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 КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:

Моделирование реалистичных изображений облаков

Студент ИУ7-51б	_____	Сушина А.Д.
(Группа)	(Подпись, дата)	(И.О.Фамилия)
Руководитель курсового проекта	_____	Мартынюк Н.Н.
	(Подпись, дата)	(И.О.Фамилия)
Консультант	_____	_____
	(Подпись, дата)	(И.О.Фамилия)

2019г.

Оглавление

ВВЕДЕНИЕ

С появлением систем виртуальной реальности, в которых наблюдатель погружается в мир модели, возникает потребность визуализации виртуальной среды, максимально приближенной к естественным условиям. Такие объекты как ландшафт, водная поверхность, растительность, небо и облака являются неотъемлемой частью практически любой естественной сцены. Моделирование облаков как природного феномена является одной из наиболее важных и трудных задач, решаемых при визуализации реалистичного неба. Алгоритмы визуализации облаков используются во всех приложениях, где необходимо создать максимально естественную среду, а также в приложениях, в которых облака играют основную роль — в авиасимуляторах.

Целью данной работы является изучение различных алгоритмов визуализации облаков и реализация одного из них. При этом необходимо обеспечить возможность настройки параметров облака (размеров, плотности).

Для достижения этой цели поставлены следующие задачи.

1. Изучить предметную область.
2. Изучить существующие алгоритмы визуализации облаков и провести их сравнительный анализ.
3. Разработать программу на основе одного из существующих алгоритмов. Программа должна соответствовать техническому заданию и создавать реалистичные изображения облаков.

1. АНАЛИТИЧЕСКИЙ РАЗДЕЛ

В данном разделе будут рассмотрены существующие алгоритмы и их особенности, будет проведен анализ преимуществ и недостатков каждого метода. Также будет выбран метод реализации.

1.1 Анализ предметной области

При решении проблемы отображения больших открытых пространств качественная визуализация облаков оказывается мощным инструментом повышения реализма виртуальной обстановки.

Облака представляют собой динамический объект, структура которого сложна и неоднородна: плотность, с которой распределены по облаку капли воды неравномерна как на границах, так и внутри облака. Она меняется постоянно, в особенности при изменении погоды и при появлении осадков. Но при визуализации внимание уделяется в основном частицам на поверхности облака, тогда как плотность частиц, находящихся внутри, может считаться постоянной, поскольку она влияет лишь на преломление и рассеивание света, и при визуализации это можно отразить, придавая облакам белый цвет.

1.2 Алгоритмы моделирования облаков

Облака по своей структуре являются объектами, напрямую не представимыми в полигональной форме. На данный момент известно три способа аппроксимации неба и облаков:

1. Использование заранее нарисованных текстур неба и интерполяция между ними.
2. Генерация двумерных изображений облаков с помощью шума Перлина.
3. Представление облака как набор полигонов с текстурами, всегда обращенными к наблюдателю.

Рассмотрим каждый из способов отдельно.

1.2.1. Использование заранее нарисованных текстур неба и интерполяция между ними

Если камера находится достаточно близко к поверхности земли, то можно использовать заранее нарисованные текстуры для отображения неба. Тогда небесную

поверхность можно представить как плоскость, с нанесенной на нее текстурой. Для достижения большей реалистичности следует представить небесную поверхность как небесную сферу, т.е. вместо плоскости использовать некую поверхность, представляющую собой часть сферы. Эта поверхность должна быть пологой, чтобы создавался эффект удаленности горизонта. За счет использования готовых текстур высокого качества можно добиться практически фотореалистичного качества.

Для визуализации динамического неба можно использовать следующие подходы:

- Использовать несколько слоев текстур и сдвигать их относительно друг друга.
- Использовать интерполяцию между несколькими картинами неба.

Для визуализации смены дня и ночи можно изменять гамму цветов в зависимости от времени суток и применять пост-эффекты свечения солнца и луны.

У данного метода есть существенные недостатки:

1. Построить изображение можно только в случае, если наблюдатель находится близко к поверхности земли. Невозможно построить изображения для произвольной позиции наблюдателя.
2. Для того, чтобы получить реалистичную картину неба необходим большой набор текстур высокого разрешения для интерполяции. Это приводит к большим затратам текстурной памяти.

Преимущество данного метода заключается в возможности получения высоко качества изображения.

Данный метод может быть использован для систем, не предъявляющих высоких требований к динамичности неба и облаков, в которых наблюдатель расположен близко к земле. [4]

1.2.2. Генерация двумерных изображений облаков с помощью шума Перлина

Использование шума Перлина для моделирования облаков является наиболее распространенным способом из-за его простоты и достаточной степени реалистичности. Алгоритм шума Перлина широко применяется для генерации туманов, облаков, огня и различных эффектов. Он основывается на том же представлении небесной поверхности как часть сферы, а облаков как двумерную текстуру.

Облачная поверхность обладает неравномерной структурой, но она не совсем хаотична. В одних местах возможно скопление "массы", а в других некая разреженность. Следовательно, для визуализации облаков можно использовать хаотичную функцию, ядро которой подчиняется некоторому определенному закону. На роль такой функции отлично подходят функции Perlin Noise. [5]

Функции шума Перлина являются комбинацией шумовой функции и интерполяционной функции. Аргументами для шумовой функции выступают целые числа.

Чтобы реализовать шум Перлина, необходимо построить ряд функций, основными величинами для которых будут частота и амплитуда. Амплитуда - значения локальных экстремумов, а частота - величина, обратная длине волны. Сначала строится функция с минимальной выбранной частотой и максимальной амплитудой. Затем, для построения каждой следующей функции постепенно увеличивается частота и уменьшается амплитуда. Последняя функция будет иметь максимальную частоту и минимальную амплитуду. Каждая из функций называется октавой. Затем необходимо сложить все эти функции. В результате получается функция обладающая гладкостью и в то же время достаточной степенью хаотичности. [6]

Алгоритм работы показан на рисунке 1.1

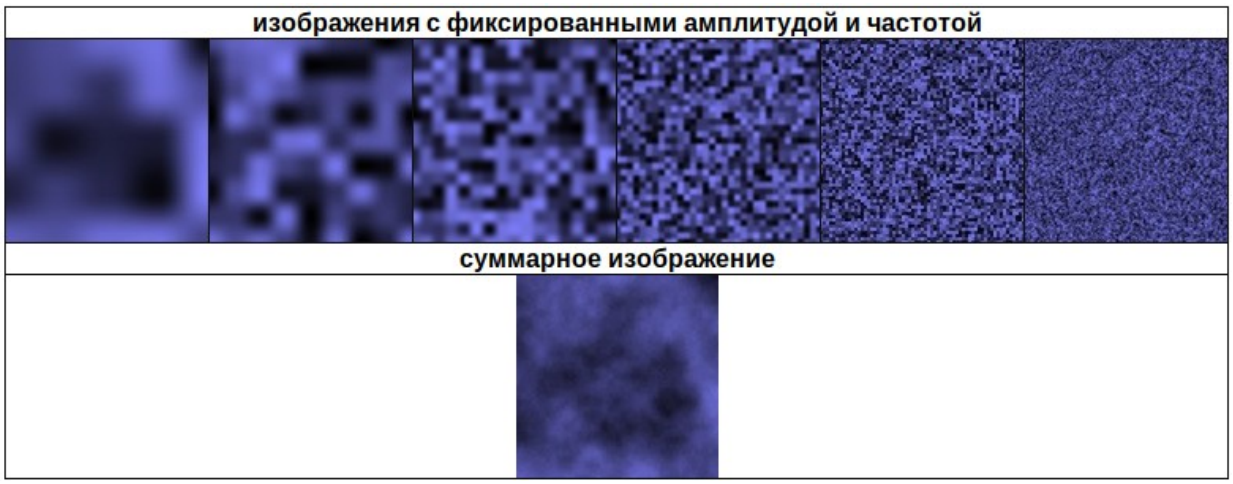

Рис 1.1 Алгоритм работы шума Перлина для реализации текстуры облаков

Аналогично, можно реализовать алгоритм, который с помощью этих функций будет создавать изображение, соответствующее конечной функции.

С помощью шума Перлина можно создавать двумерные карты облаков. Для регулирования плотности облаков достаточно будет установить все пиксели меньше

какого-то значения в цвет фона. На рисунке 1.2 можно увидеть результат такого отсечения.

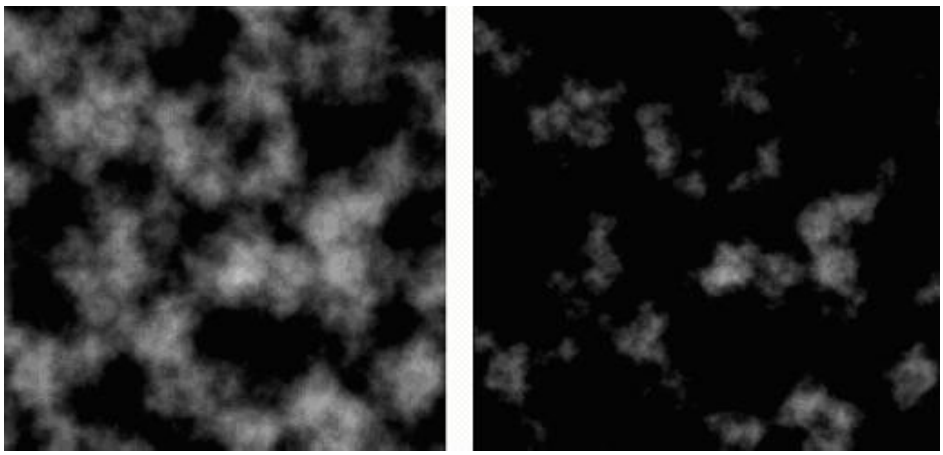

Рис 1.2 Регулирование плотности облаков посредством отсечения части пикселей с минимальными значениями

Преимущество данного метода перед предыдущим заключается в отсутствии необходимости выделения дополнительной памяти под текстуры. На реалистичность изображения влияет количество октав. Облака могут двигаться относительно наблюдателя, динамически исчезать и появляться. Однако это все также двумерная текстура и, как и в предыдущем случае, невозможно построить объемное изображение и рассмотреть облако со всех сторон.

1.2.3. Трехмерное моделирование облаков

Трехмерное моделирование облаков является более реалистичным и физически точным. Оно позволяет визуализировать гораздо большее число явлений, происходящих в облаках. В силу своей структуры облака не могут быть напрямую представлены полигональными моделями, что затрудняет их отображение как трехмерных объектов. Можно представить облако в виде набора частиц с текстурами прозрачности. В этом случае контур и изображение облаков определяется перекрытием полупрозрачных частиц, что соответствует физической структуре настоящего облака.

Модель облачной поверхности (то есть такой поверхности, которая покрывает все небо в определенном диапазоне) представляет из себя трехмерную решетку. В узлах этой решетки находятся воксели - минимальная графическая единица, операции над которой выполняются как над единым целом и которая визуализируется с помощью простейшего графического объекта.

Для создания частиц облака можно использовать возможности шума Перлина. В алгоритме шума Перлина нет никакого ограничения размерности, он может быть сколь угодно больших измерений. Результаты трехмерного шума Перлина можно с успехом использовать для создания формы объемным облакам. В этом случае яркость выступает размером частицы и может служить ключом к выбору текстуры для этой частицы, что увеличивает возможности влияния на форму облака, прозрачные места в текстуре означают отсутствие частиц. [4]

Существует также вариант реализации основанный на двумерной карте высот, полученной на основе двумерного шума Перлина. В таком случае в памяти компьютера не требуется хранить трехмерную решетку — для моделирования трехмерного облака используется только карта высот. Ее создание является наиболее важной задачей, поскольку именно она будет отвечать за форму и структуру облачности, а следовательно, влиять на ее реалистичность. При этом, все эффекты (изменение формы облака, ветер) реализуются за счет перемещения по карте высот, которая, соответственно, должна иметь достаточную площадь. На каждый пиксель карты приходится одна из градаций серого, представленная, например, числом от 0 до 255. На основе карты строится трехмерная сетка, длина и ширина которой совпадают с размерами текстуры, а высота – с некоторым заранее определенным числом. Для каждого вокселя предлагается использовать параметр, отвечающий за присутствие облака в соответствующей точке решетки. Это значение должно проставляться для всех элементов трехмерной сетки в соответствии с картой высот. Воздействие ветра на такую структуру, ее перемещение, изменение, можно реализовать за счет сдвига заполненных вокселей относительно трехмерной сетки. [5]

Использование шума Перлина привносит все свои возможности и достоинства в алгоритм генерации – высокая скорость просчета, возможность управлять плотностью облаков, эффекты клубящихся облаков, постепенного проявления и так далее. Для динамической картины неба с успехом можно использовать четырехмерный шум Перлина, где еще одной координатой выступает время.

Основная проблема этого метода визуализации облаков — затрата ресурсов на их просчет и отображение. Для физического моделирования приходится использовать большие массивы данных с существенным с точки зрения затрат вычислительных ресурсов количеством итераций для получения реалистичной картинки. Для получения

динамической картины необходимо продолжать просчеты в реальном времени в масштабах видимости облаков, что может ощутимо повлиять на производительность.

Вторая проблема – затраты ресурсов на отображение облаков. В зависимости от необходимой детализации облако может включать от сотен до тысяч частиц, каждая из которых занимает определенный размер на экране, и каждая всегда рисуется с учетом полупрозрачности. С другой стороны, метод является наиболее общим и универсальным методом представления облаков и легко модифицируется для любых требований – таких, как полет сквозь облако, возможность рассмотреть облако под любым углом, полет над облачным слоем и т. д. [3]

1.3 Алгоритмы визуализации облаков

Из предыдущего раздела видно, что методы моделирования облаков сильно отличаются от методов моделирования других объектов. В методах, основанных на представлении объектов поверхностями, сначала создается промежуточная модель на базе плоских треугольников. Далее выполняется визуализация объектов. В то же время методы, основанные на воксельном представлении объемов, создают трехмерное изображение объекта непосредственно из объемных данных. Объемно-ориентированная технология визуализации отличается от традиционной растровой полигональной графики кардинальным образом. В полигональной растровой графике объекты задаются поверхностями, представленными полигонами, накладываемыми на проволочный каркас модели. Такое задание объектов достаточно для игровых приложений, анимационных и синтетических объектов, но совсем недостаточно для отображения внутренней структуры естественных объектов или явлений. [3]

1.3.1. Использование билбордов

Билборд – это полигон, все время направленный на наблюдателя. В данном подходе облачная поверхность представляет собой трехмерную решетку, в узлах которой находятся билборды. С помощью шума Перлина, путем складывания октав шума, генерируется трехмерная карта облачности. На основе этой карты выставляются позиции билбордов. После чего к каждому билборду применяется небольшая текстура, соответствующая части облака. Для корректного отображения облачной поверхности необходимо производить сортировку билбордов от дальнего к ближнему относительно позиции камеры. Также возможно применение следующих оптимизаций: облачная поверхность делится на сектора, и выполняется проверка, какие из секторов попадают в

область видимости камеры. Видимые сектора сортируются от дальнего к ближнему, а не входящие в область видимости отсекаются. Необходимо отсекать сектора, полностью скрытые за другими секторами. Также для всех билбордов в видимых секторах необходимо рассчитать цвет с учетом источников света, плотности и рассеяния. Данный подход дает хорошие показатели реалистичности и обеспечивает работу в режиме реального времени. [1]

1.3.2. Визуализация с помощью вокселей

Отличием данного подхода от предыдущего является использование вокселей вместо билбордов. Таким образом, небо представляется трехмерной решеткой, в узлах которой находятся вокселы. С помощью шума Перлина генерируется трехмерная карта облачности, далее на основе этой карты происходит выставление позиций вокселей. После чего, при помощи трассировки лучей света, рассчитывается цвет вокселей с учетом источников света, плотности облаков и рассеяния света. Данный подход дает более детализированный результат в сравнении с билбордами, но является более ресурсозатратным и сложным в оптимизации. [1]

В таком случае вводятся три битовых параметра для каждого вокселя, которые отражают его физическую природу как части облака:

hum - если равен 1, то это означает, что в данном вокселе собралось достаточно пара для формирования облака.

act - если равен 1, то это означает, что в данном вокселе началась фаза преобразования из пара в воду (то есть, в облако).

cld - если равен 1, то это означает, что в данном вокселе существует облако.

Работая только с этими параметрами, можно изменять структуры облачной поверхности. Можно реализовывать различные эффекты, такие как зарождение облаков, угасание облаков, перемещение облаков, изменение структуры облаков и др. [2,5] На рисунке 1.3 представлено графическое представление данного метода.

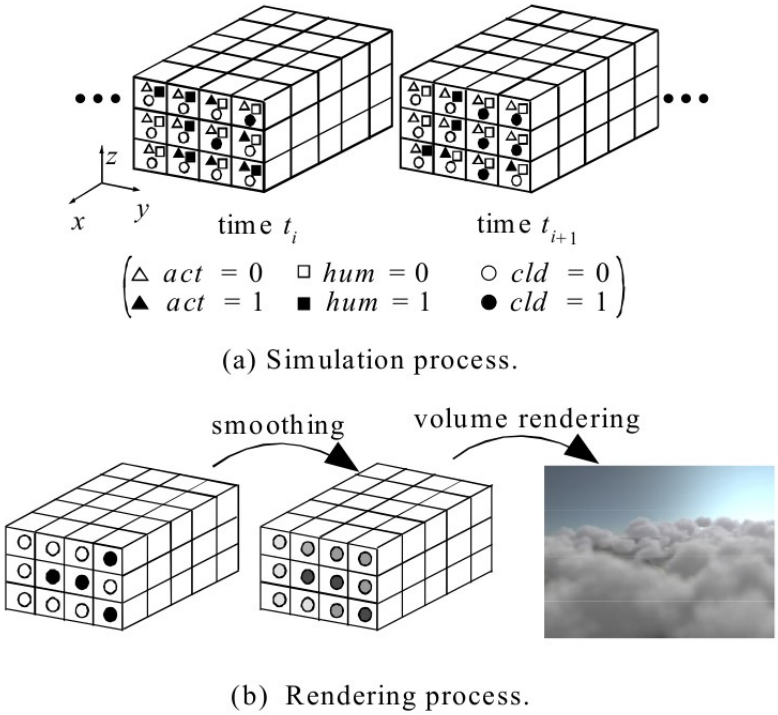

Рис 1.3 Графическое представление метода визуализации облаков на основе вокселов и трех битовых полей

1.3.3. Перспективная проекция

Важной составляющей сцены является точка обзора или камера. Камера позволяет получить реалистичное изображение объектов, основываясь на принципах проецирования. Важными параметрами для камеры являются точка обзора, угол обзора, размеры окна просмотра и расстояние до окна просмотра. Основываясь на этих параметрах можно получить изображение в перспективе, близкое к реальному. Так, объекты, находящиеся вдали, визуально выглядят меньше, а объекты, расположенные ближе к наблюдателю, кажутся больше. На рисунке 1.4 можно наблюдать пример того, как точка проецируется на экран камеры.

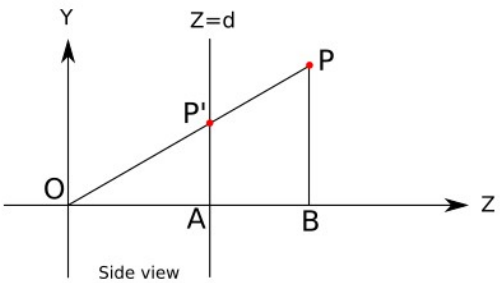

Рис 1.4 Пример проецирования точки в окно просмотра.

Этот метод реализации перспективы подходит и для облаков. Так, можно показать приближение облаков и даже реализовать пролет через них. Точки, которые находятся вне видимости камеры (находятся за ней или их проекции не попадают в окно просмотра) отбрасываются и появляется возможность наблюдать «внутренности» облачного слоя.

1.4. Выбор, обоснование метода моделирования и алгоритма

Для реализации этого проекта был выбран метод визуализации облаков, основанный на трехмерной решетке вокселов. Этот метод позволяет получить изображения высокого качества и высокой степени реалистичности. Облака, полученные этим методом, можно будет рассмотреть со всех сторон, реализовать эффект пролета сквозь облако и другие эффекты. Этот метод наиболее универсален, поэтому в дальнейшем может использоваться в различного рода приложениях.

Методы, основанные на 2D текстурах, не дадут достаточно реалистичного трехмерного изображения, поэтому их невозможно использовать.

Для создания таблицы вокселов будет использован трехмерный шум Перлина, так как задание значений вокселов вручную требует слишком большого количества усилий. Таким образом облака будут генерироваться процедурно. Использование шума Перлина позволит использовать его преимущества и настраивать плотность облаков.

Для визуализации будет использован метод билбордов, то есть каждая частица облака будет представлена как небольшая текстура, всегда обращенная к наблюдателю. Также будет реализована камера и использована перспектива для создания наиболее реалистичной сцены и возможности пролета сквозь облака.

2. КОНСТРУКТОРСКИЙ РАЗДЕЛ

В этом разделе будет представлен алгоритм решения поставленной задачи, а также описаны используемые типы и структуры данных, разработана структура программы.

2.1. Алгоритм работы приложения

Исходя из вышесказанного, можно составить следующий алгоритм работы приложения:

1. ввод пользовательских данных;
2. создание сцены;
3. создание трехмерной сетки на основе шума Перлина;
4. применение пользовательских настроек плотности и выборка значимых точек;
5. преобразования сцены: масштабирование, поворот и перемещение;
6. преобразование к координатам камеры;
7. визуализация полученной сцены.

Далее опишем применяемые алгоритмы и их математическую модель.

2.2. Разработка алгоритмов

2.2.1. Шум Перлина

Функция шума Перлина является результатом сложения нескольких шумовых функций с разными значениями амплитуды (amplitude) и частоты (frequency). Для построения этих функций введены следующие величины:

- seed — случайная величина, используемая в шумовой функции;
- persistence — величина, обеспечивающая зависимость амплитуда от частоты;
- octaves — количество октав в шуме.

Октавой (octave) называется каждая сгенерированная и добавленная в суммарный поток функция. Количество октав, также, является основной величиной в данном процессе и влияет на качество изображения.

Эти величины соотносятся между собой по следующим формулам:

$$frequency = 2^i, \quad (2.1)$$

$$amplitude = persistence^i, \quad (2.2)$$

где i — номер генерируемой функции.

Ядром функции служит шумовая функция, которая зависит от параметра seed. На вход функции подаются целые числа x, y, z, а на выходе получается число в диапазоне от -1 до 1.

Далее используется интерполяция, например, косинусная. На этапе интерполяции происходит сглаживание функции. Значение функции вычисляется по следующим формуле:

$$y = a * \left(1 - \frac{(1 - \cos(x * \Pi))}{2}\right) + \frac{b * (1 - \cos(x * \Pi))}{2}, \quad (2.3)$$

где y — искомое значение функции; a — начало отрезка; b — конец отрезка; x — точка, в которой необходимо найти значение.

Эти операции выполняются для каждой октавы, а затем полученный результат складывается в итоговую функцию шума Перлина.

Схема полученного алгоритма представлена на рисунке 2.1.

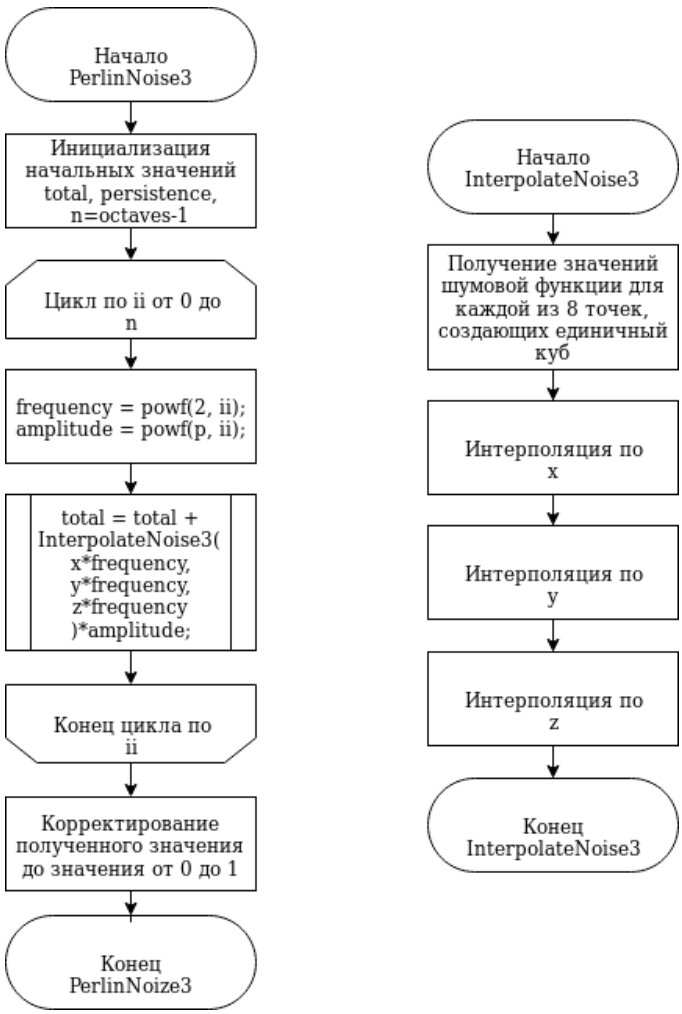

Рис 2.1 Схема алгоритма получения шума Перлина

2.2.2. Создание трехмерной сетки на основе шума Перлина

С использованием функции Перлина можно реализовать трехмерную сетку — VoxelGrid. Для каждого узла сетки необходимо получить значение функции шума Перлина. Это значение будет отвечать за прозрачность данного вокселя.

Используем следующие структуры данных:

1. Структура одного вокселя:

```
Voxel {
    QColor qcolor;
    double density;
};
```

2. Структура сетки:

```
VoxelGrid {
    int xcount, ycount, zcount;
    double voxelsize;
    double defaultDensity;
    vector<Voxel> grid;
}
```

Для заполнения сетки достаточно пройтись по всем элементам и присвоить каждому вокселю новое значение прозрачности, полученное с помощью функции Перлина.

При такой реализации многие воксели получают значения близкие или равные нулю. При визуализации сцены они будут только замедлять работу системы. Чтобы увеличить скорость работы, реализуем кеш, в котором будем хранить все значимые точки, а так же их цвета. Стандартный цвет каждого вокселя — белый. Также во время выполнения записи в кеш можно наложить пользовательские настройки и отсечь некоторые точки, которые не удовлетворяют установленному значению плотности.
Схема алгоритма представлена на рисунке 2.2

Рис 2.2 Схема алгоритма создания трехмерной сетки на основе шума Перлина

2.2.3. Преобразования камеры

Для того, чтобы была возможность рассмотреть сцену со всех сторон, необходимо реализовать преобразования камеры: масштабирование, поворот и перенос.

Для реализации матричных преобразований перейдем к однородным координатам [x,y,z,1] . В таком виде можно преобразовывать точки, умножая на матрицу преобразований размером 4x4.

Трехмерный перенос реализуется матрицей $T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ dx & dy & dz & 1 \end{bmatrix}$.

Однако, гораздо проще и быстрее просто прибавить необходимое смещение к текущей координате.

Трехмерное частичное изменение масштаба реализуется с помощью матрицы

$$S=\begin{bmatrix} kx & 0 & 0 & 0 \\ 0 & ky & 0 & 0 \\ 0 & 0 & kz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Однако умножение на эту матрицу дает масштабирование только относительно начала координат. Более полное масштабирование реализуется с помощью формул:

$x = Cx + kx * (x - Cx)$, (2.4)

$y = Cy + ky * (y - Cy)$, (2.5)

$z = Cz + kz * (z - Cz)$, (2.6)

где C(Cx, Cy,Cz) — центр масштабирования.

Поворот вокруг оси Z описывается матрицей $Rz=\begin{bmatrix} \cos(a) & \sin(a) & 0 & 0 \\ -\sin(a) & \cos(a) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$

Поворот вокруг оси X описывается матрицей $Rx=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(a) & \sin(a) & 0 \\ 0 & -\sin(a) & \cos(a) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$

Поворот вокруг оси Y описывается матрицей $Ry=\begin{bmatrix} \cos(a) & 0 & -\sin(a) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(a) & 0 & \cos(a) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$

Применяя эти формулы, можно двигать, поворачивать и масштабировать изображение.

2.2.4. Перспектива

Рассмотрим любую камеру как точку - центр проецирования и экран - плоский прямоугольник в 3D пространстве, на плоскость которого идет проецирование. Наша стандартная камера, например, задается точкой (0, 0, -dist) и экраном с вершинами (-xSize/2, ySize/2), ..., (xSize/2, -ySize/2). Можно задать эту систему тремя векторами, задающими с точки зрения камеры направления вперед, вправо и вверх; вектор "вперед" соединяет центр проецирования и центр экрана, вектор "вправо" соединяет центр экрана и правую его границу, вектор "вверх", соответственно, центр экрана и

верхнюю его границу. Обозначим эти вектора как p, q и r соответственно, а центр проецирования за s. Пример для стандартной камеры показан на рисунке 2.3.

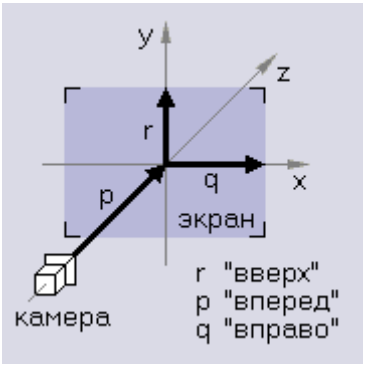

Рис 2.3 Стандартная камера

Здесь (для стандартной камеры; обозначим ее вектора как Sp, Sq, Sr, Ss)

$$Sp = p = (0,0,dist),$$

$$Sq = q = \left(\frac{xSize}{2},0,0\right),$$

$$Sr = r = \left(0,\frac{ySize}{2},0\right),$$

$$Ss = s = (0,0,-dist).$$

Любые три взаимно перпендикулярных вектора и точка - центр координат задают в 3D пространстве систему координат. Так что объект мы можем рассматривать в системе обычных координат (x,y,z), в системе координат стандартной камеры (Sp,Sq,Sr) или в системе (p,q,r), соответствующей какой-то произвольной камере. В любом случае, если (a,b,c) - координаты точки в системе координат камеры (точнее, в системе координат с центром в точке s и базисом (p,q,r)), то координаты проекции точки на экране равны

$$screenX = \frac{xSize}{2} + \frac{xSize}{2} * \left(\frac{a}{c}\right), \quad (2.7)$$

$$screenY = \frac{ySize}{2} + \frac{ySize}{2} * \left(\frac{b}{c}\right), \quad (2.8)$$

В случае стандартной камеры переход от обычной системы координат к системе координат камеры очевиден:

$$a = \frac{x}{\left(\frac{xSize}{2}\right)}, \quad (2.9)$$

$$b = \frac{y}{\left(\frac{ySize}{2}\right)}, \quad (2.10)$$

$$c = \frac{z}{\left(\frac{zSize}{2}\right)}. \quad (2.11)$$

Подставив это в формулы для screenX, screenY, получим как раз те самые формулы для проекции на стандартную камеру.

3. ТЕХНОЛОГИЧЕСКИЙ РАЗДЕЛ

3.1 Выбор и обоснование языка программирования

В данном проекте будет использоваться объектно-ориентированный подход, в связи с тем, что проект достаточно объемен и может расширяться или модифицироваться во время его написания.

Для разработки был выбран язык C++ и среда разработки QtCreator. Выбор сделан в пользу данного языка программирования, так как он быстрее аналогов, предоставляет все необходимые возможности, а также знаком лично мне. Среда разработки позволяет создать удобный интерфейс для приложения, а так же упрощает работу с классами и позволяет использовать встроенные типы. Также среда разработки позволяет работать с пиксельными картами, что является необходимым для реализации данного курсового проекта.

3.1. Структура приложения

В программе реализованы следующие классы:

- class point — описывает трехмерную точку и возможные операции над ней;
- class Matrix — описывает матрицу любого размера типа double и операции над ней;
- class MyVector(vec3) — описывает трехмерный математический вектор и операции над ним;
- class Noise — описывает функцию шума Перлина;
- class Voxel — описывает структуру вокселя и возможные операции над ним;
- class VoxelGrid — описывает трехмерную решетку, состоящую из вокселей;
- class Scene — описывает текущую сцену и методы обращения к ней;
- class Cloud — описывает облачный слой, методы его генерации.
- class camera — описывает камеру;
- class MainWindow — реализует главный интерфейс приложения и работу с пользователем.

Описание классов представлено в приложении А.

Диаграмма классов представлена на рисунке 3.1.

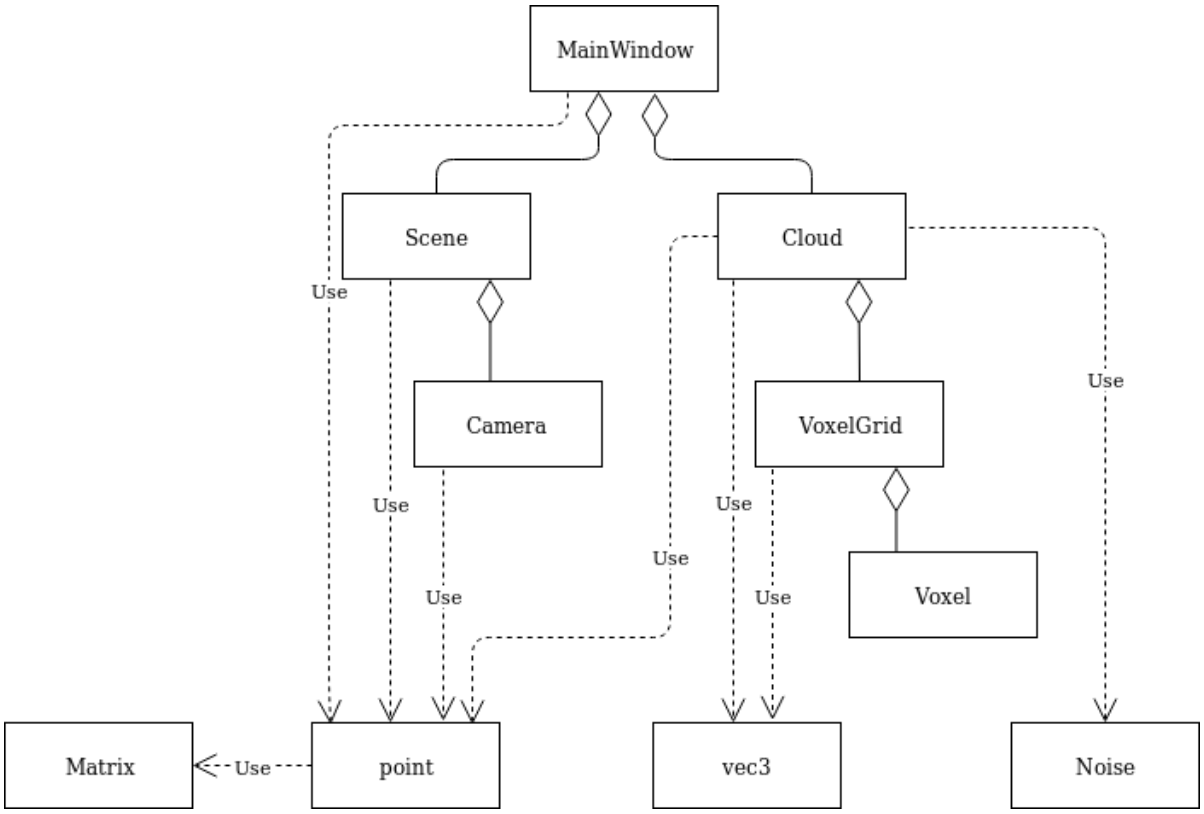

Рис 3.1 Диаграмма классов

3.2. Интерфейс пользователя

На рисунке 3.2 представлен интерфейс пользователя. Интерфейс предоставляет возможность ввести собственный размеры сетки, указать плотность, а так же запустить некоторые примеры. Окно включает в себя: экран, на который выводится изображение; колонку настроек, расположенную справа; а также выпадающее меню в верхней панели.

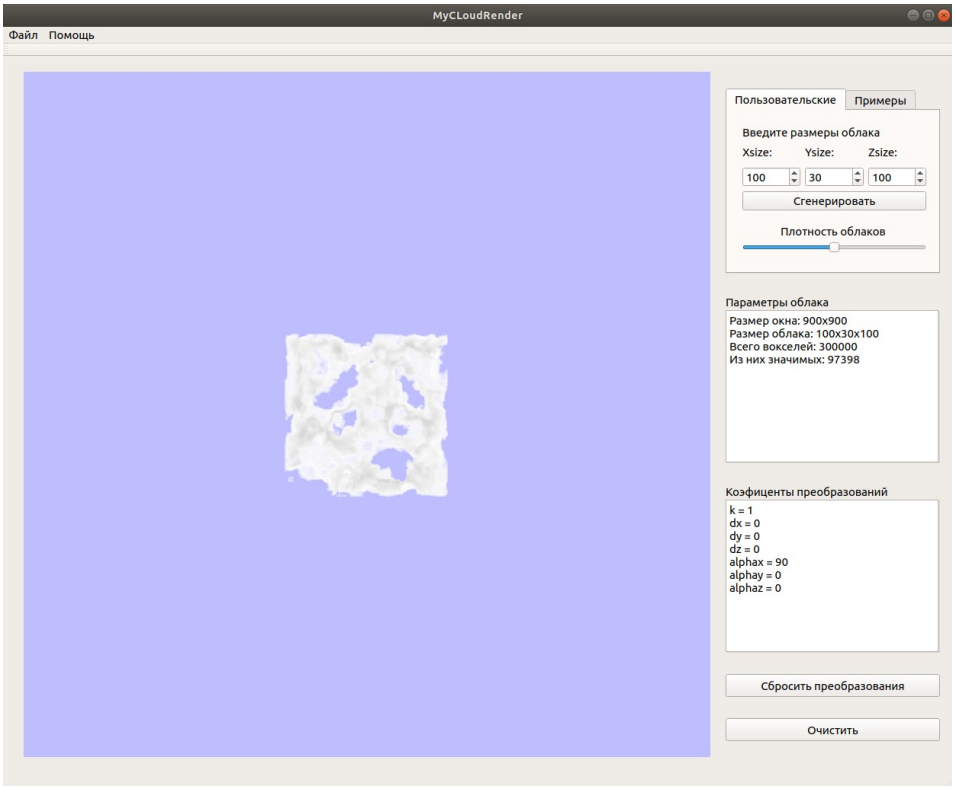

Рис 3.2 Интерфейс пользователя

Значения Xsize, Ysize, Zsize могут быть в диапазоне от 1 до 200. Значения плотности задаются с помощью ползунка.

В окне «Параметры облака» (рис 3.3) выводятся сведения о размере окна, размере сетки, количестве вокселей и количестве значимых точек. В окне «Коэффициенты преобразований» (рис 3.4) содержится информация о текущих преобразованиях, совершенных над объектом.

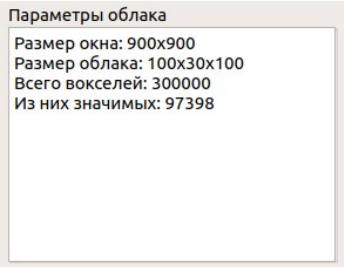

Рис 3.3 Окно «Параметры облака»

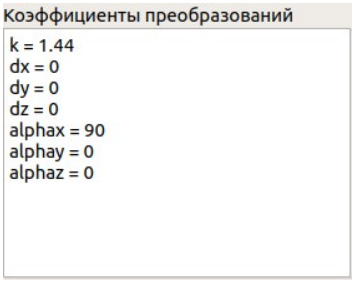

Рис 3.4 Окно «коэффициенты преобразований»

Описание кнопок:

- «Сгенерировать»

При нажатии на кнопку, считываются значения Xsize, Ysize, Zsize, а затем запускается процесс генерации рандомного облака на сетке с заданными размерами. После чего, к полученному результату применяются преобразования и выводится изображение на экран.

- «Сбросить преобразования»

При нажатии на кнопку, сбрасываются все преобразования до стандартных и выполняется перерисовка сцены.

- «Очистить»

При нажатии на клавишу очищается сцена, возвращаются к стандартным параметры преобразования, удаляются все данные о ранее сгенерированных облаках.

- «Примеры»

При нажатии на таб «Примеры» появляются новые кнопки, каждая из которых реализует один из стандартных примеров того, как может работать программа.

В верхней панели расположено выпадающее меню (рис 3.5). С его помощью можно сохранить данные об облаке в файл, загрузить из файла, а также изучить доступные команды.

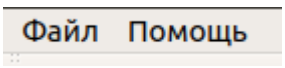

Рис 3.5 Верхняя панель приложения

3.3. Схема управления с помощью клавиатуры

Параметры преобразования можно менять с помощью клавиш клавиатуры. Клавиши выполняют следующие действия:

1. Клавиши **W/S** предназначены для вращения облака вокруг оси OX.
2. Клавиши **A/D** предназначены для вращения облака вокруг оси OY.
3. Клавиши **Z/E** предназначены для вращения облака вокруг оси OZ.
4. Клавиши **H/K** предназначены для перемещения облака по оси OX.
5. Клавиши **J/U** предназначены для перемещения облака по оси OY.
6. Клавиши **N/I** предназначены для перемещения облака по оси OZ.
7. Клавиши **+/-** предназначены для масштабирования облака относительно начала координат.

3.4. Входные и выходные данные

Входными данными в программе являются размер сетки и плотность облака. Также можно задавать параметры преобразований с помощью клавиш клавиатуры.

На выходе получаем изображение облака, построенное по введенным данным и преобразованное с заданными параметрами.

4. ЭКСПЕРИМЕНТАЛЬНО-ИССЛЕДОВАТЕЛЬСКИЙ РАЗДЕЛ

В данном разделе проводится исследование зависимости времени генерации облака от размера сетки, а также исследование зависимости времени отображения сцены от количества изображаемых точек.

4.1 Технические характеристики ЭВМ, на которой произведены тесты

1) Процессор Intel® Core™ i3-7100U CPU @ 2.40GHz × 4
2) 4Gb RAM.
3) Операционная система Ubuntu 18.04.3 LTS.

4.2. Эксперименты по замеру времени

Для произведения замеров времени выполнения алгоритмов будет использована формула

$$t = \frac{Tn}{N}, \quad (4.1)$$

где N – количество замеров, t – время выполнения реализации алгоритма, Tn — время выполнения N замеров. Неоднократное измерение времени необходимо для построения более гладкого графика и получения усредненного значения времени.

Количество замеров будет взято равным 10.

Тестирование будет проведено на сетках размером от 10x10x10 до 150x150x150 с шагом 10. Будет замерено время генерации сетки, посчитано количество значимых точек и время визуализации сцены. Значение плотности точек для всех замеров одинаково.

Результаты эксперимента представлены в таблице 1.

Таблица 1 - Результаты эксперимента по замеру времени

Количество узлов сетки	**Время генерации облака, тики**	**Количество значимых точек**	**Времена визуализации сцены, тики**
1000	15793	73	1961

8000	123963	3457	10297
27000	430895	8013	23646
64000	1013586	17024	52471
125000	1925635	30741	103081
216000	3341274	57876	189617
343000	5408875	77101	260350
512000	8477204	148363	564600
729000	11198445	233115	758442
1000000	15408596	254394	976543
1331000	21565744	413814	1492167
1728000	27742464	491559	1748068
2197000	35175769	578615	2002050
2744000	45757372	794293	2878776

На рисунке 4.1 представлен график зависимости времени генерации облака от размера сетки. По графику видно, что образуется линейная зависимость. Соответственно, чем больше размер сетки, тем дольше будет генерироваться облако.

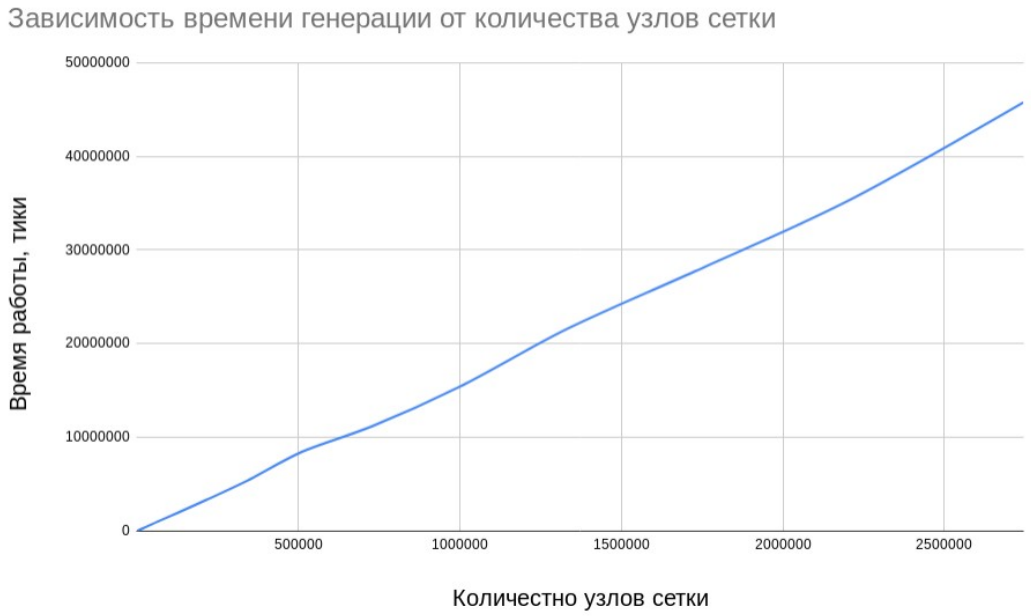

Рис 4.1 График зависимости времени генерации облака от количества узлов сетки

На рисунке 4.2 представлен график зависимости времени отображения сцены от количества значимых точек. Этот график также линеен.

Рис 4.2 График зависимости времени отображения сцены от количества значимых точек.

По результатам экспериментов можно сделать вывод, что алгоритмы работают линейно. Скорость генерации облака зависит только размера сетки. Время визуализации сцены зависит от количества значимых точек, которое меняется в зависимости от плотности облака. Таким образом, большое облако с маленьким количеством значимых точек может отображаться быстрее, чем маленькое облако с большим количеством значимых точек.

4.2. Примеры использования программы

4.2.1 Круглое облако

Примеры круглых облаков, сгенерированных полученной программы представлены на рисунках 4.3 — 4.5.

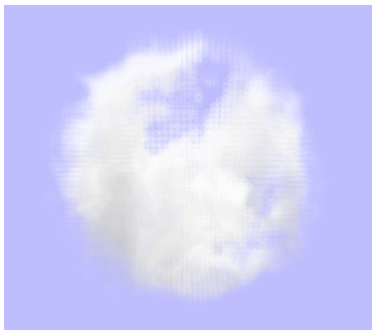

Рис 4.3 Круглое облако

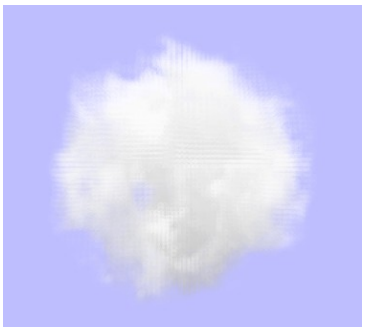

Рис 4.4 Круглое облако

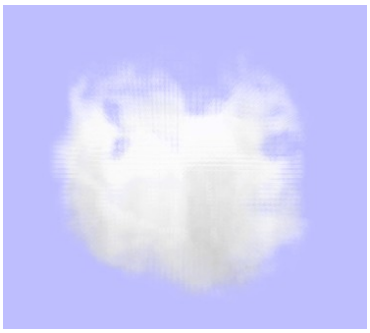

Рис 4.5 Круглое облако

4.2.2 Облачный слой разной плотности

На рисунках 4.6 — 4.7 представлены примеры облачного слоя разной плотности.

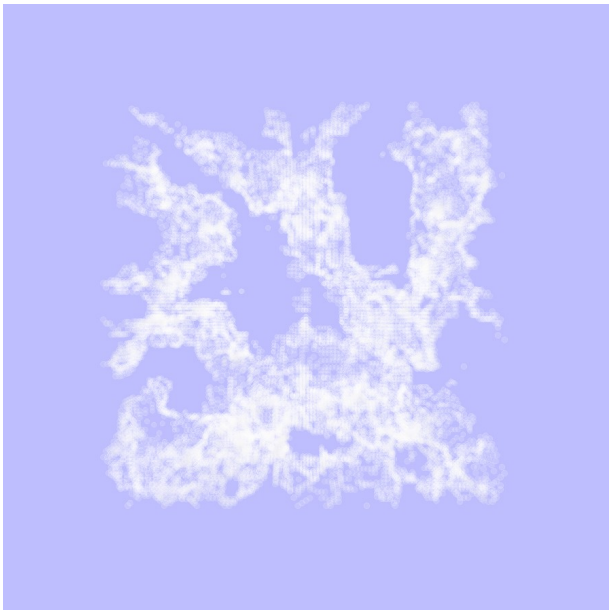

Рис 4.6 Облако малой плотности, сгенерированное программой

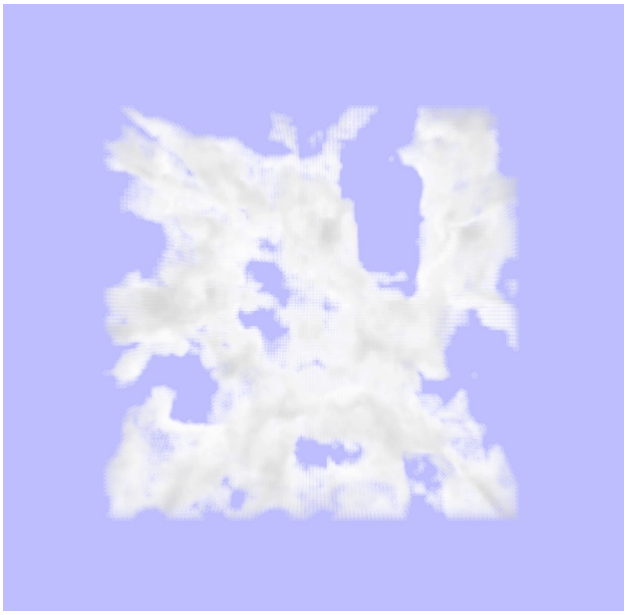

Рис 4.7 Облако средней плотности, сгенерированное программой

4.2.3 Пролет сквозь облака

На рисунках 4.8 — 4.14 представлены изображения, полученные изнутри облачного слоя при пролете через него.

Рис 4.8 Подлет к облаку

Рис 4.9 Движение внутри облачного слоя

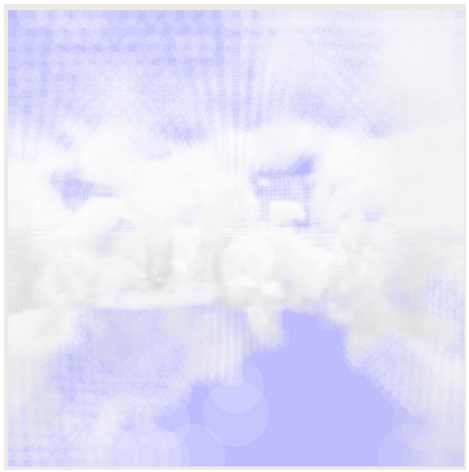

Рис 4.10 Движение внутри облачного слоя

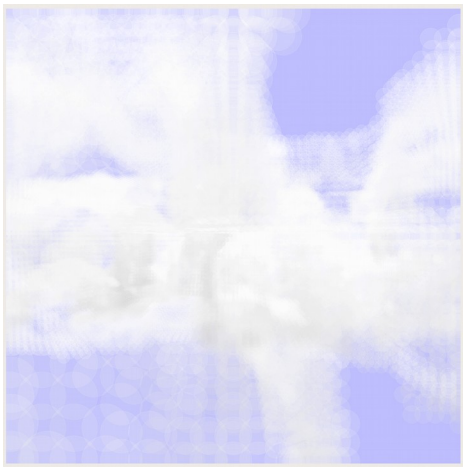

Рис 4.11 Движение внутри облачного слоя

Рис 4.12 Движение внутри облачного слоя

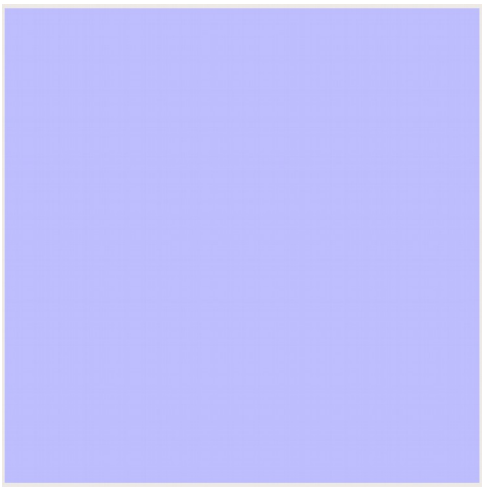

Рис 4.13 Окончание облачного слоя, вылет из облаков

ЗАКЛЮЧЕНИЕ

В рамках курсового проекта разработано программное обеспечение, реализующее визуализацию реалистичного изображения облаков.

Для визуализации трехмерных облаков была использована трехмерная сетка, в узлах которой находятся билборды — объекты, всегда обращенные к пользователю. Для генерации реалистичного облачного слоя был использован трехмерный шум Перлина.

Мной были изучены методы процедурной генерации облаков, способы их визуализации, а также алгоритмы отрисовки сцены.

Разработанная программа может генерировать облака разной плотности. Реализована возможность поворота, перемещения и масштабирования облачного слоя. Также реализована возможность пролета сквозь облака.

В результате сделаны следующие выводы.

1. Методы генерации облаков отличаются от традиционной растровой полигональной графики кардинальным образом. Поэтому многие алгоритмы к таким объектам неприменимы.
2. Трехмерная модель представления облака с помощью небольших частиц позволяет реализовывать различные природные эффекты, такие как пролет сквозь облака. Это дает возможность использовать данное представление для авиасимуляторов, где пролет сквозь облака — важнейшая составляющая.
3. Время генерации облачного слоя линейно зависит от количества узлов сетки. Однако время визуализации сцены зависит от количества значимых точек, которое может быть небольшим даже для больших сеток.

СПИСОК ЛИТЕРАТУРЫ

1. Агафонов, Н.А. МЕТОДЫ ВИЗУАЛИЗАЦИИ ПОГОДНЫХ ЯВЛЕНИЙ
2. В ИМИТАЦИОННЫХ СИСТЕМАХ / Н. А. Агафонов, А. М. Гиацинтов, А. В. Родителев // Федеральный научный центр Научно-исследовательский институт системных исследований РАН. Вестник кибернетики. - 2018.
3. Аксенов, Андрей. Компьютерная графика. URL: http://algolist.ru/graphics/3dfaq/index.php
4. Вяткин, С.И. ВИЗУАЛИЗАЦИЯ ПОЛУПРОЗРАЧНЫХ ОБЪЕКТОВ НА БАЗЕ ФУНКЦИЙ ВОЗМУЩЕНИЯ И ПРОЗРАЧНОСТИ / С.И. Вяткин, Б.С. Долговесов // Российская академия наук, Сибирское отделение, Автометрия. - Новосибирск, 2005. - Том 41, №3.
5. Елыков, Н.А. Практическая модель динамических атмосферных явлений при визуализации открытых пространств в системах визуализации реального времени / Н.А.Елыков, И.В.Белаго, С.М.Козлов С.А.Кузиковский, М.М.Лаврентьев // Лаборатория программных систем машинной графики ИАиЭ СО РАН. - Новосибирск, Россия.
6. Мальковский, Александр. Визуализация неба и облаков.//Введение в компьютерную графику. Полугодовой курс ВМиК МГУ, 2002. URL:https://www.graphicon.ru/oldgr/courses/cg02b/assigns/hw-5/hw5_cld.htm
7. Слеповичев, Сергей Олегович. ШУМ ПЕРЛИНА КАК СПОСОБ ПОЛУЧЕНИЯ КОМПЬЮТЕРНЫХ СПЕЦЭФФЕКТОВ ПРИРОДНЫХ ЯВЛЕНИЙ // Инновационное развитие ИП Сигитов Т.М. - Пермь, 2017.
8. Dobashi, Yoshinori. A Simple, Efficient Method for Realistic Animation of Clouds / Yoshinori Dobashi, Kazufumi Kaneda, Hideo Yamashita, Tsuyoshi Okita, Tomoyuki Nishita // Proceedings of the 27th annual conference on Computer graphics and interactive techniques. ACM Press/Addison-Wesley Publishing Co. New York, NY, USA — 2000. - Pages 19-28.
9. Wang, Niniane. Realistic and Fast Cloud Rendering / Niniane Wang, Microsoft Corporation - November 11, 2003.

ПРИЛОЖЕНИЯ

Приложение А

Листинг кода программы

Листинг А.1 - Класс point

```
struct point
{
    point();
    point(double x, double y);
    point(double x, double y, double z);
    point(double mas[3]);
    point(const point &p);
    double X;
    double Y;
    double Z;

    double x() const;
    double y() const;
    double z() const;
    void setY(double y);
    void setX(double x);
    void setZ(double z);

    point& operator =(const point &p);
    void rorateX(double a);
    void rorateY(double a);
    void rorateZ(double a);
    void move(double dx, double dy, double dz);
    void scaleUniform(point c, double k);
};
```

Листинг А.2 - Класс Matrix

```
class Matrix {
public:
    Matrix(size_t n, size_t m);
    void setMatrix(std::vector<double> elems);
    void addRow(size_t n, std::vector<double> row);
    double get_elem( size_t i , size_t j) const;
    std::vector<double>& operator [](size_t n);
    bool operator ==(const Matrix &m2);
    Matrix operator *(const Matrix &m2);
    Matrix Mul(const Matrix &m2);
    void make_random() ;
    size_t rows() const ;
    size_t cols() const ;
private:
```

```
    std::vector<std::vector<double>> matrix;

    Matrix MatrixMul(Matrix m1, Matrix m2);
};
```

Листинг А.3. - Класс MyVector

```
template <typename T, size_t size>
class MyVector{
public:
    MyVector() ;
    MyVector(T a1, T a2, T a3);
    MyVector(const MyVector<T, size> &v);
    T& operator [](size_t n);
    T operator [](size_t n) const;
    double length(); //длина
    double length2(); //длина в квадрате
    MyVector& normalize() const;
    MyVector& operator += ( const MyVector& v );
    MyVector& operator -= ( const MyVector& v );
    MyVector& operator *= ( const double d );
    MyVector& operator /= ( const double d );
    MyVector& operator - ();
    MyVector operator + (const MyVector& b);
    MyVector operator - (const MyVector& b);
    MyVector operator * (const double d);
    //скалярное произведение
    T operator * (const MyVector& b);
    MyVector operator / (const double d);
    bool operator == (const MyVector& b) ;
    bool operator != ( const MyVector& b);
private:
    std::vector<T> vec;
};

typedef MyVector<double, 3> vec3;
```

Листинг А.4 - Класс Noise

```
class Noise
{
public:
        Noise();
        ~Noise();
        Noise(float persistence, int octaves, int seed);

        float PerlinNoise3(float x, float y, float z);

private:
        float Noise3(int x, int y, int z);
```

```
        float SmoothNoise3(int x, int y, int z);

        float LinearInterpolate(float a, float b, float x);
        float CosineInterpolate(float a, float b, float x);
        float CubicInterpolate(float v0, float v1, float v2, float v3, float
x);

        float InterpolateNoise3(float x, float y, float z);

        float persistence;
        int octaves;
        int seed1, seed2;
};
```

Листинг А.5 - Классы Voxel и VoxelGrid

```
struct Voxel {
    QColor qcolor;
    double density; //плотность
};

class VoxelGrid
{
public:
    VoxelGrid();
    VoxelGrid(double size, int x, int y, int z, double d);
    ~VoxelGrid();
    void setVoxelColor(int x, int y, int z, vec3 rgb);
    void setVoxelDensity(int x, int y, int z, double q);

    double getVoxelDensity(int x, int y, int z);
    int getVoxelIndex(int x, int y, int z);
    std::vector<Voxel> getVoxelGrid();
    double getVoxelSize();
    double getDefaultDensity();
    int getMaxX(); int getMaxY(); int getMaxZ();
private:
    bool outside(int x, int y, int z);

    int xcount, ycount, zcount;
    double voxelsize;
    double defaultDensity;

    std::vector<Voxel> grid;
};
```

Листинг А.6 - Класс Scene

```
class Scene {
private:
    QPainter *painter;
```

```
    QPixmap *scene;
    bool clear_flag;
    camera cam;

public:
    double alphax=0, alphay=0, alphaz=0;
    double k = 1;
    double dx = 0, dy = 0, dz = 0;

    Scene();
    ~Scene();
    void init();
    void clear();
    QPixmap getPixmap();
    void drawCircle(point p, double r, QColor color = Qt::black);
    void drawPoint(point p, QColor color = Qt::black);
    void drawLine(point p1, point p2) ;
    void setColor(QColor color);
};
```

Листинг А.7 - Класс Cloud

```
class Cloud
{
public:
    Cloud();
    ~Cloud();
    void generateVoxelGridRandom(int seed);
    void generateVoxelGridRandom(int seed, int x, int y, int z);
    void putPointsToCache(double densityDelta);
    void renderFromCache(Scene &scene);

    VoxelGrid* getGrid();

    point getCenter();
    size_t cacheCount();
    void clear();

    void saveToFile(std::string filename);

    void readFromFile(std::string filename);

private:
    VoxelGrid *vGrid;
    camera *m_camera;

    std::vector<point> pointsCache;
    std::vector<QColor> colorCache;
}
```

Листинг А.8 - Класс camera

```
class camera
{
public:
    camera();
    CamPoint pointToCam(CloudPoint p);
    ScreenPoint CamToScreenStandart(CamPoint p);
    ScreenPoint ProjectVertex(point p, double &r);
    bool inCameraView(point p) ;
private:
    point center;
    int d;
    int pov;
    int Vh, Vw;
    vec3 up, straight, right;
};
```

Листинг А.9 - Класс MainWindow

```
class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
    void keyPressEvent(QKeyEvent *event);

private slots:
    void on_clear_button_clicked();
    void on_pushButton_2_clicked();
    void on_densitySlider_valueChanged(int value);
    void on_button_size_clicked();
    void on_clear_button_2_clicked();
    void on_button_exsample_2_clicked();
    void on_button_exsample_3_clicked();
    void on_button_exsample_4_clicked();
    void on_action_triggered();
    void on_action_2_triggered();
    void on_action_4_triggered();

private:
    void liting();
    void renderGrid();
    void renderFromCache();

    Ui::MainWindow *ui;
    Scene myScene;
    Axis xyz;
    Cloud generateCloud;
    VoxelGrid *grid;
    double densityDelta;
```

```
    double density;
};
```