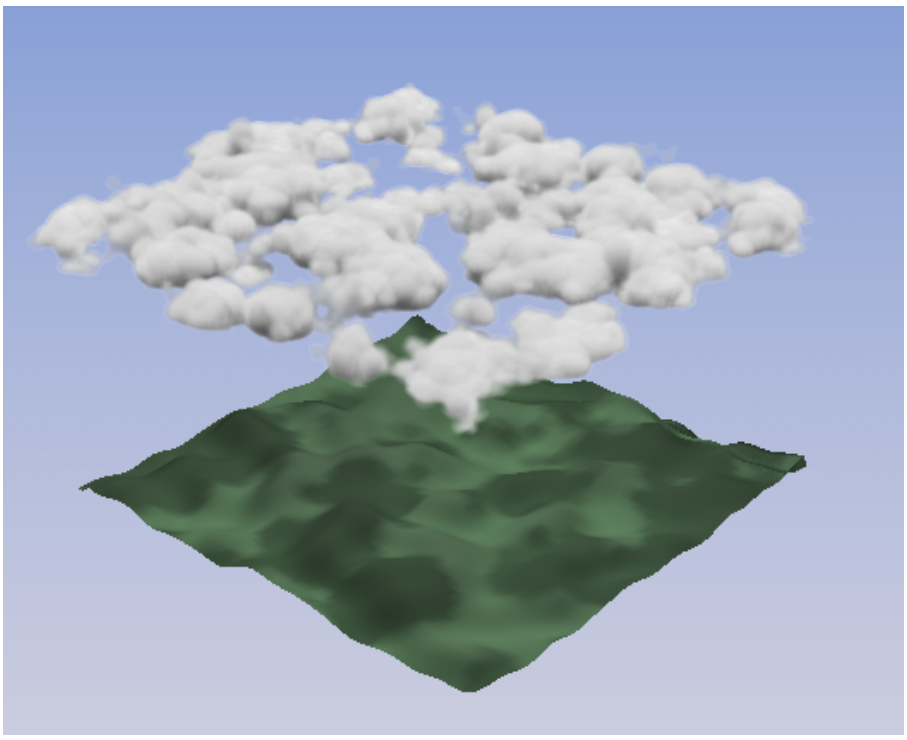


Рисунок 3.1 — Визуализация динамической сцены с облаками.

На рисунке 3.2 представлен результат визуализации динамической сцены с облаками с другого ракурса и с другим положением солнца.

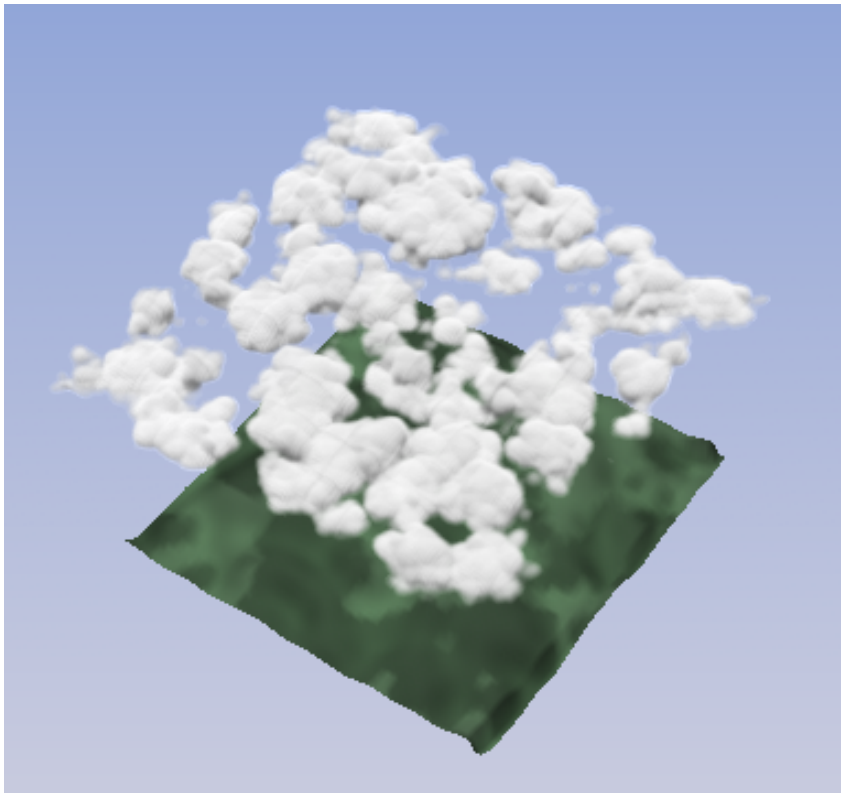


Рисунок 3.2 — Визуализация динамической сцены с облаками с другого ракурса.

На рисунке 3.3 представлен интерфейс управления азимутальным углом солнца.

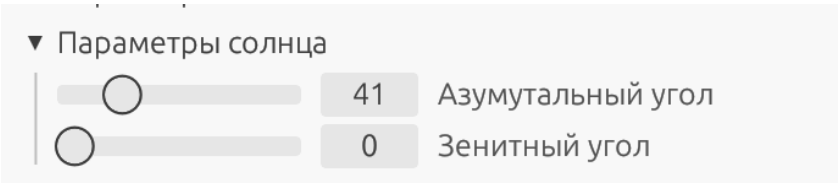


Рисунок 3.3 — Интерфейс управления азимутальным углом солнца.

На рисунке 3.4 представлен интерфейс управления параметрами облаков.

Рисунок 3.4 — Интерфейс управления параметрами облаков.

На рисунке 3.5 представлен интерфейс управления параметрами ландшафта.

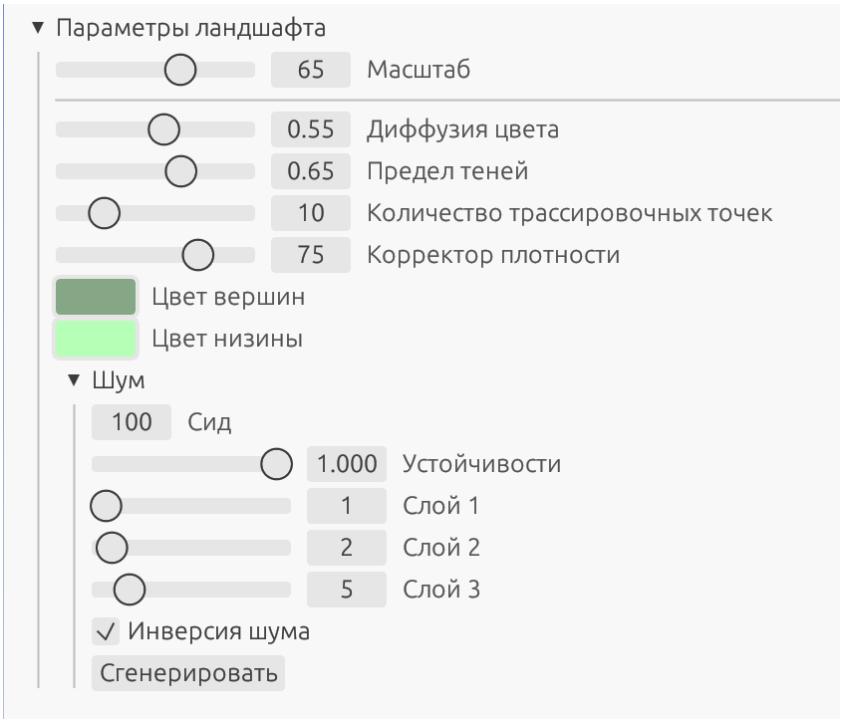


Рисунок 3.5 — Интерфейс управления параметрами ландшафта.

Вывод

В данном разделе был обоснован выбор средств реализации программного обеспечения и представлены детали реализации: реализация алгоритмов и интерфейс программы. Проведено модульное тестирование программы – тесты пройдены успешно. Также собрана информация о покрытии программы модульными тестами – 19.83%.

4 Исследовательский раздел

В исследовательском разделе проведено исследование разработанного программного обеспечения.

Технические характеристики устройства, на котором было проведено исследование [18]

— Intel® Core™ Ultra 5 Processor 4.50 ГГц 14 логических ядер [19];

— Оперативная память: 16 Гб;

4.1 Цель исследования

Ключевым для алгоритма визуализации облаков является количество точек, которые берутся на трассирующем луче от наблюдателя для вычисления плотности (рисунок 1.3). Чем больше точек используется, тем лучше визуальный результат, но тем дольше происходит отрисовка облака.

Цель исследования — найти оптимальное количество трассирующих лучей. Для этого проведено сравнение алгоритма визуализации облаков с распараллеливанием цикла обхода пикселей и без него. Распараллеливание было выполнено с помощью библиотеки *rayon* [20] на всех ядрах устройства.

Временные параметры были сопоставлены с оценкой качества изображения, полученного при визуализации облаков, через опрос респондентов.

4.2 Исследование

Исследование проводилось на ноутбуке, подключённом к сети электропитания, с минимальной нагрузкой (только необходимые программы: терминал и программа для запуска исследования). Использовалась компиляция без оптимизаций.

Для каждого шага проводилось 300 прогонов. Рассчитывалось общее время, из которого вычислялось среднее значение. Сцена была отрисована с тем же ракурсом, что и на рисунке 3.1, но без ландшафта и неба. Этот ракурс использовался и в оцениваемых изображениях опроса.

На рисунке 4.1 представлен график зависимости времени от количества точек на трассируемом луче. Зависимость в табличной форме показана в таблице 4.1.

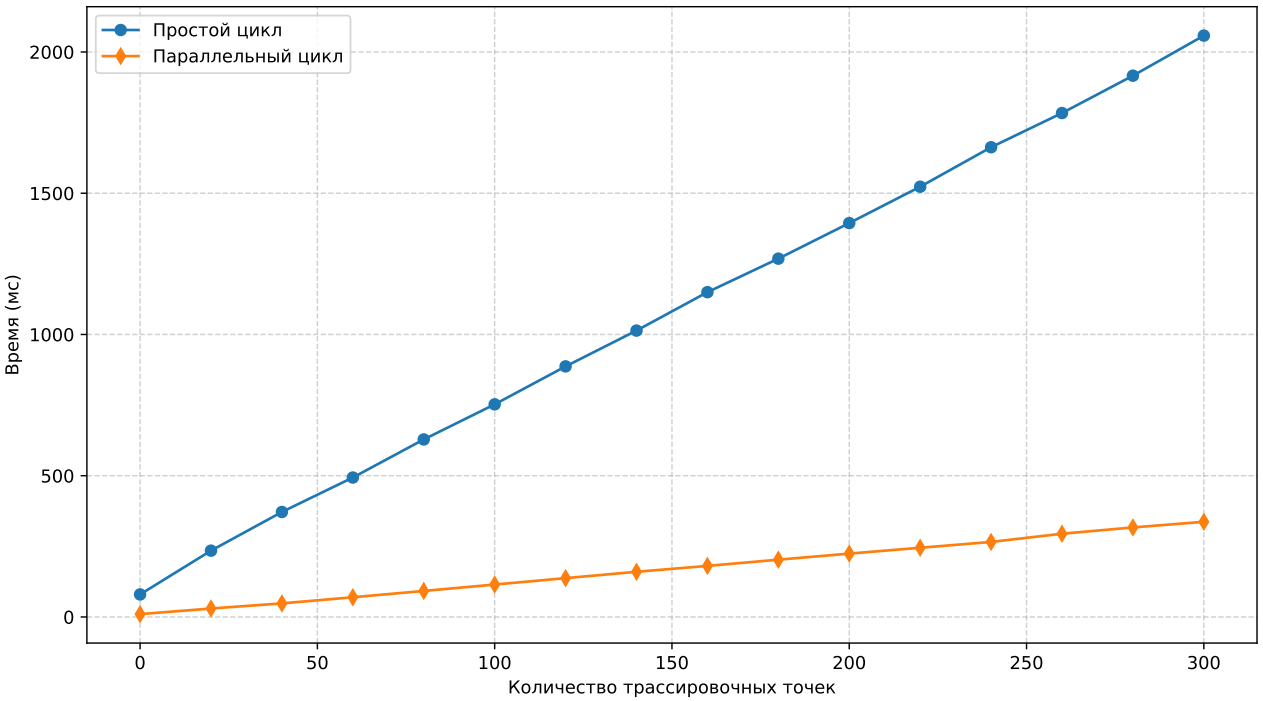


Рисунок 4.1 — График зависимости времени выполнения от количества точек на трассируемом луче.

Таблица 4.1 — Зависимость времени выполнения от количества точек на трассируемом луче

Количество точек	**Параллельный цикл (мс)**	**Простой цикл (мс)**
0	10	79
20	29	234
40	48	371
60	69	493
80	92	628
100	114	752
120	137	887
140	159	1013
160	180	1149
180	202	1268
200	224	1394
220	244	1522
240	265	1662
260	294	1783
280	316	1916
300	336	2057

Респондентам было преложено оценить реалистичность облачного пейзажа 10-балльной шкале. (1 - нереалистичное, 10 - реалистичное). В опросе приняли участие 20 человек. Сбор данных происходил анонимно. Результат опроса в таблице 4.2

Таблица 4.2 — Средня оценка изображения при глубине трассировки

Количество точек	**Оценка реалистичности**
20	3.7
40	5.1
60	6.2
80	6.95
100	7.4
120	7.4
140	7.4
160	7.35
180	7.55
200	7.55
220	7.6875
240	7.8125
260	7.6875
280	7.6875
300	7.6875

Вывод

В данном разделе была описана цель исследования, технические характеристики устройства, на котором проведено исследование, а также приведены результаты исследования.

Было проведено исследование зависимости времени выполнения реализации алгоритма визуализации облака с использованием распараллеливания цикла и без.

Респонденты оценили вариант с количеством точек на трассируемом луче равным 240 максимальной оценкой 7.8/10. При таком значении время выполнения алгоритма визуализации облака занимает 265 мс, что обеспечивает получение 4 кадров в секунду. Это свидетельствует тому, что алгоритм нуждается в оптимизациях для достижения результата в 30 кадров в секунду.

ЗАКЛЮЧЕНИЕ

В ходе работы были решены задачи, связанные с визуализацией ландшафтной сцены с облаками. Для этого был выбран алгоритм обратной трассировки лучей для визуализации объемных облаков, а также метод аппроксимации ландшафта примитивами. Для процедурной генерации ландшафта применён шум Перлина, а для облаков — шум Ворли—Вороного.

Была предложена оптическая модель облаков, а также выбраны алгоритмы для решения ключевых задач компьютерной графики. Для устранения невидимых линий и поверхностей использован Z-буфер, а для обработки теней и освещения — закон Ламберта и метод Фонга.

Разработано программное обеспечение, произведена его декомпозиция с описанием алгоритмов. Было выполнено модульное тестирование, и программа успешно прошла все тесты. Покрытие программы тестами составляет 19.83%.

Также были проанализированы результаты исследования, в ходе которого проведена оценка производительности алгоритма визуализации облаков. Оценка респондентами качества реализации с количеством точек на трассируемом луче равным 240 составила 7.8/10. Однако время выполнения алгоритма (265 мс на кадр) требует оптимизаций для достижения производительности в 30 кадров в секунду.

При дальнейшем развитии работы потребуется улучшение алгоритма визуализации облака с целью повышения производительности, например, за счёт распараллеливания на графической карте. Усложнение ландшафта и улучшение качества облаков, например, за счёт использования других оптических моделей [4] и других шумов, также могут стать направлениями дальнейшего развития работы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Häggström Fredrik. Real-time rendering of volumetric clouds: Master's thesis. 2018. URL: https://api.semanticscholar.org/CorpusID:49429466.

2. de Parga Jiménez, Carlos, Palomo Gómez [и др.]. Efficient Algorithms for Real-Time GPU Volumetric Cloud Rendering with Enhanced Geometry. 2022. Дата доступа: 2024-11-27. URL: https://doi.org/10.3390/sym10040125.

3. Ушаков Д.Н. Толковый словарь русского языка: В 4-х томах. Москва: Государственный институт «Советская энциклопедия», 1935–1940.

4. Guerrilla Games. The Real-Time Volumetric Cloudscapes of Horizon Zero Dawn. 2023. Дата доступа: 2024-10-07. URL: https://www.guerrilla-games.com/read/the-real-time-volumetric-cloudscapes-of-horizon-zero-dawn.

5. UNIGINE. Волюметрические облака в UNIGINE2. 2022. Дата доступа: 2024-11-27. URL: https://habr.com/ru/companies/unigine/articles/721458.

6. Weinrauch A., Lorbek S., Tatzgern W. [и др.]. Efficient Cloud-Based Rendering of Real-TimeVolumetric Clouds: Master's thesis. 2013. URL: https://diglib.eg.org/bitstream/handle/10.2312/hpg20231138/077-088.pdf.

7. Real-time Volumetric Rendering: Master's thesis. 2013. http://patapom.com/topics/Revision2013/Revision%202013%20-%20Real-time%20Volumetric%20Rendering%20Course%20Notes.pdf.

8. Perlin Noise: What is it, and how to use it. 2024. Дата доступа: 2024-12-08. URL: https://blog.hirnschall.net/perlin-noise/.

9. Bird K., Dickerson T., George J. Techniques for Fractal Terrain Generation. 2013. HRUMC Presentation. URL: web.williams.edu/Mathematics/sjmiller/public_html/hudson/Dickerson_Terrain.pdf.

10. More Noise: Cellular Noise. 2024. Дата доступа: 2024-12-08. URL: https://thebookofshaders.com/12/.

11. Роджерс Д. Алгоритмические основы машинной графики. Москва: Мир, 1989.

12. А.В. Куров. Конспект лекций по дисциплине «Компьютерная графика». 2024 год.

13. The Rust Programming Language. 2024. Дата обращения: 2024-11-27. URL: https://doc.rust-lang.org/.

14. egui: A simple, fast, and portable GUI library for Rust. 2024. Дата обращения: 2024-11-27. URL: https://github.com/emilk/egui.

15. Cargo – the Rust package manager. 2024. Дата обращения: 2024-12-10. URL: https://doc.rust-lang.org/cargo/.

16. cargo-test - Execute unit and integration tests of a package. 2024. Дата обращения: 2024-12-10. URL: https://doc.rust-lang.org/cargo/commands/cargo-test.html.

17. Cargo-Tarpaulin is a tool to determine code coverage achieved via tests. 2024. Дата обращения: 2024-12-10. URL: https://docs.rs/crate/cargo-tarpaulin/0.6.11.

18. Huawei. Huawei Matebook 14. 2024. Дата обращения: 06.10.2024. URL: https://consumer.huawei.com/en/laptops/matebook-14-2024/.

19. Intel Corporation. Intel® Core™ Ultra 5 Processor 125H. 2024. Дата обращения: 2024-11-05. URL: https://www.intel.com/content/www/us/en/support/ru-banner-inside.html.

20. Rayon – data-parallelism library. 2024. Дата обращения: 2024-12-10. URL: https://docs.rs/rayon/latest/rayon/.

Приложение А

Презентация к курсовой работе