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

Тема: Моделирование изображений облаков

Дисциплина: Компьютерная графика

Студент ИУ7-55Б _____ Д.О. Склифасовский

(Группа) (Подпись, дата) (И.О. Фамилия)

Руководитель проекта _____ А.А. Павельев

(Подпись, дата) (И.О. Фамилия)

Москва, 2020

Содержание

Введение

Область применения компьютерной графики не ограничивается одними художественными эффектами. Во всех отраслях науки, техники, медицины, в коммерческой и управленческой деятельности используются построенные с помощью компьютера схемы, графики, диаграммы, предназначенные для наглядного отображения разнообразной информации.

В наше время визуализация таких природных явлений, как ландшафт, водная поверхность, растительность, небо и облака стала важнейшей задачей. Все эти детали используются в разных сферах, например: разработка компьютерных игр, моделирование спецэффектов, создание анимации.

Существуют несколько алгоритмов компьютерной графики, которые решают данную задача, но чаще всего, они являются слишком ресурсозатратными. Для того, чтобы изображение выглядело более реалистичным, нужно потратить больше времени и памяти на его построение. Это является проблемой, для того чтобы создать динамическую сцену.

Цель данной работы - создание программного продукта, позволяющего генерировать изображения облаков.

Для достижения представленной цели, требуется решить следующие задачи:

1) проанализировать методы генерации облаков;
2) выбрать и/или модифицировать существующие алгоритмы трехмерной графики для визуализации трехмерного изображения;
3) реализовать выбранные алгоритмы для создания трехмерного изображения;
4) разработать программное обеспечение для отображения трехмерного изображения облаков.

1 Аналитический раздел

В данном разделе представлены и проанализированы способы представления данных об облаке и ландшафте, алгоритмы генерации карты высот, удаления невидимых линий и поверхностей и алгоритмы закрасок. Также поставлена задача.

1.1 Представление данных об облаке и ландшафте

Существуют несколько основных принципов представления данных для хранения информации об облаках и ландшафтах:

1) использование регулярной сетки высот (Карта высот - HeightMap);
2) использование иррегулярной сетки вершин и связей, их соединяющих (т.е. хранение простой триангулизированной карты);
3) посегментная карта высот.

1.1.1 Карта высот

Данные представлены в виде двухмерного массива. Уже заданы две координаты (x, y — по высоте и ширине массива), и третья координата задается значением в конкретной ячейке, это высота. *Преимущества данного принципа:* простота изменения этих самых данных, так как существует множество программ для работы с растровой графикой.

Минусы данного принципа: слишком много описаний для точек, а также, в некоторых случаях, наблюдается избыточность данных.

1.1.2 Иррегулярная сетка

Зачастую такие решения применяются в специализированных пакетах для игр или специальных пакетах для работы с трехмерной графикой. И

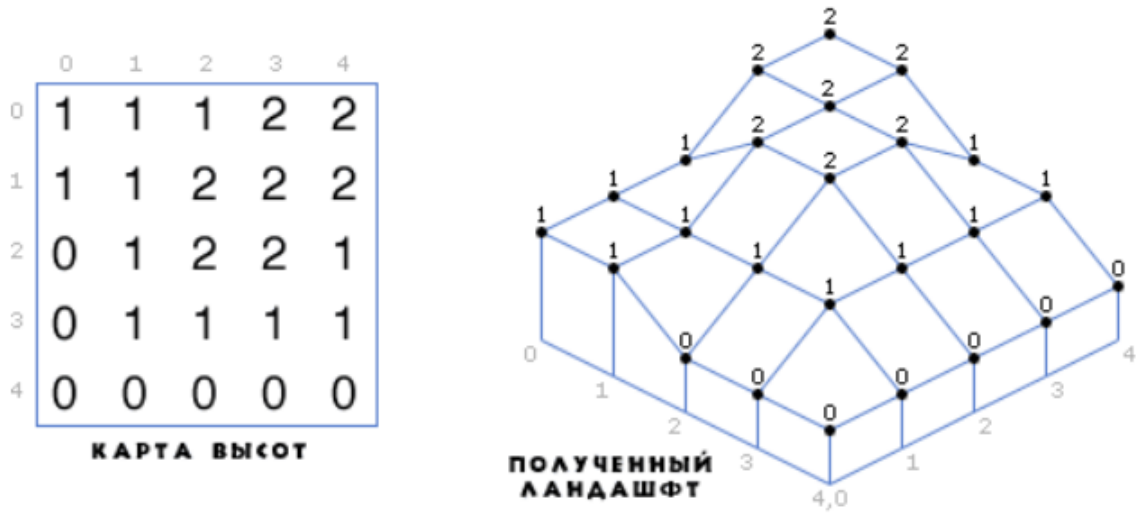

Рисунок 1 – Пример карты высот

хранятся в виде трехмерных моделей. *Преимущества данного принципа:* используется значительно меньше информации для построения ландшафта.

Минусы данного принципа:

1) сложности при динамическом освещении — вершины расположены достаточно далеко друг от друга и неравномерно;

2) хранение, просмотр, модификация такого ландшафта также представляет сложности.

1.1.3 Посегментная карта высот

В данном способе также используются карты высот. Только вместо высот в ней хранятся индексы ландшафтных сегментов. Как эти сегменты представлены, в принципе, роли не играет. Они могут быть и регулярными, и иррегулярными.

Преимущества данного принципа: возможность представления огромнейших открытых пространств.

Минусы данного принципа: проблема стыковки разных сегментов и неочевидность данных.

1.1.4 Вывод

Наиболее эффективным принципом является использование карты высот, так как с помощью этого способа можно представлять обширные пространства, а также можно достаточно просто изменять данные.

1.2 Генерация карты высот

Трехмерное моделирование облаков является более реалистичным и физически точным, а также позволяет визуализировать гораздо большее число явлений, происходящих в облаках. Простым решением является использование шумовых функций для генерации карты высот. «Шумовая» функция – это псевдослучайная функция, зависящая от всех своих параметров. Для генерации карты высот чаще всего используют градиентные шумы, а именно:

1) Perlin Noise;

2) Simplex Noise.

1.2.1 Perlin Noise

Шум Перлина — это градиентный шум, состоящий из набора псевдослучайных единичных векторов (направлений градиента), расположенных в определенных точках пространства и интерполированных функцией сглаживания между этими точками. Для генерации шума Перлина в одномерном пространстве необходимо для каждой точки этого пространства вычислить значение шумовой функции, используя направление градиента (или наклон) в указанной точке.

Алгоритм шума Перлина можно масштабировать одно-, двух- и трёхмерного вида. Более того, в алгоритм можно ввести четвёртое временное измерение, позволяя алгоритму динамически изменять текстуры во времени.

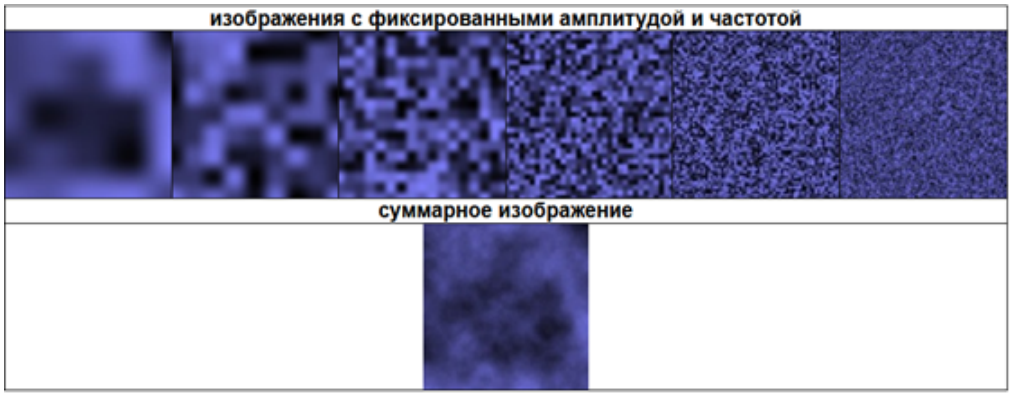

Рисунок 2 – Результат создания изображения с помощью шума Перлина

1.2.2 Simplex Noise

Идея данного алгоритма заключается в том, что нужно использовать сетку из симплексов: в двумерном случае – треугольник, в трехмерном случае – тетраэдр.

Симплексный шум полезен для приложений компьютерной графики, где шум обычно вычисляется по 2, 3, 4 или, возможно, 5 измерениям. Для более высоких измерений n-сферы вокруг n-симплексных углов не упакованы достаточно плотно, что снижает поддержку функции и делает ее нулевой в больших частях пространства.

Рисунок 3 – Абстрактная композиция в 3D, созданная с помощью алгоритма генерации шума

1.2.3 Вывод

В качестве алгоритма для генерации карты высот был выбран шум Перлина, так как облачная поверхность обладает неравномерной структурой, но также не хаотической. Данный алгоритм достаточно прост в реализации.

1.3 Алгоритмы удаления невидимых линий и поверхностей

Существует несколько алгоритмов для решения данной проблемы. Необходимо провести анализ и выбрать самый подходящий для решения поставленной задачи.

1.3.1 Алгоритм Робертса

Алгоритм Робертса представляет собой первое известное решение задачи об удалении невидимых линий. Это математически элегантный 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.

Минусы данного алгоритма: вычислительная трудоемкость алгоритма Робертса растет теоретически как квадрат числа объектов. Это в сочетании с ростом интереса к растровым дисплеям, работающим в пространстве изображения, привело к снижению интереса к алгоритму Робертса.

Преимущества данного алгоритма: математические методы, используемые в этом алгоритме, просты, мощны и точны.

1.3.2 Алгоритм Варнока

Алгоритм Варнока решает задачу удаление невидимых ребер и поверхностей в пространстве изображений. Идеей данного является разбиение окна

на подокна, пока для каждого окна не сможем сказать, что в нем изобразить. Большая часть времени и труда затрачивается на области с высоким информационным содержимым.

1.3.3 Алгоритм трассировки лучей

В этом методе для каждого пикселя картинной плоскости определяется ближайшая к нему грань, для чего через этот пиксель выпускается луч, находятся все его пересечения с гранями и среди них выбирается ближайшая.

Преимущества данного алгоритма:

1) возможность рендеринга гладких объектов без аппроксимации их полигональными поверхностями (например, треугольниками);

2) вычислительная сложность метода слабо зависит от сложности сцены;

3) высокая алгоритмическая распараллеливаемость вычислений — можно параллельно и независимо трассировать два и более лучей, разделять участки (зоны экрана) для трассирования на разных узлах кластера и т. д.;

4) отсечение невидимых поверхностей, перспектива и корректное изменения поля зрения являются логическим следствием алгоритма.

Минусы данного алгоритма: Серьёзным недостатком метода обратного трассирования является производительность. Метод трассирования лучей каждый раз начинает процесс определения цвета пикселя заново, рассматривая каждый луч наблюдения в отдельности.

1.3.4 Алгоритм, использующий Z-буфер

Это один из простейших алгоритмов удаления невидимых поверхностей. Работает этот алгоритм в пространстве изображений. Идея z-буфера

является простым обобщением о буфере кадра. Буфер кадра используется для запоминания атрибутов (интенсивности) каждого пиксела в пространстве изображения. Z-буфер – это отдельный буфер глубины, используемый для запоминания координаты z или глубины каждого видимого пиксела в пространстве изображения. По сути, алгоритм является поиском по x и y наибольшего значения функции z (x, y).

Преимущества данного алгоритма: главное преимущество алгоритма – его простота. Кроме того, этот алгоритм решает задачу об удалении невидимых поверхностей и делает тривиальной визуализацию пересечений сложных поверхностей. Сцены могут быть любой сложности. Поскольку, габариты пространства фиксированы, оценка вычислительной трудоемкости алгоритма не более чем линейна.

Минусы данного алгоритма: основной недостаток алгоритма – большой объем требуемой памяти. Если сцена подвергается видовому преобразованию и отсекается до фиксированного диапазона координат z значений, то можно использовать z-буфер с фиксированной точностью.

1.3.5 Вывод

Так как в сцене не происходит преломление и отражение света, достаточным будет использовать алгоритм z-буфера для удаления невидимых линий и поверхностей.

1.4 Алгоритмы закрасок

1.4.1 Модель освещения Ламберта

Модель Ламберта является одной из самых простых моделей освещение. Считается, что свет, падающий в точку, одинакового рассеивается по всем направлениям полупространства. Таким образом, освещенность в точке определяется только плотностью света в точке поверхности, а она линейно

зависит от косинуса угла падения. При этом положение наблюдателя не имеет значение, т. к. диффузно отраженный свет рассеивается равномерно по всем направлениям.

Формула расчета интенсивности будет выглядеть:

$$I = I_0 \cos \Theta \tag{1}$$

Где I - интенсивность отраженного света, I_0 - интенсивность точечного источника, Θ - угол между направлением света и нормалью к поверхности.

1.4.2 Затенение по Гуро

Метод закраски, который основан на интерполяции интенсивности и известен как метод Гуро, позволяет устранить дискретность изменения интенсивности.

Закраска происходит в четыре этапа:

1) вычисляются нормали ко всем полигонам;

2) определяются нормали в вершинах путем усреднения нормалей по всем полигональным граням, которым принадлежит вершина;

3) используя нормали в вершинах и применяя произвольный метод закраски, вычисляются значения интенсивности в вершинах;

4) каждый многоугольник закрашивается путем линейной интерполяции значений интенсивностей в вершинах сначала вдоль каждого ребра, а затем и между ребрами вдоль каждой сканирующей строки.

1.4.3 Закраска по Фонгу

В методе закраски, разработанном Фонгом, используется интерполяция вектора нормали к поверхности вдоль видимого интервала на сканирующей

строке внутри многоугольника, а не интерполяция интенсивности. Интерполяция выполняется между начальной и конечной нормалями, которые сами тоже являются результатами интерполяции вдоль ребер многоугольника между нормалями в вершинах. Нормали в вершинах, в свою очередь, вычисляются так же, как в методе закраски, построенном на основе интерполяции интенсивности.

1.4.4 Вывод

Из выбранных закрасок была выбрана модель освещения Ламберта из-за простоты ее реализации.

1.5 Постановка задачи

Разработать программное обеспечение для генерации трехмерного изображения облаков и ландшафта. Использовать способ построения карты высот, для ее генерации использовать шум перлина. Дать возможность пользователю задавать необходимые данные для изменения результата генерации облаков и ландшафта. Также предоставить возможность поворота, масштабирования и сдвига полученного изображения. В качестве алгоритма удаления невидимых линий использовать Z-буффер, а для закраски полученного изображения использовать модель освещения Ламберта.

1.6 Вывод

В данном разделе были рассмотрены и проанализированы существующие способы представления данных об облаке и ландшафте, алгоритмы генерации карты высот, удаления невидимых линий и поверхностей и алгоритмы закрасок. Также была поставлена задача.

2 Конструкторский раздел

В данном разделе будут рассмотрены требования к программе и алгоритмы решения поставленной задачи.

2.1 Требования к программе

Программа должна предоставлять следующие возможности:

1) генерировать изображение облаков и ландшафта;
2) задавать данные для генерации облаков и ландшафта;
3) поворот полученного изображения;
4) масштабирование полученного изображения;
5) перенос полученного изображения.

2.2 Общий алгоритм решения поставленной задачи

Алгоритм работы программы:

1) установка первоначальных параметров для генерации карты высот облаков;
2) установка первоначальных параметров для генерации карты высот ландшафта;
3) генерация облачной поверхности;
4) генерация ландшафта;
5) преобразование координат (поворот, масштабирование, перенос) объектов сцены;
6) отображение изображения.

2.3 Шум Перлина

2.3.1 Двумерное моделирование

Шум Перлина — алгоритм генерации текстур, который придумал Кен Перлин в 1983 году. Позже он улучшил свой алгоритм, который теперь так и называется — улучшенный шум Перлина. Особенность такой генерации в том, что шум Перлина генерирует случайные числа для любой точки пространства, но случайные значения отличаются незначительно для точек, находящихся недалеко друг от друга.

Для начала всем точкам плоскости с целыми координатами сопоставляется случайный вектор. В оригинальном алгоритме это был любой вектор единичной длины, в улучшенной версии алгоритма должен быть 1 из следующих векторов: $(1, 0), (-1, 0), (0, 1), (0, -1)$.

При генерации значения для точки (x, y) находим ближайшие к ней точки с целочисленными координатами $(x0, y0), (x0, y1), (x1, y0), (x1, y1)$.

Далее для каждой точки считаем скалярное произведение двух векторов: случайного, сгенерированного ранее, и вектора до точки (x, y).

Теперь считаем средневзвешенное значение. В качестве веса используем расстояние до точки (x, y) по каждой из осей.

Особенность этой реализации средневзвешенного в том, что вес дополнительно прогоняется через функцию Fade. Эта функция как бы прижимает значение к одному из краев отрезка $[0, 1]$. Т.е. если X близок к 0, то Fade(X) еще ближе к 0. При этом Fade(0) == 0, Fade(1) == 1, Fade(0.5) == 0.5.

Теперь осталось только вернуть нормализованное значение. Функция Normalize переводит отрезок $[-1, 1]$ в $[0, 1]$.

Данный алгоритм позволяет генерировать вот такие текстуры:

Рисунок 4 – Получившаяся текстура

2.3.2 Трехмерное моделирование

Простым решением будет использование полученной текстуры как карты высот для облачно поверхности. Каждое значение карты высот можно перемножить на заданную величину высоты.

2.4 Алгоритм Z-буфера

1) заполнить буфер кадра фоновым значением интенсивности или цвета;

2) заполнить z-буфер минимальным значением z;

3) преобразовать каждый многоугольник в растровую форму в произвольном порядке;

4) для каждого Пиксел(x, y) в многоугольнике вычислить его глубину $z(x, y)$.

5) сравнить глубину $z(x, y)$ со значением Zбуфер(x, y), хранящимся в z-буфере в этой же позиции;

если $z(x, y) >$ Zбуфер(x, y), то записать атрибут этого многоугольника (интенсивность, цвет и т. п.) в буфер кадра и заменить Zбуфер(x, y) на $z(x, y)$. В противном случае никаких действий не производить;

6) отобразить результат.

2.5 Модель освещения Ламберта

Интенсивность рассчитывается по закону Ламберта:

$$I = I_0 \cos \Theta \tag{2}$$

Где I - интенсивность отраженного света, I_0 - интенсивность точечного источника, Θ - угол между направлением света и нормалью к поверхности.

2.6 Трехмерные преобразования на плоскости

2.6.1 Поворот

Поворот изображения относительно осей X, Y, Z на угол φ:

$$\begin{cases} X = x \\ Y = centerY + \cos\varphi * (y - centerY) - \sin\varphi * z \\ Z = \cos\varphi * z + \sin\varphi * (y - centerY) \end{cases} \tag{3}$$

$$\begin{cases} X = centerX + \cos\varphi * (x - centerX) - \sin\varphi * z \\ Y = y \\ Z = \cos\varphi * z + \sin\varphi * (y - centerX) \end{cases} \tag{4}$$

$$\begin{cases} X = centerX + \cos\varphi * (x - centerX) - \sin\varphi * (y - centerY) \\ Y = centerY + \cos\varphi * (y - centerY) + \sin\varphi * (x - centerX) \\ Z = z \end{cases} \tag{5}$$

2.6.2 Масштабирование

Масштабирование относительно начала координат:

$$\begin{cases} X = x * k_x \\ Y = y * k_y \\ Z = z * k_z \end{cases} \tag{6}$$

2.6.3 Перенос

Перенос точки:

$$\begin{cases} X = x + dx \\ Y = y + dy \\ Z = z + dz \end{cases} \tag{7}$$

2.7 Используемые типы и структуры данных

Необходимо реализовать следующие типы и структуры данных:

1) карта высот - хранит в себе матрицу и алгоритм для ее генерации;
2) источник света - направленный вектор;
3) объект сцены - хранит в себе карту высот и список полигонов;
4) примитивы:
    4.1) вектор - хранит направление по x, y, z;
    4.2) точка - хранит координаты x, y, z;
    4.3) полигон - хранит список вершин и точек внутри этого полигона;
5) сцена - хранит список объектов.

2.8 Вывод

В данном разделе были представлены требования к программе, алгоритм решения поставленной задачи, был описан улучшенный алгоритм Шума Перлина, алгоритм Z-буфера и модель освещения Ламберта. Также были представлены используемые типы и структуры данных.

3 Технологический раздел

В данном будут рассмотрены средства реализации, сведения о модулях программы, структура и состав классов и интерфейс программы.

3.1 Средства реализации

В качестве языка программирования был выбран C#, так как:

1) я знаком с данным языком программирования, имею представление о способах тестирования программы;
2) данный язык программирования объектно-ориентирован.

В качестве среды разработки была выбрана "Visual Studio 2019 так как:

1) я знаком с данной средой разработки;
2) она является бесплатной для студентов;
3) она имеет множество удоств для написания и отладки кода.

3.2 Сведения о модулях программы

Программа состоит из:

Program.cs - главный файл программы, в котором располагается точка входа в программу;

Form1.cs - интерфейс программы;

Color.cs - смешивание цветов;

Zbuffer.cs - алгоритм Z-буфера;

Map.cs - карта высот;

PerlinNoise.cs - алгоритм Шума Перлина;

Light.cs - источник света;

Moving.cs - сдвиг точек;

Rotating.cs - поворот точек относительно осей;

Scaling.cs - масштабирование точек относительно начала координат;

Object.cs - абстрактный класс объекта;

Shape.cs - форма;

FloatVector.cs - вектор с координатами x, y;

PointInt.cs - точка с координатами x, y, z;

Polygon.cs - полигон;

Vector.cs - вектор с координатами x, y, z;

Scene.cs - сцена.

3.3 Структура и состав классов

На рисунке 5 изображены классы примитивов, а именно: FloatVector, Vector, Polygon, PointInt.

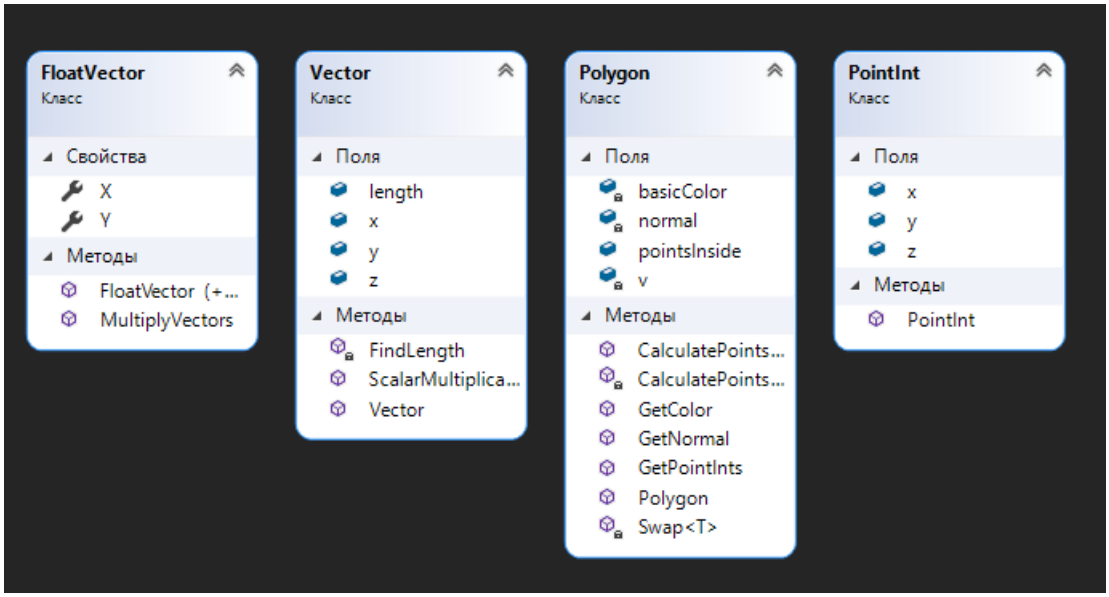

Рисунок 5 – Примитивы

На рисунке 6 изображены статические классы для преобразования координат и классы, содержащие информацию о необходимых преобразованиях.

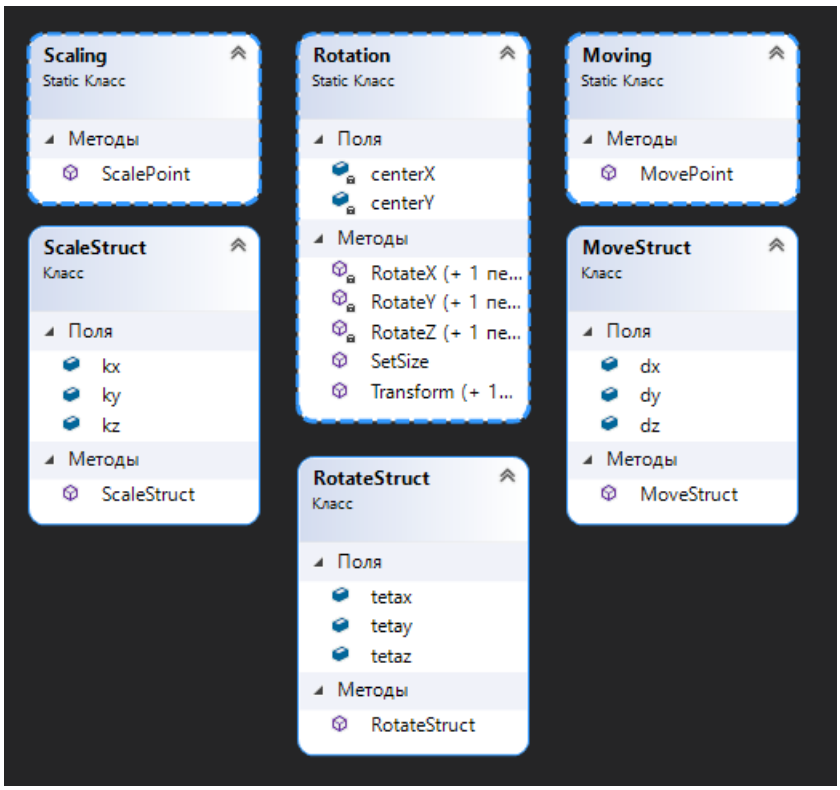

Рисунок 6 – Преобразования координат

На рисунке 7 класс содержащий карту высот (матрицу) и класс алгоритма Шума Перлина для генерации карты высот.

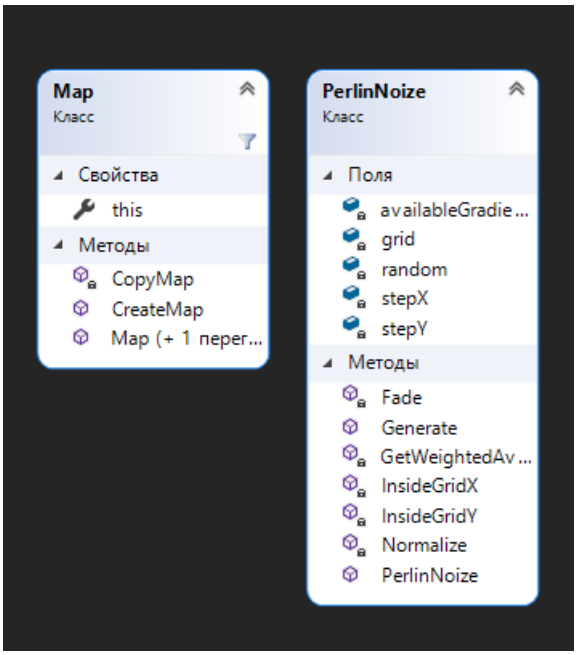

Рисунок 7 – Карта высот и Шум Перлина

На рисунке 8 представлены классы алгоритма Z-буфера, Colors для смешивания цветов и LightSource для определения источника света.

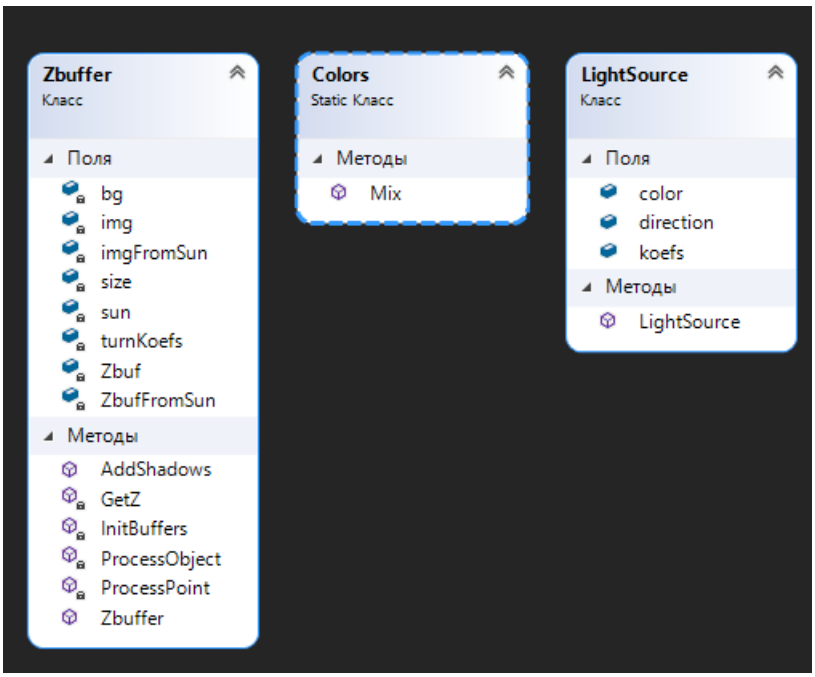

Рисунок 8 – Z-буфер, Colors, LightSource

На рисунке 9 представлены абстрактный класс объекта, класс формы, который хранит список полигонов и класс сцена, который хранит список объектов.

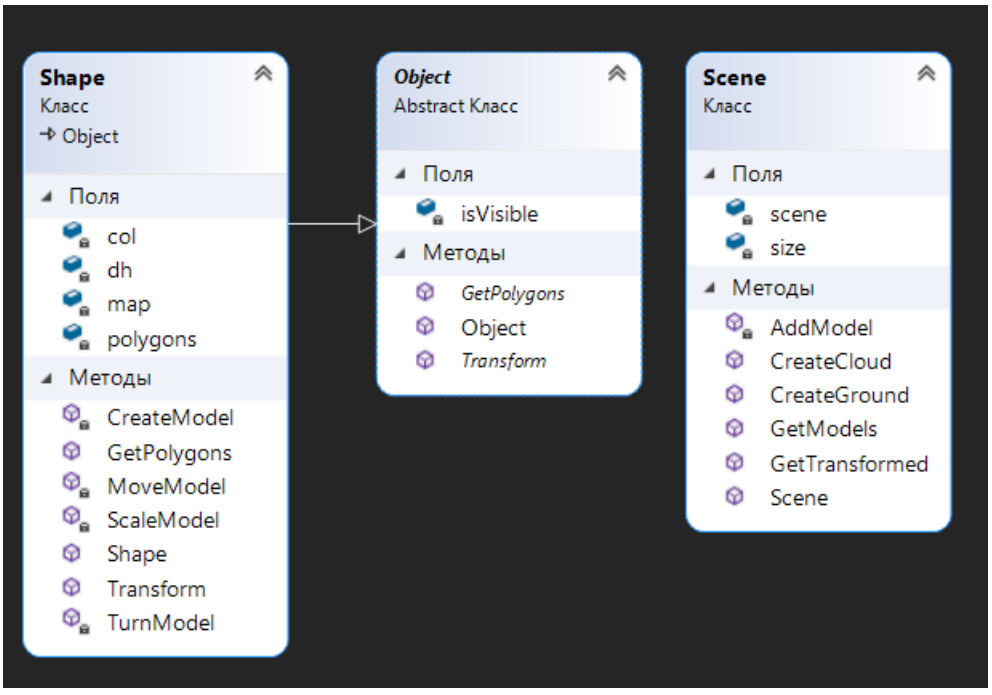

Рисунок 9 – Object, Shape, Scene

3.4 Интерфейс программы

Группа "Создание изображения"позволяет сгенерировать изображение облаков и ландшафта. Для создание изображения необходимо ввести данные в группе "Облака"для установки параметров генерации облаков и, если генерировать вместе с землей, в группе "Земля"для установки параметров генерации ландшафта.

Следующие три группы позволяют сдвигать, поворачивать и масштабировать сгенерированное изображение.

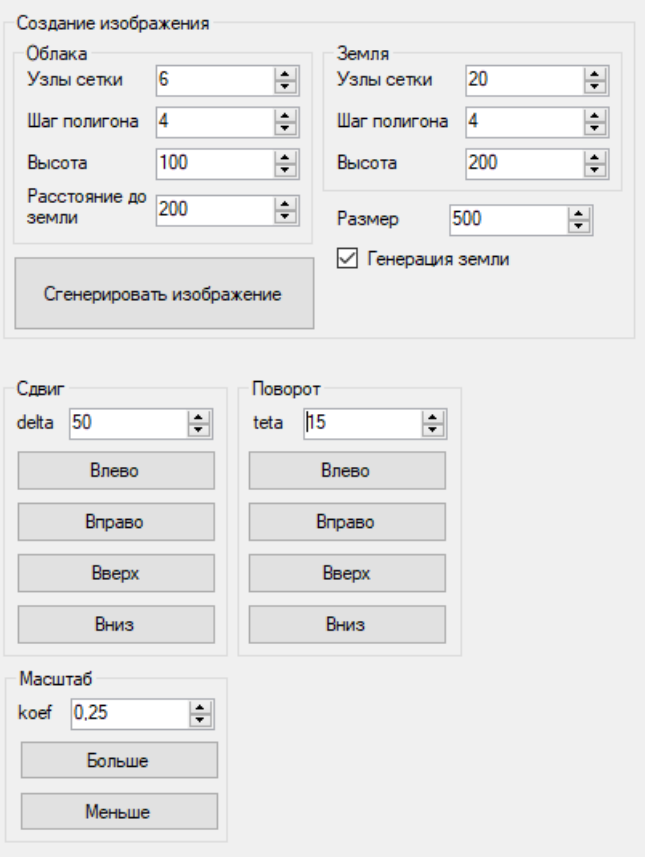

Рисунок 10 – Интерфейс программы

3.5 Вывод

В данном разделе были представлены средства реализации, сведения о модулях программы, структура и состав классов и интерфейс программы.