	<p align="center"> 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) </p>
---	--

ФАКУЛЬТЕТ _____ Информатика и системы управления _____

КАФЕДРА _____ Программное обеспечение ЭВМ и информационные технологии _____

ОТЧЕТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ

Студент _____ Малышев Иван Алексеевич _____
фамилия, имя, отчество

Группа _____ ИУ7-41Б _____

Тип практики _____ Технологическая _____

Название предприятия _____ кафедра _____

Студент	_____	_____ Малышев И. А.
	<i>подпись, дата</i>	<i>фамилия, и.о.</i>

Руководитель практики	_____	_____ Куров А. В.
	<i>подпись, дата</i>	<i>фамилия, и.о.</i>

Оценка _____

2021 г.

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)

Кафедра «Программное обеспечение ЭВМ и информационные технологии» (____ ИУ7 ____)

З А Д А Н И Е

на прохождение производственной практики

на предприятии _____ МГТУ им. Н.Э. Баумана _____

Студент _____ Малышев Иван Алексеевич, И. А., ИУ7-41Б _____
(фамилия, имя, отчество; инициалы; индекс группы)

Во время прохождения производственной практики студент должен сделать следующее.

1. Начать разработку программного обеспечения для генерации трёхмерного ландшафта в дополненной реальности.
2. Решить вопрос о способе генерации и представления данных о ландшафте, проанализировать алгоритмы для их обработки и осуществить выбор алгоритмов.
3. Закрепить знания и навыки, полученные в ходе аудиторных занятий по пройденным курсам.

Дата выдачи задания « ____ » _____ 2021 г.

Руководитель практики от кафедры _____ / _____ Куров А.В.
(подпись, дата) (Фамилия И.О.)

Студент _____ / _____ Малышев И.А.
(подпись, дата) (Фамилия И.О.)

Оглавление

Введение.....	5
1. Аналитическая часть.....	7
1.1. Формализация объектов и преобразований над ними.....	7
1.2. Анализ способов задания трёхмерных моделей	8
1.3. Анализ способов хранения полигональной сетки.....	9
1.4. Анализ способов представления данных о ландшафте.....	9
1.4.1. Карта высот.....	10
1.4.2. Иррегулярная сетка.....	10
1.4.3. Посегментная карта высот	11
1.5. Анализ алгоритмов процедурной генерации ландшафта	12
1.5.1. Алгоритм Diamond-Square	12
1.5.2. Шум Перлина.....	13
1.5.3. Холмовой алгоритм	15
1.6. Анализ алгоритмов удаления невидимых линий и поверхностей	17
1.6.1. Алгоритм Робертса	17
1.6.2. Алгоритм, использующий Z-буфер.....	18
1.6.3. Алгоритм обратной трассировки лучей.....	18
1.7. Анализ методов закрашивания	19
1.7.1. Простая закрашка	20
1.7.2. Закраска Гуро	20
1.7.3. Закраска Фонга	21
1.8. Анализ алгоритмов построения теней.....	22
1.9. Анализ способов реализации технологии дополненной реальности...	23
1.9.1. ArUco	24
1.9.2. AprilTag	24

1.9.3. ARToolKit.....	25
2. Конструкторская часть	27
2.1. Требования к программному обеспечению	27
2.2. Разработка алгоритмов	27
2.2.1. Общее решение.....	28
2.2.2. Наложение текстур	30
2.2.3. Синтез изображения	34
2.2.4. Построение теней.....	35
2.3. Используемые типы и структуры данных	35
3. Технологический раздел.....	38
3.1. Выбор и обоснование средств реализации	38
3.2. Структура и состав классов.....	39
3.3. Разработка и реализация программных модулей.....	41
Заключение	45
Литература	46

Введение

В настоящее время многие компании заинтересованы в технологии дополненной реальности и стараются активно её использовать, так как это очень наглядный, интерактивный метод представления цифровой информации в контексте физического пространства [1]. Дополненная реальность – одна из многих технологий взаимодействия человека и компьютера. Её специфика заключается в том, что она программным образом визуально совмещает два изначально независимых пространства: мир реальных объектов вокруг нас и виртуальный мир, воссозданный на компьютере. Новая виртуальная среда образуется путём наложения запрограммированных виртуальных объектов поверх видеосигнала с камеры, и становится интерактивной путём использования специальных маркеров [2].

Дополненная реальность уже много лет используется в медицине, в образовании, в рекламной отрасли, в ландшафтном дизайне, в военных технологиях, в игровой индустрии, для мониторинга объектов и в мобильных устройствах.

Задачи генерации ландшафта актуальны для создания полностью или частично виртуальных ландшафтов, которые выглядят правдоподобно. Подобные ландшафты используются в фильмах с компьютерной графикой, трёхмерных играх и в качестве демонстрационного материала для различных архитектурных объектов или дизайнерских решений по ландшафту [3].

Процесс моделирования вручную даже с помощью вспомогательного ПО крайне трудоёмкий, неудобный и неблагодарный, по сравнению с моделированием других трёхмерных моделей. Основой виртуального мира должен быть ландшафт значительных размеров, а для реалистичности он должен быть детализированным и разнообразным. Кроме того, чтобы увидеть эту детализацию, нужна система освещения, а значит понадобятся корректные, реалистичные карты нормали и в большом количестве. Очевидно, что создавать всё это вручную нецелесообразно, а результат будет, в лучшем случае, посредственным.

Альтернативой ручного моделирования ландшафта является процедурная генерация с использованием генераторов случайных чисел, что включает в себя построение математической модели ландшафта и реализации алгоритма генерации. Именно этот способ моделирования ландшафта и будет рассмотрен в работе.

Применение технологии дополненной реальности к системам трёхмерного моделирования ландшафта даёт более наглядное представление о виртуальной сцене, объектах на ней и позволяет интерактивно взаимодействовать с ней. Подобная интеграция технологий может быть полезна для более доступной демонстрации архитектурных объектов или проектов ландшафтных дизайнеров: достаточно будет одного смартфона, чтобы самому увидеть планируемое.

Целью работы является проектирование программного обеспечения для генерации трёхмерного ландшафта и отображения его в дополненной реальности. Таким образом, необходимо решить следующие задачи:

1. Формализовать объекты синтезируемой сцены и преобразования над ней;
2. Провести анализ существующих алгоритмов синтеза ландшафта и отображения виртуальной сцены в дополненной реальности, обосновать оптимальность выбранных алгоритмов;
3. Реализовать выбранные алгоритмы;
4. Разработать программный продукт для визуализации и преобразования виртуальной сцены в дополненной реальности.

1. Аналитическая часть

1.1. Формализация объектов и преобразований над ними

Сцена содержит следующие объекты:

- Модель ландшафта – трёхмерный объект, состоящий из следующих частей:
 - Полигональная сетка – совокупность связанных между собой плоских многоугольников [4].
 - Текстуры – набор растровых изображений, накладываемые на поверхность полигональной модели для придания ей цвета, окраски или иллюзии рельефа [3].
- Источник света – точка пространства, подобная точке положения наблюдателя. Принимает ортогональную проекцию визуализируемой сцены из своего положения с некоторым ограниченным обзором. В зависимости от расположения источника и направления распространения лучей света, определяет тень от объектов, расположенных на сцене.

На сцене видна лишь часть модели ландшафта, размер которой задаётся пользователем, для уменьшения нагрузки на ЭВМ. Это и есть видимая часть ландшафта. Для такого способа отображения модели нужно формализовать преобразования над ней, чтобы устранить неоднозначности. Согласно условию задачи, для модели существует три типа преобразований:

1. *Поворот*. Задаётся только один угол поворота. Совершается вокруг оси, перпендикулярной плоскости основания, то есть поворот совершается только в одной плоскости. Никак иначе модель повернуть нельзя. Центр вращения — барицентр части модели.
2. *Масштабирование*. Задаётся тремя вещественными числами больше нуля для каждой оси. Центр масштабирования такой же, как и центр вращения.
3. *Перемещение*. Задаётся двумя целыми числами по осям, параллельные плоскости основания модели. Перемещение представляет собой перемещение границ видимой части модели ландшафта, поэтому

перемещение по оси, перпендикулярной плоскости основания модели, не предусматривается.

1.2. Анализ способов задания трёхмерных моделей

Модели являются отображением формы и размеров объектов. Основное назначение модели – правильно отображать форму и размеры определённого объекта [4].

В основном используются следующие три формы моделей:

- 1) Каркасная (проволочная) модель. В данной модели задаётся информация о вершинах и рёбрах объекта. Это одна из простейших форм задания модели, но она имеет один существенный недостаток: модель не всегда однозначно передаёт представление о форме объекта;
- 2) Поверхностные модели. Этот тип модели часто используется в компьютерной графике. Поверхность может описываться аналитически, либо задаваться отдельными её частями. При этом вложенные криволинейные поверхности можно полигонально аппроксимировать: такая поверхность будет задаваться в виде поверхности многогранника. Данная форма имеет свой недостаток: отсутствует информация о том, с какой стороны поверхности находится материал;
- 3) Твёрдотельные (объёмные) модели. Отличие данной формы задания модели от поверхностной формы состоит в том, что в объёмных моделях к информации о поверхностях добавляется информация о том, где расположен материал путём указания направления внутренней нормали.

Из состава модели ландшафта можно сделать вывод, что сама модель будет поверхностной, так как для решения задачи важно правильное восприятие ландшафта, чего не может дать каркасная модель, и при этом для решения задачи не важны физические свойства ландшафта, поэтому твёрдотельное моделирование тоже не подойдёт.

1.3. Анализ способов хранения полигональной сетки

Существует множество различных способов хранения информации о сетке [5]:

- 1) Вершинное представление. Хранятся лишь вершины, которые указывают на другие вершины. Простота представления даёт возможность проводить над сеткой множество операций;
- 2) Список граней. Объект – это множество граней и множество вершин. В каждую грань входят как минимум 3 вершины;
- 3) «Крылатое» представление. Каждая точка ребра указывает на две вершины, две грани и четыре ребра, которые её касаются. Такое представление даёт гибко в динамическом режиме изменять геометрию сетки.

Стоит отметить, что главный фактор при выборе способа хранения полигональной сетки — скорость выполнения преобразований объектов на сцене.

Оптимальным способом является список граней. Такое представление позволяет эффективно преобразовывать модели, так как структура будет включать в себя список вершин. Грани представляются треугольниками, это предоставит возможность описать все требующиеся в программном обеспечении объекты сцены.

1.4. Анализ способов представления данных о ландшафте

Существует несколько основных принципов представления данных для хранения информации о ландшафтах [6]:

- Первый — использование регулярной сетки высот (или ещё другое название Карта Высот — HeightMap).
- Второе — использование иррегулярной сетки вершин и связей, их соединяющих (т.е. хранение простой триангулированной карты).

- Третий — хранение карты ландшафта, но в данном случае хранятся не конкретные высоты, а информация об использованном блоке. В этом случае создаётся некоторое количество заранее построенных сегментов, а на карте указываются только индексы этих сегментов.

1.4.1. Карта высот

В первом способе данные представлены в виде двумерного массива. Индексы массива задают две координаты вершины, X и Y , а третья координата задаётся конкретным значением в данной ячейке, то есть высота точки относительно плоскости XY . Это и есть карта высот. Обычно её представляют в виде монохромного изображения, что даёт нам предел высоты от 0 до 255 [7]. Это позволяет легко вносить изменения и более-менее наглядно просматривать данные. Тогда двумя координатами будет положение конкретного пикселя на картинке, а третья координата - цвет (чем выше значение, прямая зависимость от яркости пикселя — тем больше значение высоты для этой точки). С помощью этого способа можно представить достаточно обширные пространства.

Преимуществами данного подхода являются наглядность данных (информация о форме ландшафта легко представляется в виде изображения) и простота реализации (лёгкость нахождения координат вершины на карте).

Также есть и недостаток: избыточность данных, особенно при построении поверхностей, близкие к плоским.

1.4.2. Иррегулярная сетка

Во втором способе форма ландшафта представляется в виде трёхмерной модели. Это даёт основной выигрыш против карты высот в плане экономии памяти [6]. Но у этого способа есть множество недостатков:

1. Алгоритмы построения ландшафтов в основном предназначены для регулярных карт высот. Оптимизация таких алгоритмов под этот способ потребует значительных усилий;

2. Хранение, просмотр, модификация такого ландшафта также представляет сложности, по сравнению с картой высот.

1.4.3. Посегментная карта высот

В третьем способе также используются карты высот. Только вместо высот в ней хранятся индексы ландшафтных сегментов. Как эти сегменты представлены – не имеет значения. Они могут быть и регулярными, и иррегулярными (причём можно использовать и те и другие одновременно) [6].

Преимуществами такого подхода являются возможность хранить в блоках дополнительные структуры (здания, строения) и особенности ландшафта (пещеры или скалы), а также возможность создания нескольких вариантов одного и того же сегмента, но при разной степени детализации (LOD — Level Of Detail). Но при этом существует проблема стыковки разных сегментов (не понятно, как состыковывать регулярные и иррегулярные сетки), неочевидность данных (взглянув на данные, нельзя моментально представить, как это будет выглядеть в итоге) и проблема модификации (для разных сегментов используются разные инструменты редактирования).

Итог

Проанализировав все способы представления данных о ландшафте, можно прийти к следующим выводам:

- Первый способ для решения задачи является самым простым в обращении и для него уже существует множество алгоритмов построения ландшафтов.
- Второй способ для решения задачи не подойдёт – алгоритмы генерации ландшафта в основном созданы для первого способа, поэтому придётся потратить силы для модификации под второй способ.
- Третий способ для решения задачи тоже не подойдёт, так как возможность содержания иррегулярных сеток влечёт за собой проблемы второго способа.

На основе этого приходим к выбору первого способа, карты высот.

1.5. Анализ алгоритмов процедурной генерации ландшафта

За последние годы в этой области было описано множество методов генерации случайных карт высоты, и большинство из них сводилось к единственному простому условию: создать случайный набор значений и фильтровать его значения до тех пор, пока ландшафт не станет достаточно гладким, то есть смежные элементы высотной карты содержат значения, отличные на некую величину [7].

Главным критерием алгоритма является качество генерируемого ландшафта, так как для решения задачи важно создать правдоподобный рельеф.

Рассмотрим несколько из них.

1.5.1. Алгоритм Diamond-Square

Этот алгоритм основан на алгоритме *midpoint displacement* и имеет всего два шага, «diamond» и «square» [8]:

1. «Diamond»: берётся квадрат из четырёх точек. Вычисляется центральная точка этого квадрата. Её значение вычисляется как среднее значений вершин квадрата с небольшим случайным смещением;
2. «Square»: вычисляются значения точек, лежащие на середине рёбер квадрата как среднее значений центра и вершин квадрата, принадлежащие данным рёбрам;
3. Процесс повторяется с новыми квадрантами до тех пор, пока не установятся все значения карты высот.

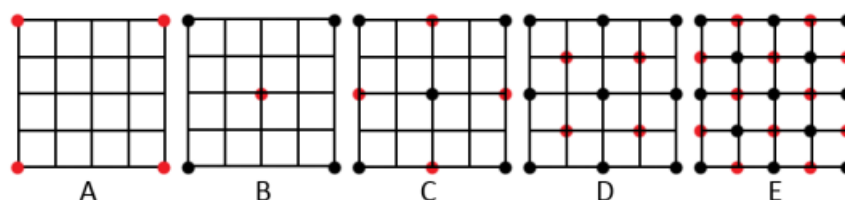


Рис. 1.1. Два этапа работы алгоритма diamond-square

Преимуществами данного алгоритма являются простота и скорость: алгоритм довольно простой с точки зрения реализации и является довольно быстрым по сравнению с другими алгоритмами.

Также есть и недостатки: вероятность появления артефактов на ландшафте, напоминающие всплески, и сложность контролировать рельеф местности.

1.5.2. Шум Перлина

В общем виде является n -мерной функцией шума. В данном разделе будет рассмотрена двухмерная версия шума Перлина.

Данный алгоритм использует два инструмента: шумовую функцию и интерполяцию. Основная идея такова:

I. Задать случайные данные на основе данных ячейки сетки.

Для этого нужно определить сетку поверх карты высот и в каждой точке сетки определить случайный градиент единичной длины, который указывает в случайном направлении в пределах каждого из квадратов. Для каждого пикселя изображения определяется ячейка, где он находится и строится значение высоты, которое основано на векторах узлов этой ячейки и 4-х диагональных векторах, соединяющих углы ячейки с текущим пикселем [7].

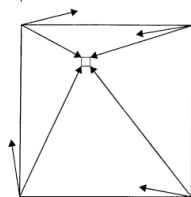


Рис. 1.2. Расчёт значения высоты пикселя при помощи градиентов и векторов, соединяющих точки сетки с расчётной позицией

II. Интерполировать полученные данные для вычисления значения высоты пикселя.

Вычисляем скалярные произведения векторов, лежащие на узлах сетки. Имея четыре значения, нужно объединить их, произвести три смешивания. Для этого нужно вычислить веса смешиваний на основе положения текущего пикселя в ячейке. Веса определяются функцией *smootherstep* (полином $6t^5 - 15t^4 +$

$10t^3$, где вместо t подставляются значения x и y). Далее смешиваются значения скалярных произведений в верхних и нижних углах с помощью линейной интерполяции с использованием веса по x . Эти результаты также смешиваются через линейную интерполяцию, но уже с использованием веса по y . Данный результат и будет высотой пикселя [7].

Чтобы контролировать генерацию шума, существует набор параметров [9]:

- Масштаб(scale) – число пикселей на единицу длины сетки;
- Октавы(octaves) – количество уровней детализации шума;
- Лакунарность(lacunarity) – множитель, который определяет изменение частоты с ростом октавы (по умолчанию равен двум, что соответствует определению октавы);
- Стойкость(persistence) – множитель частотной амплитуды, который определяет изменение амплитуды с ростом октавы.

Для достижения качественных ландшафтов, нужно комбинировать шумы разных масштабов и количества октав, складывая или перемножая их. Таким образом, можно добиться ландшафтов, сочетающие разные типы местности: луга, горы, берега и т. д. [7].

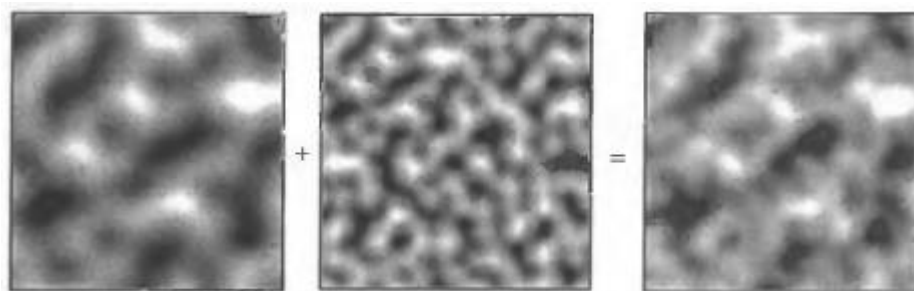


Рис. 1.3. Пример изображения шума Перлина, полученный сложением двух октав.

Преимуществами данного алгоритма являются относительная быстрота и детерминированность: зная лишь параметры генерации, можно получить значение высоты любой точки мира, не зная значений высоты соседних точек. Также благодаря механизму комбинаций октав, можно более тщательно настроить детали ландшафта: без комбинаций октав ландшафт получается однообразным и не детализированным.

1.5.3. Холмовой алгоритм

Это простой итерационный алгоритм, основанный на нескольких входных параметрах [10]. Основная идея алгоритма проста:

- 1) Создаём двухмерный массив и инициализируем его нулевым уровнем (заполняем все ячейки нолями);
- 2) Берём случайную точку на ландшафте или около его границ (за границами), а также берём случайный радиус в заранее заданных пределах. Выбор этих пределов влияет на вид ландшафта — либо он будет пологим, либо скалистым;
- 3) В выбранной точке "поднимаем" холм заданного радиуса;
- 4) Возвращаемся ко второму шагу и так далее до выбранного количества шагов. От него потом будет зависеть внешний вид нашего ландшафта;
- 5) Проводим нормализацию ландшафта;
- 6) Проводим "долинизацию" ландшафта.

Поднять холм - это создать параболоид вращения на сетке ландшафта. Выбранный радиус задаёт радиус основания холма, а квадрат выбранного радиуса – высоту холма. Уравнение «холма» выглядит так:

$$z = R^2 - ((x - x_c)^2 + (y - y_c)^2)$$

Здесь (x_c, y_c) - заданная точка (центр холма), R - выбранный радиус, z – высота точки, принадлежащая холму.

Чтобы сгенерировать ландшафт полностью, нужно построить множество таких холмов. Но есть ещё две вещи на которые необходимо обратить внимание:

1. Целесообразно игнорировать отрицательные значения высоты холма.
2. При генерации последующих холмов лучше добавлять полученное значение для данного холма к уже существующим значениям. Это позволяет построить более правдоподобный ландшафт, нежели правильно очерченные округлые холмы.

Полученный ландшафт будет иметь слишком крутые горы и слишком мало равнин, поэтому нужен этап долинизации. Долинизация – процесс возведения в

квадрат каждого значения высоты [10]. Это эффективно снизит значения высоты среднего диапазона, не слишком сильно влияя на максимумы. Таким образом, мы сгладим местность, не сглаживая горы.

На следующем изображении показано, как долинизация(выравнивание) влияет на сгенерированный ландшафт:

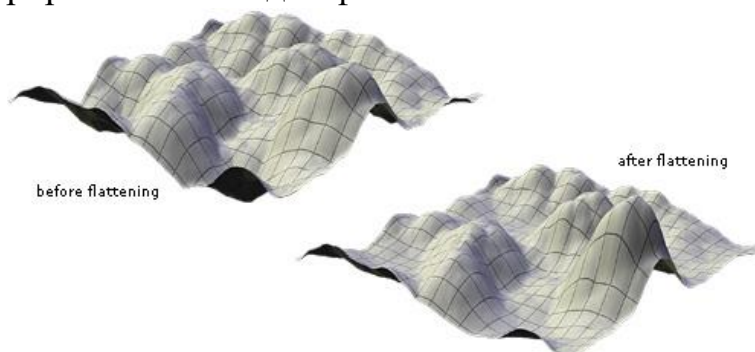


Рис. 1.4. Разница между ландшафтом до долинизации(слева) и после(справа)

Преимуществами данного алгоритма являются возможность контролировать «гористость» ландшафта, получая простые горные и пологие ландшафты, и лёгкая модификация под другие типы ландшафта (одинокий остров, озёрные ландшафты и т. д.), но в некоторых случаях требуется слишком много корректировок для того, чтобы добиться правдоподобности ландшафта.

Итог

Проанализировав несколько алгоритмов генерации ландшафта, можно привести краткую характеристику в виде таблицы 1.1:

Название	Качество ландшафта
Diamond-Square	Среднее
Шум Перлина	Высокое
Холмовой алгоритм	Среднее

Таблица 1.1. Сравнение алгоритмов генерации

Из всех представленных самым подходящим алгоритмом для решения задачи является шум Перлина: он может обеспечить высокое качество

ландшафта за счёт механизма комбинаций разных октав шумов. Это даёт полный контроль над формой рельефа.

1.6. Анализ алгоритмов удаления невидимых линий и поверхностей

В компьютерной графике нет единого алгоритма для удаления невидимых линий, поэтому существует множество таких алгоритмов для разных областей, работающие в объектном пространстве (в мировой системе координат) или в пространстве изображений (в экранных координатах) [11]. Факт работы программы в реальном времени при взаимодействии с пользователем устанавливает главное требование к алгоритму — скорость работы. Рассмотрим алгоритмы для решения задачи.

1.6.1. Алгоритм Робертса

Данный алгоритм работает в объектном пространстве, решая задачу только с выпуклыми телами [11].

Алгоритм выполняется в 4 этапа:

1. Этап подготовки исходных данных (корректные матрицы тел);
2. Этап удаления рёбер, экранируемых самим телом;
3. Этап удаления невидимых рёбер, экранируемых другими телами сцены;
4. Этап удаление линий пересечения тел, экранируемых самими телами, связанными отношением протыкания и другими телами (повтор пункта 2 и 3 для линий протыкания).

Обладая высокой точностью, данный алгоритм имеет ряд недостатков: квадратичную вычислительную сложность алгоритма от числа объектов, потребность делить невыпуклые многогранники на выпуклые и отсутствие возможности работы с прозрачными и просвечивающими объектами.

1.6.2. Алгоритм, использующий Z-буфер

Данный алгоритм является одним из простейших алгоритмов удаления невидимых поверхностей, работает в пространстве изображения [11].

Используется идея буфера кадра. Буфер кадра нужен для заполнения атрибутов (интенсивности) каждого пикселя в пространстве изображения.

Алгоритм использует два буфера: буфер регенерации, в котором хранятся цвета каждого пикселя в пространстве изображения, и z-буфер, куда помещается информация о координате z для каждого пикселя.

Первоначально в z-буфере находятся минимально возможные значения z , а в буфере кадра располагаются пиксели, описывающие фон. Каждый многоугольник преобразуется в растровую форму и записывается в буфер кадра.

В процессе подсчёта глубины нового пикселя, он сравнивается с тем значением, которое уже лежит в z-буфере. Если новый пиксель расположен ближе к наблюдателю, чем предыдущий, то он заносится в буфер кадра и происходит корректировка z-буфера.

Преимуществами данного алгоритма являются простота реализации, линейная вычислительная сложность и отсутствие потребности в сортировке элементов сцены. При этом недостатки выражаются в большом объёме требуемой памяти и сложности реализации эффектов прозрачности.

1.6.3. Алгоритм обратной трассировки лучей

Работает в пространстве изображения.

Отслеживать ход лучей от источника света до объекта и от него до наблюдателя неэффективно, так как лишь малая часть таких лучей дойдёт до наблюдателя, поэтому было предложено отслеживать лучи, проходящие через каждый пиксель пространства изображения, от наблюдателя к объектам [11]. Это и есть обратная трассировка лучей.

Предполагается, что [11]:

- Точка зрения или наблюдатель находится бесконечности на положительной полуоси z , поэтому все лучи параллельны оси z ;

- Каждый луч проходит через центр пикселя на растре до сцены.

Траектория каждого луча отслеживается, чтобы определить, какие именно объекты пересекаются с данным лучом. Необходимо проверить пересечение каждого объекта с каждым лучом. Пересечение с z_{\max} представляет видимую поверхность для данного пикселя.

Преимуществами данного алгоритма являются высокая реалистичность синтезируемого изображения, возможность работы с поверхностями в математической форме и слабая зависимость вычислительной сложности от сложности сцены. При этом сами вычисления являются довольно трудоёмкими.

Итог

Алгоритм Робертса не подходит по следующей причине: модель ландшафта далеко не всегда является выпуклым многогранником. Это потребует многократного деления ландшафта на выпуклые составляющие, что не соответствует требованию к алгоритму по скорости.

Обратная трассировка лучей не отвечает главному требованию – скорости работы. Также от реализуемого продукта не требуется высокой реалистичности синтезируемого изображения и возможности работы с поверхностями, заданными в математической форме, поэтому данный алгоритм не подходит для решения поставленной задачи.

Таким образом, в качестве алгоритма удаления невидимых рёбер и поверхностей был выбран алгоритм с использованием z-буфера.

1.7. Анализ методов закрашивания

Методы закрашивания используются для затенения полигонов (или поверхностей, аппроксимированных полигонами) в условиях сцены, где есть источник света. С учётом взаимного расположения полигона и источника света его уровень освещённости находится по закону Ламберта [11]:

$$I_{\alpha} = I_0 \cdot K \cdot \cos(\alpha)$$

где I_α – уровень освещённости в рассматриваемой точке; I_0 – максимальный уровень освещённости; α — угол между вектором нормали к плоскости и вектором, направленным от рассматриваемой точки к источнику света; K – коэффициент диффузного отражения ($0 \leq K \leq 1$), зависит от длины и волны света, но в простых моделях освещения считается постоянным.

Существует несколько методов закрашивания изображаемых поверхностей. Рассмотрим их.

1.7.1. Простая закрашка

В данном алгоритме вся грань закрашивается одним уровнем интенсивности, который вычисляется по закону Ламберта [12]. Используется при выполнении трёх условий:

1. Источник находится в бесконечности;
2. Наблюдатель находится в бесконечности;
3. Закрашиваемая грань является реально существующей, а не полученной в результате аппроксимации поверхности (в ином случае будут видны рёбра, которых не должно быть, например, на сфере).

Данный алгоритм является простым и быстрым, используется для работы с многогранниками с диффузным отражением, но при этом плохо подходит для фигур вращения: видны рёбра.

1.7.2. Закраска Гуро

Метод Гуро основывается на идее закрашивания каждой грани не одним цветом, а плавно изменяющимися оттенками, вычисляемыми путём интерполяции цветов примыкающих граней [12]. Закрашивание граней по методу Гуро осуществляется в четыре этапа:

1. Вычисляются нормали к поверхности;
2. Определяются нормали в вершинах. Нормаль в вершине определяется усреднением нормалей примыкающих граней;

3. На основе нормалей в вершинах вычисляются значения интенсивностей в вершинах по закону Ламберта;
4. Каждый многоугольник закрашивают путём линейной интерполяции значений интенсивности в вершинах, сначала вдоль каждого ребра, а затем и между рёбрами вдоль каждой сканирующей строки

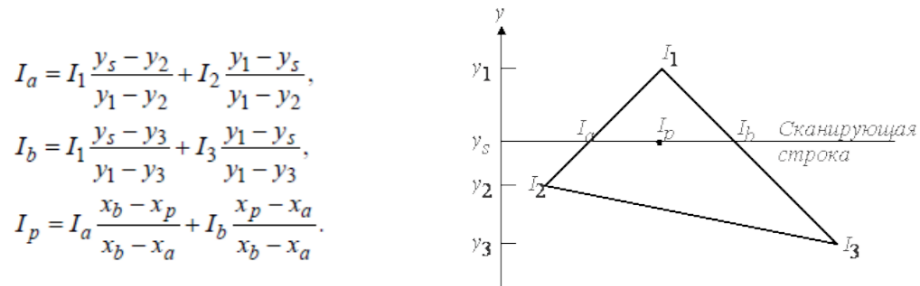


Рис. 1.5. Интерполяция интенсивностей

Для цветных объектов отдельно интерполируется каждая из компонент цвета.

Данный алгоритм используется с фигурами вращения, аппроксимированные полигонами, но при закраске многогранников могут исчезнуть рёбра (случай детской книжки-раскладушки).

1.7.3. Закраска Фонга

В методе закраски, разработанном Фонгом, используется интерполяция вектора нормали к поверхности вдоль видимого интервала на сканирующей строке внутри многоугольника, а не интерполяция интенсивности [12].

Интерполяция выполняется между начальной и конечной нормальями, которые сами тоже являются результатами интерполяции вдоль рёбер многоугольника между нормальями в вершинах. Нормали в вершинах, в свою очередь, вычисляются так же, как в методе закраски, построенном на основе интерполяции интенсивности [12].

Таким образом, закон Ламберта применяется к каждой точке на грани, а не только на вершинах, что позволяет получить более качественное изображение, особенно блики, но с потерей в скорости.

Данный алгоритм используется с фигурами вращения, аппроксимированные полигонами, с зеркальным отражением, но является самым трудоёмким из всех представленных.

Итог

В таблице 1.2 представлена сравнительная характеристика методов закраски. Так как критериями для выбора метода являются скорость, качество изображения и качество работы алгоритма с диффузным отражением (ДО), то по ним же и были оценены методы:

Метод	Скорость	Качество	ДО
Простой	Высокая	Низкое	Высокое
Гуро	Средняя	Среднее	Высокое
Фонга	Низкая	Высокое	Низкое

Таблица 1.2. Сравнение методов закраски

С учётом результатов таблицы 1.2 был выбран метод Гуро.

1.8. Анализ алгоритмов построения теней

В качестве реализуемого алгоритма была выбрана модификация алгоритма с использованием z-буфера путём добавления вычисления теневого z-буфера из точки наблюдения, совпадающей с источником света [11], так как в качестве алгоритма удаления невидимых линий был выбран алгоритм с z-буфером.

Такой подход позволит не усложнять структуру программы, а также избежать проблем адаптации двух различных методов друг к другу, а, следовательно, уменьшить время отладки программного продукта.

Преимущества и недостатки данного подхода точно такие же, как и у алгоритма, на котором он основан.

1.9. Анализ способов реализации технологии дополненной реальности

Основа технологии дополненной реальности – это система оптического трекинга. Это значит, что «глазами» системы становится камера, а «руками» - маркеры. Камера распознает маркеры в реальном мире, «переносит» их в виртуальную среду, накладывает один слой реальности на другой и таким образом создаёт мир дополненной реальности [2].

Существует три основных направления развития технологии:

1. *«Безмаркерная» технология.* Она работает по особым алгоритмам распознавания, где на окружающий ландшафт, снятый камерой, накладывается виртуальная «сетка». На этой сетке программные алгоритмы находят некие опорные точки, по которым определяют точное место, к которому будет «привязана» виртуальная модель.
2. *Технология на базе маркеров.* Технология на базе специальных маркеров, или меток, удобна тем, что они проще распознаются камерой и дают ей более жёсткую привязку к месту для виртуальной модели. Такая технология гораздо надёжнее «безмаркерной» и работает практически без сбоев.
3. *«Пространственная» технология.* Основана на пространственном расположении объекта. В ней используются данные GPS/ГЛОНАСС, гироскопа и компаса, встроенного в мобильный телефон. Место виртуального объекта определяется координатами в пространстве. Активация программы дополненной реальности происходит при совпадении координаты, заложенной в программе, с координатами пользователя.

Стараясь исключить технологические риски и обойти проблемные моменты, при разработке ПО, для решения задачи была выбрана надёжная и проверенная маркерная технология дополненной реальности.

Под маркером понимается объект, расположенный в окружающем пространстве, который находится и анализируется специальным программным обеспечением для последующей отрисовки виртуальных объектов [2]. На основе информации о положении маркера в пространстве, программа может достаточно точно спроецировать на него виртуальный объект, от чего будет достигнут эффект его физического присутствия в окружающем пространстве.

Зачастую в роли маркера выступает лист бумаги с некоторым специальным изображением [13]. Тип рисунка может варьироваться достаточно сильно и зависит от алгоритмов распознавания изображений.

Главные требования к выбору технологии генерации и обнаружения маркеров — лёгкость работы с ней и меньшее количество ложных срабатываний. Рассмотрим существующие технологии для генерации и обнаружения маркеров.

1.9.1. ArUco

ArUco – библиотека с открытым исходным кодом, базирующаяся на библиотеке OpenCV, для обнаружения квадратных фидуциальных маркеров на изображения [14]. При этом она может обнаруживать маркеры других библиотек, таких как: AprilTag и ARToolKit.

Преимуществами ArUco являются лёгкая настройка (с генератором маркеров ArUco) и меньшее количество ложных срабатываний, хотя и более восприимчива к неоднозначности вращения на средних и больших расстояниях.

1.9.2. AprilTag

AprilTag – это визуальная проверочная система, полезная для решения широкого круга задач, включая дополненную реальность, робототехнику и калибровку камеры [15]. AprilTag предназначен для простого включения в другие приложения, а также для переносимости на встроенные устройства. Производительность в реальном времени может быть достигнута даже на процессорах класса сотовых телефонов.

Данная система неплохо работает даже на большом расстоянии, имеет более гибкий дизайн маркеров (например, маркеры не обязательно квадратные) и встроенную поддержку пакетов тегов, в которых несколько тегов объединяются в один тег (использование нескольких тегов с разной ориентацией эффективно устраняет проблему неоднозначности вращения). Хотя и имеет большее количество ложных срабатываний и менее прямолинейна в настройке.

1.9.3. ARToolKit

ARToolKit – это библиотека компьютерного отслеживания с открытым исходным кодом для создания мощных приложений дополненной реальности, которые накладывают виртуальные изображения на реальный мир [16]. Существует с 2004 года и является фундаментальной библиотекой для многих технологий в области AR.

ARToolKit имеет возможность генерации и отслеживания квадратных и естественных маркеров и работы со стереокамерами, но при этом излишне функциональна: данная библиотека предназначена для решения более широкого спектра задач в области AR.

Итог

Все представленные библиотеки для генерации и обнаружения маркеров являются достаточно быстрыми, чтобы работать в реальном времени [14, 15, 16]. Из них для решения задачи была выбрана библиотека ArUco, как самая лёгкая в освоении и с наименьшим количеством ложных срабатываний.

Вывод

В данном разделе были формально описаны объекты на виртуальной сцене и преобразования над ними, были рассмотрены способы задания и хранения моделей и выбрана поверхностная форма задания модели со способом хранения в виде списка граней. Также были рассмотрены способы хранения и алгоритмы

генерации данных ландшафта. В качестве таких были выбраны карта высот и шум Перлина.

Для визуализации ландшафта были рассмотрены алгоритмы удаления невидимых линий и поверхностей, методы закрашивания, алгоритмы построения теней и способы реализации технологии дополненной реальности. В качестве таких были выбраны алгоритм z-буфера, метод Гуро, алгоритм с теневым z-буфером и маркерная технология ArUco.

2. Конструкторская часть

В данном разделе представлены требования к программному обеспечению, общий алгоритм решения задачи, алгоритмы для решения задачи, а также используемые структуры данных.

2.1. Требования к программному обеспечению

Программа должна предоставлять доступ к следующему функционалу:

- 1) Генерация карты высот заданного размера;
- 2) Загрузка карты высот;
- 3) Назначение размера видимой части модели ландшафта;
- 4) Поворот, перемещение и масштабирование всей визуализируемой сцены или отдельно модели. Центром поворота и масштабирования для источника(ов) света в случае преобразования всей сцены является центр модели;
- 5) Изменение положения источника света.

К программе предъявляются следующие требования:

- 1) Время отклика программы должно быть менее 1 секунды для корректной работы в интерактивном режиме;
- 2) Размер видимой части не должен превышать размер карты высот;
- 3) Программа должна корректно реагировать на любые действия пользователя.

2.2. Разработка алгоритмов

В данном подразделе представлены схемы алгоритмов общего решения, генерации карты высот и синтеза изображения. Алгоритм генерации карты высот основан на шуме Перлина [7]. Алгоритм синтеза изображения использует в своей основе алгоритмы z-буфера [11] и метод Гуро [11].

2.2.1. Общее решение

На рисунке 2.1-2.5 представлены схемы алгоритма общего решения.

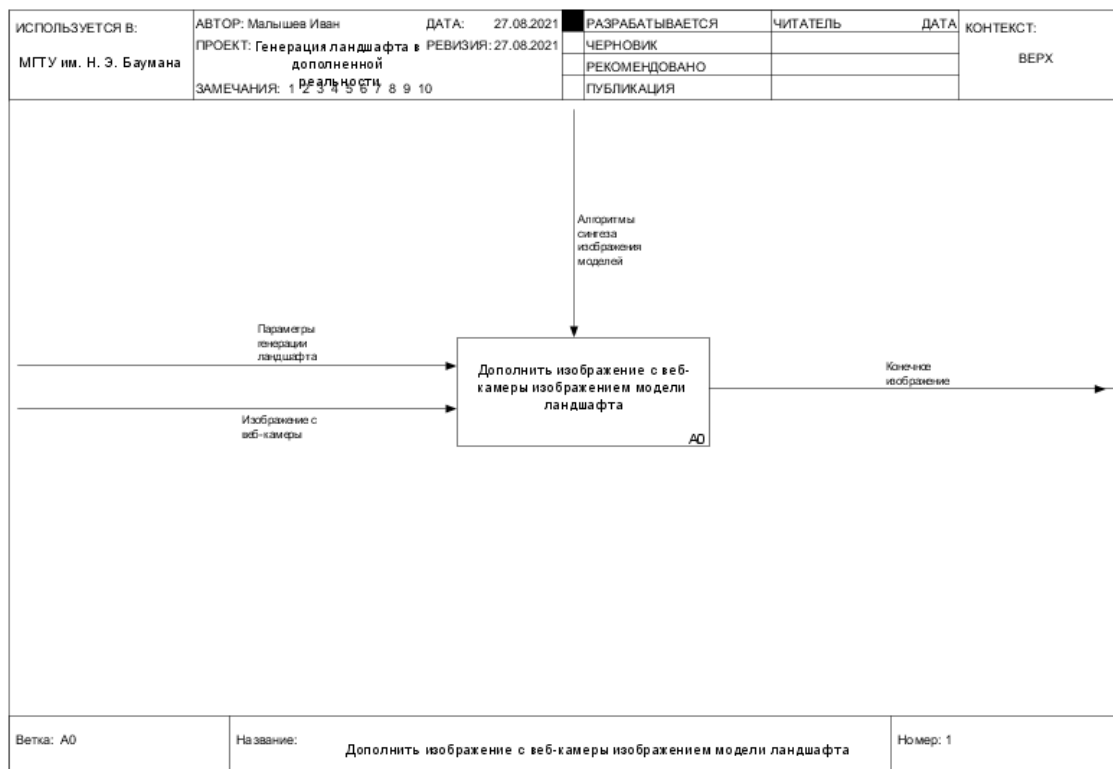


Рис. 2.1. Схема общей постановки задачи

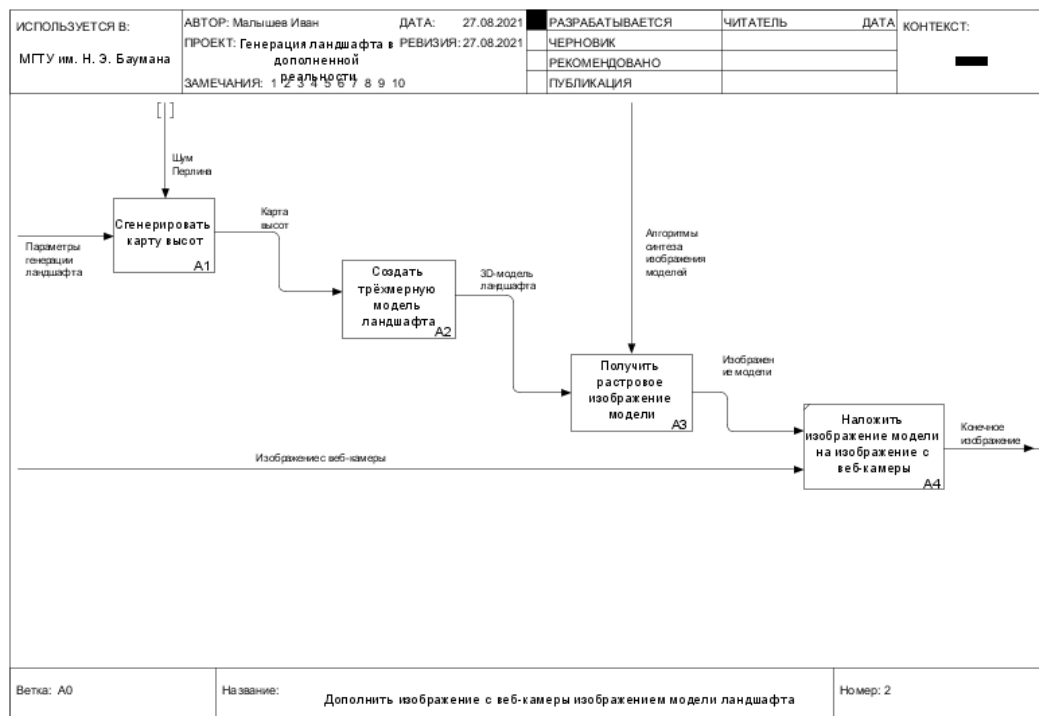


Рис. 2.2. Схема алгоритма синтеза виртуального и реального изображения

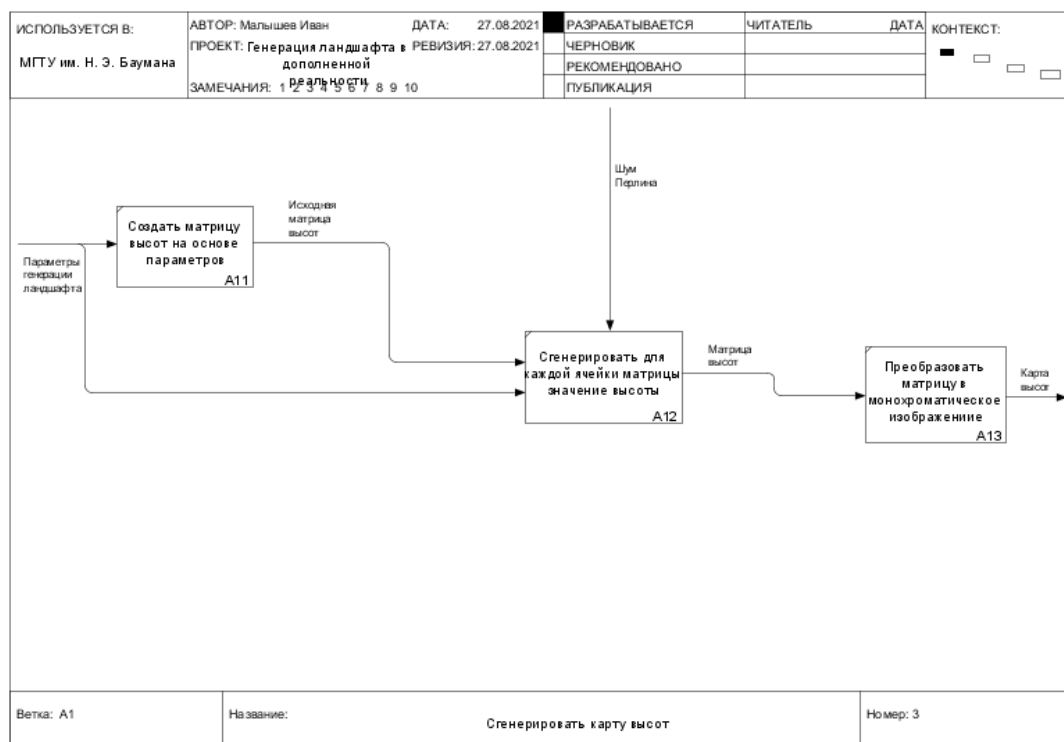


Рис. 2.3. Схема алгоритма генерации карты высот

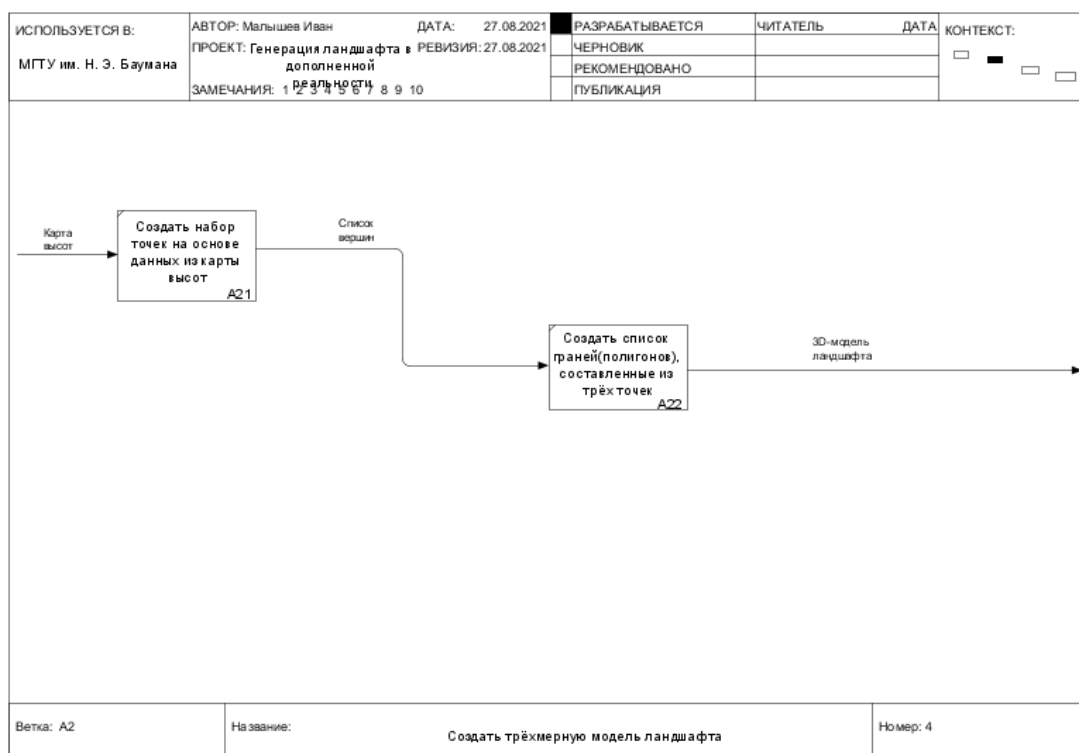


Рис. 2.4. Схема алгоритма создания 3D-модели ландшафта

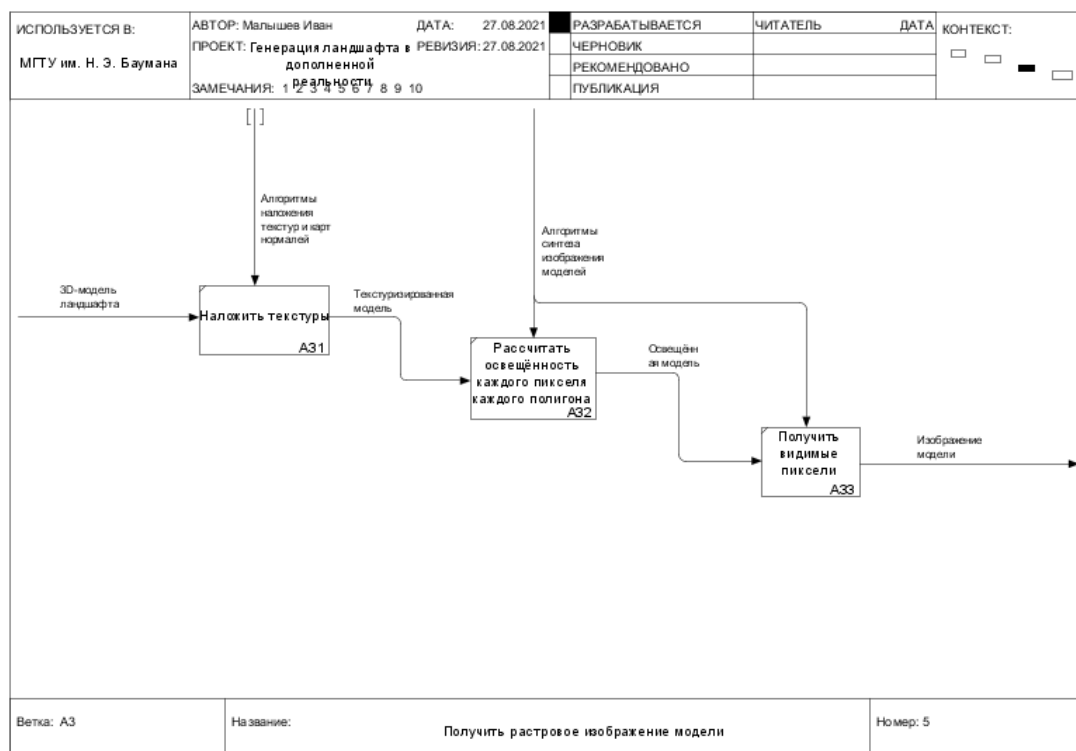


Рис. 2.5. Схема алгоритма растеризации модели

2.2.2. Наложение текстур

Текстуры позволяют увеличить уровень детализации изображения, не добавляя в сцену дополнительную геометрию, и поэтому широко распространены в трёхмерной графике.

Каждая вершина каждой грани трёхмерной модели помимо пространственных координат обладает ещё и *текстурными координатами* – UV-координаты. Это двумерные координаты, принимающие значения от 0 до 1 [17].

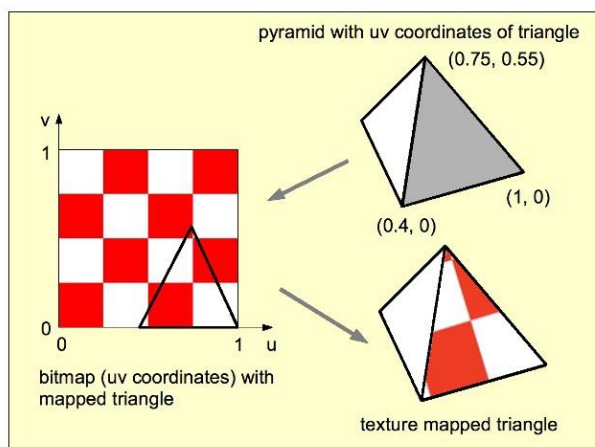


Рис. 2.6. Представление UV-координат

С использованием UV-координат задача наложения текстур сводится к интерполяции текстурных координат U и V по всем точкам грани.

В *аффинном текстурировании* используется линейная интерполяция:

$$u_{\alpha} = (1 - \alpha)u_0 + \alpha u_1, 0 \leq \alpha \leq 1$$

, где u_0 и u_1 – значения текстурной координаты U , заданные на концах некоторого отрезка; $\alpha = \frac{x-x_0}{x_1-x_0}$. Точно такая же формула используется для координаты V , где α вычисляется с использованием координаты y .

Но такой подход не учитывает глубину точки, из-за чего полигоны под углом отображаются неверно, поэтому существует *перспективно-корректное текстурирование*. В нём интерполяция происходит после деления на координату z , а затем используется интерполированное обратное значение для восстановления правильной координаты:

$$u_{\alpha} = \frac{(1 - \alpha) \frac{u_0}{z_0} + \alpha \frac{u_1}{z_1}}{(1 - \alpha) \frac{1}{z_0} + \alpha \frac{1}{z_1}}$$

, где z_0, z_1 – глубины концов отрезка, на котором проводится интерполяция [18].



Рис. 2.7. Методы текстурирования

На рисунке 2.8 представлена схема алгоритма наложения.

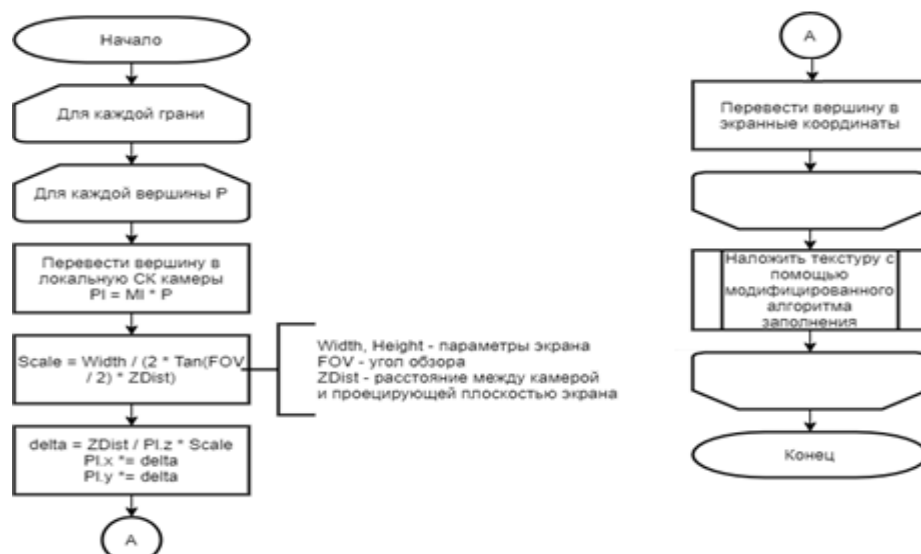


Рис. 2.8. Схема наложения текстур на полигон

Модифицированная версия алгоритма основана на построчном сканировании, как самый простой, быстрый и в случае треугольника не вызывает особых проблем с реализацией [11]. Отличается от оригинальной версии только определением цвета очередного пикселя, который вычисляется следующим образом:

$$1. \quad u_{\alpha} = \frac{(1-\alpha)\frac{u_0}{z_0} + \alpha\frac{u_1}{z_1}}{(1-\alpha)\frac{1}{z_0} + \alpha\frac{1}{z_1}}, \quad \alpha = \frac{x-x_0}{x_1-x_0}$$

$$2. \quad v_{\alpha} = \frac{(1-\alpha)\frac{v_0}{z_0} + \alpha\frac{v_1}{z_1}}{(1-\alpha)\frac{1}{z_0} + \alpha\frac{1}{z_1}}, \quad \alpha = \frac{y-y_0}{y_1-y_0}$$

$$3. \quad x_{pixel} = floor(M_x * u_{\alpha}), y_{pixel} = floor(M_y * v_{\alpha}), \text{ где } M_x, M_y - \text{ ширина и высота текстуры в растре}$$

Часто у полигонов, располагающиеся слишком далеко, текстельная плотность (соотношение количества текселов к пикселям[19]) превышает единицу, то есть на один пиксель «претендует» более одного текселя. В таких случаях появляется *муар* (Рис. 2.9).

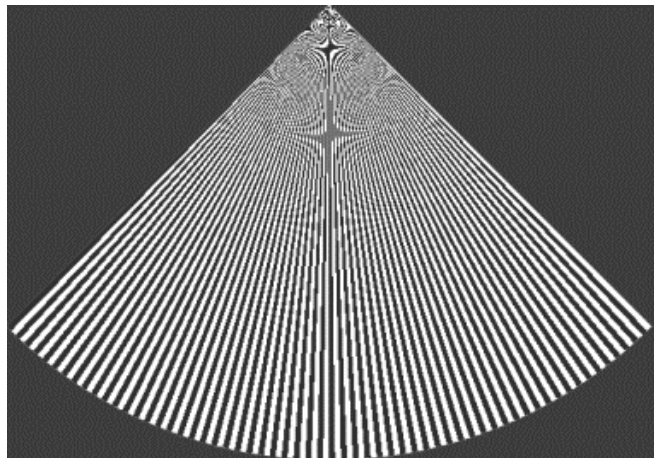
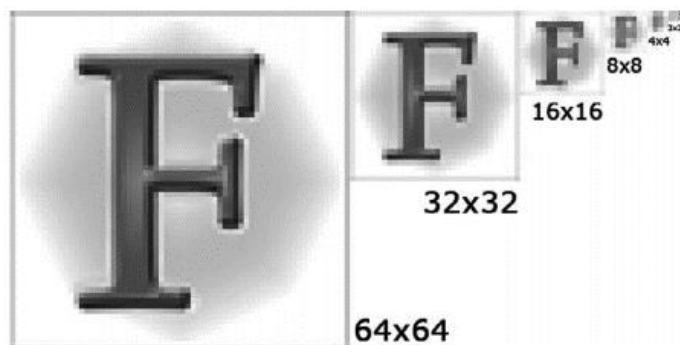


Рис. 2.9. Пример случая появления муара

Для решения этой проблемы существует решение: для дальних объектов использовать ту же текстуру, но с меньшей детализацией. Этот метод называется *MIP-текстурирование*. Метод заключается в следующем: для каждой текстуры создаётся ряд растровых изображений с последовательно уменьшающимся разрешением, где каждое следующее изображение в два раза меньше предыдущего по ширине и высоте. Каждое из этих изображений называется *MIP-*

картой или MIP-уровнем. Так, например, текстура 64×64 будет иметь MIP-уровни со следующими размерами: 64×64 , 32×32 , 16×16 , 8×8 , 4×4 , 2×2 и



1×1 (Рис. 2.10) [19].

Рис. 2.10. MIP-уровни для текстуры размером 64×64

Для вычисления MIP-уровня используется следующая формула:

$$miplevel = mipcount - \lceil \log_4((x_{max} - x_{min}) * (y_{max} - y_{min})) \rceil - 1$$

, где x_{min} , x_{max} , y_{min} , y_{max} – минимальные и максимальные координаты x и y полигона в растре; $mipcount$ – общее число мип-уровней.

При рендеринге пикселя текстурируемого треугольника бывает ситуация, когда в качестве пикселя конечного изображения претендует не один тексель. Поэтому для получения цвета фактически конечного тексела обычно выбирается несколько текселов и смешиваются полученные цвета. Это называется *фильтрацией текстур* [19].

Фильтрация текстур решает проблему резкого перехода между мип-уровнями, используя *трилинейную фильтрацию*, в основе которой *билинейная фильтрация*.

- *Билинейная фильтрация*. При этом подходе вокруг центра пиксела отбираются четыре тексела и результирующий цвет вычисляется как средневзвешенное значение их цветов, где весовые коэффициенты зависят

от удалённости центров текселов от центра пиксела. При использовании MIP-текстурирования выбирается ближайший MIP-уровень.

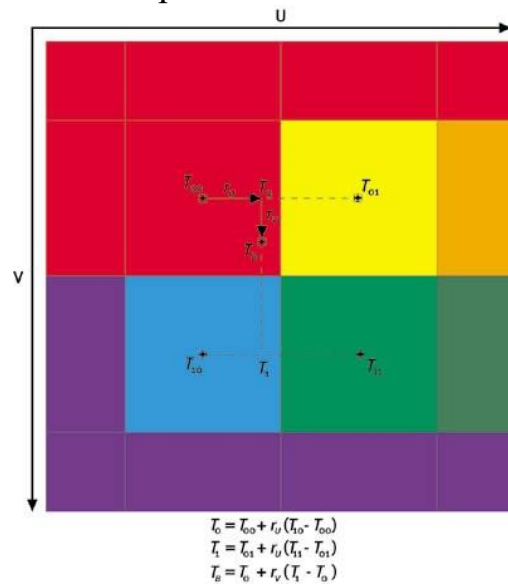


Рис. 2.11. Механизм билинейной фильтрации

- *Трилинейная фильтрация.* При этом подходе производится билинейная фильтрация сразу на двух ближайших MIP-уровнях (уровнях с более высоким и текущем разрешении) с последующей линейной интерполяцией полученных результатов. Это позволяет устранить резкие визуальные границы между MIP-уровнями на экране.

2.2.3. Синтез изображения

На рисунке 2.12 представлена схема алгоритма синтеза изображения.

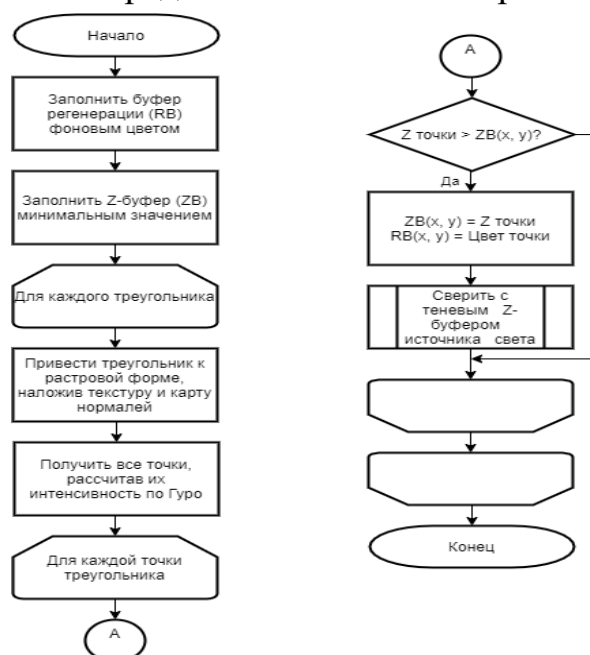


Рис. 2.12. Схема синтеза реалистичного изображения

2.2.4. Построение теней

На рисунке 2.13 представлена схема алгоритма построения теней.

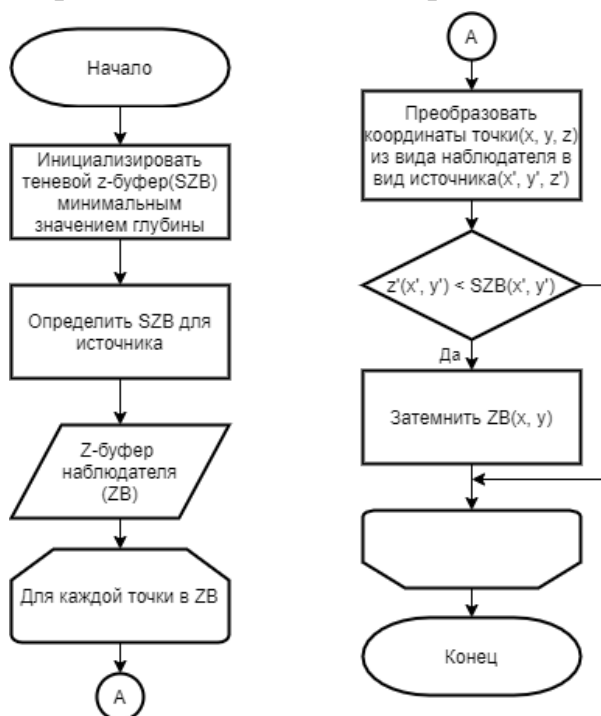


Рис. 2.13. Схема алгоритма построения теней

2.3. Используемые типы и структуры данных

Для реализации программного обеспечения потребуется реализовать типы и структуры данных, представленные в Таблице 2.1.

Данные	Представление
Точка трёхмерного пространства	Координаты X, Y, Z
Вектор	Точка трёхмерного пространства
Мировая система координат (МСК)	Точка трёхмерного пространства с тремя ортонормированными векторами
Локальная система координат (ЛСК)	Точка трёхмерного пространства относительно МСК с тремя ортонормированными векторами
Вершина	Точка трёхмерного пространства; текстурные координаты U, V; нормаль вершины

Полигон	Массив из трёх индексов списка вершин, нормаль поверхности
Модель ландшафта	Список вершин, полигонов, список текстур; пространство модели (ЛСК)
Текстура	Растровое изображение формата .png или .bmp
Карта высот	Растровое черно-белое изображение формата .bmp
Камера	Пространство обзора (ЛКС), ширина и высота виртуального экрана W и H ; отступы от ЛКС по оси Z передней и задней плоскости N и F
Направленный источник света	Цвет света C и вектор направления L
Пользовательский интерфейс	Библиотечные классы
Дополненная реальность	Классы <code>ArUco</code>

Таблица 2.1. Представление данных в программном обеспечении

Мировая система координат задаётся маркером `ArUco` [14].

Выбор форматов .png и .bmp обоснован популярностью и распространённостью представления растровых изображений [19].

Выбор направленного источника света обоснован схожестью с натуральным освещением ландшафтов. Ландшафт планеты преимущественно освещается некой звездой, например Солнцем, или набором звёзд, где лучи света падают параллельно из-за большой дистанции, от чего освещение получается равномерным. Такой же принцип освещения соблюдается и у выбранного источника света.

Вывод

В данном разделе были представлены требования к программному обеспечению.

На основе теоретических данных, полученных из аналитического раздела, были описаны общий алгоритм решения задачи, алгоритмы наложения текстур с последующей их фильтрацией, а также алгоритмы синтеза изображения и наложения теней.

Для решения задачи были определены и обоснованы выбранные типы и структуры данных.

3. Технологический раздел

В данном разделе представлены средства разработки программного обеспечения, разработка программных модулей, детали реализации и тестирование модулей.

3.1. Выбор и обоснование средств реализации

В качестве языка программирования для реализации программного обеспечения был выбран C# [20]. Выбор этого языка обусловлен тем, что он обладает удобным синтаксисом, управляемым кодом и сборщиком мусора, благодаря этому не нужно заботиться об утечках памяти, об указателях и о некоторых базовых структурах и алгоритмах – все это уже реализовано. Это позволит ускорить разработку и отладку кода. Также данный язык предоставляет большую часть требуемого функционала для решения поставленной задачи, для недостающего функционала существует связанным с ним пакетным менеджер NuGet [21].

Для разработки пользовательского интерфейса программного обеспечения была выбрана платформа WPF [22]. Данная платформа обладает богатым ассортиментом объектов, среди которых существуют те, позволяющие работать напрямую с пикселями изображения. Также она обладает декларативным определением элементов интерфейса с помощью языка разметки XAML [23], независимостью от разрешения экрана. Это значит, что приложение будет корректно масштабироваться под разные экраны с разным разрешением, а также данный интерфейс не будет жёстко зависеть от логики программы.

Для реализации технологии дополненной реальности была выбрана обёртка библиотеки OpenCV [24] для C# Emgu [25]. Она позволяет вызывать функции библиотеки OpenCV, написанной на C/C++, с помощью языков .Net [26] (C#, F#, VB.Net).

Чтобы уменьшить нагрузки на процессор при расчёте конечного изображения и ускорить приложение, была выбрана библиотека managedCuda

[27]. Данная библиотека позволяет выполнять некоторые расчёты на CUDA-ядрах [28], то есть использовать мощности видеокарты компании NVIDIA.

В качестве среды разработки (IDE) была выбрана Visual Studio [29], обладающая интеллектуальными подсказками, инструментами анализа, отладки и тестирования кода, поставляющаяся вместе с языком C# и пакетным менеджером NuGet. Также данная IDE обладает большим количеством плагинов под разные инструменты, наподобие GitHub Extension [30], предназначенный для прямой работы с репозиториями через данную IDE. Плагины позволяют облегчить и ускорить разработку программного обеспечения.

3.2. Структура и состав классов

В данном подразделе предоставлена структура и состав классов в программном обеспечении и их описание.

На рисунке 3.1 представлена структура классов в программном обеспечении.

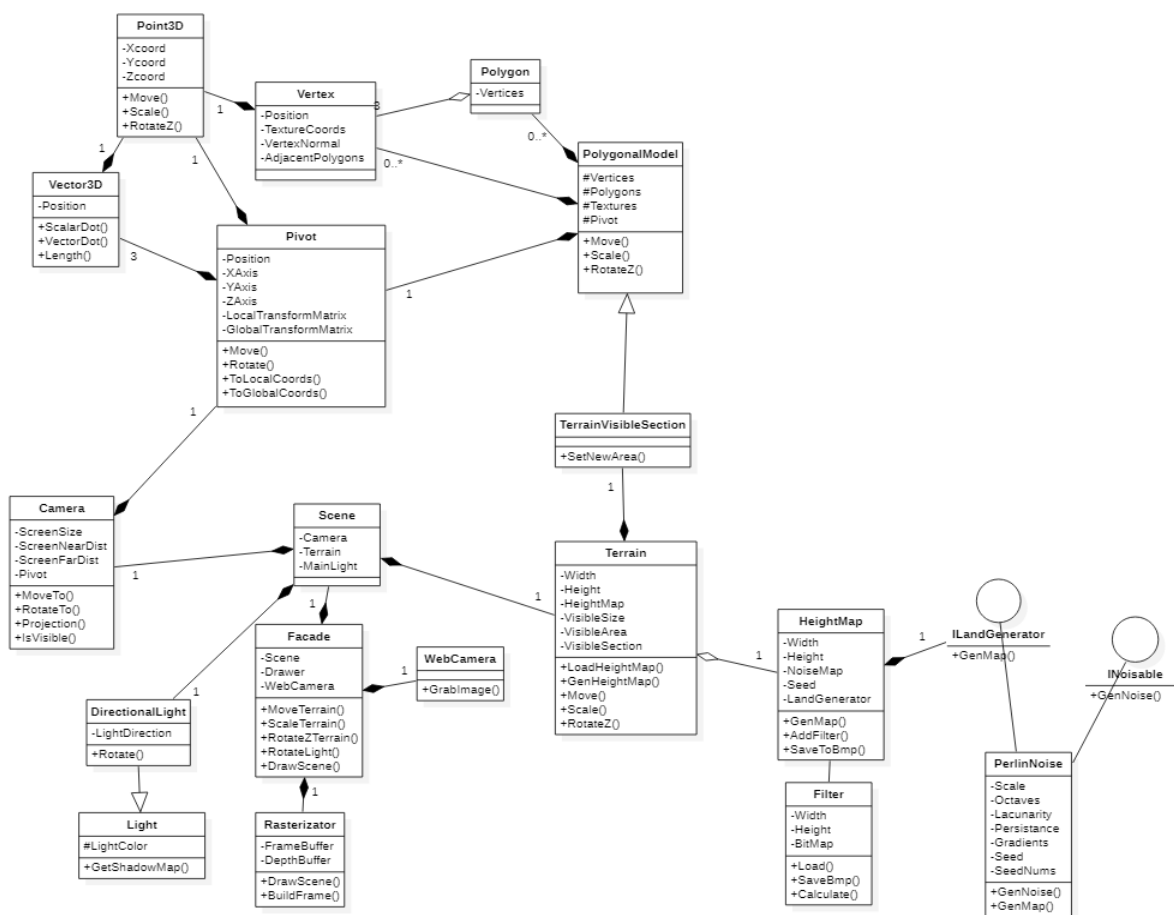


Рис. 3.1. Структура классов всего проекта

Point3D – класс точки трёхмерного пространства. Хранит координаты в пространстве. Владеет методами преобразований точки.

Vector3D – класс вектора трёхмерного пространства. Хранит координаты вектора. Владеет методами вычисления длины, скалярного и векторного произведения.

Vertex – класс вершины. Хранит координаты точки вершины, текстурные координаты, нормаль вершины и номера граней, в которых она задействована, то есть номера соседних граней.

Polygon – класс грани. Хранит номера задействованных в грани вершин.

Pivot – класс локальной системы координат. Хранит положение, три ортонормированных векторов и матрицы преобразований в локальную и мировую систему координат. Владеет методами перемещения, поворота и перевода в локальную и мировую систему координат.

PolygonalModel – класс полигональной модели. Хранит множество вершин и граней, образующих модель, положение, набор текстур и локальную систему координат. Владеет методами перемещения, масштабирования и вращения по оси Z модели.

HeightMap – класс карты высот. Хранит размеры карты, шумовую карту (матрица размером Width на Height, хранящая значения шумовой функции), зерно для инициализации генератора случайных чисел и шумовую функцию. Владеет методами генерации карты, наложения фильтра и сохранения в виде изображения формата .bmp.

Filter – класс изображения, накладываемое на карту высот. Хранит матрицу со значениями альфа-канала и её размеры. Владеет методами загрузки, сохранения в формате .bmp и аналитическим вычислением.

PerlinNoise – класс шума Перлина. Хранит параметры шумовой функции и данные для генерации. Владеет методом вычисления шумовой функции в точке (x, y) и генерации карты высот.

ILandGenerator – интерфейс генератора карты высот. Владеет методом генерации карты высот.

INoisable – интерфейс шумовой функции. Владеет методом генерации шумового значения.

Terrain – класс ландшафта. Хранит размеры ландшафта, карту высот, размер, границы и модель видимой части ландшафта. Владеет методами загрузки и генерации карты высот, преобразований видимой части ландшафта.

TerrainVisibleSection – класс модели видимой части ландшафта. Является потомком класса PolygonalModel, поэтому хранит те же данные и имеет те же методы. Владеет методом изменения видимых границ ландшафта.

Light – класс источника света. Хранит цвет света. Владеет методами вычисления теневой карты.

DirectionalLight – класс направленного источника света. Хранит вектор направления лучей. Владеет методом поворота вектора направления.

Camera – класс камеры. Хранит размеры экрана, отступы передней и задней плоскости, локальную систему координат. Владеет методами перемещения камеры в конкретную точку, поворота в конкретные углы, проекции точки на экранную плоскость и проверки видимости точки.

Scene – класс сцены. Хранит камеру, информацию о ландшафте и направленный источник света.

WebCamera – класс веб-камеры. Владеет методом захвата изображения с веб-камеры (внешнее устройство).

Rasterizator – класс растеризатора. Хранит буфер регенерации и z-буффер. Владеет методом построения буфера кадра и рисования сцены.

Facade – класс фасада. Хранит сцену, растеризатор и веб-камеру. Владеет методами преобразования ландшафта, поворота направленного источника света и рисования сцены.

3.3. Разработка и реализация программных модулей

Для генерации карт высот был разработан модуль HeightMapLib.dll. В листинге 1 представлена реализация модуля.

Листинг 1. Реализация модуля HeightMapLib.dll

```
using System;
using System.Numerics;
using System.Drawing;

namespace HeightMapLib
{
    public delegate float NoiseExpresion(float f);
    public delegate float FilterExpresion(int x, int y);
    public interface ILandGenerator
    {
        int Seed { get; set; }
        float[,] GenMap(int width, int height);
    }

    public interface INoisable
    {
        float GenNoise(int x, int t);
    }

    public class HeightMap
    {
        int width, height;
        NoiseExpresion expresion;

        private float[,] NoiseMap { get; set; }
        ILandGenerator LandGenerator { get; set; }

        public HeightMap(int width, int height, ILandGenerator lg, NoiseExpresion
exp = null)...
        public HeightMap(HeightMap h)...

        public int Width...
        public int Height...
        public int Seed...

        public void GenMap()
        {
            NoiseMap = LandGenerator.GenMap(Width, Height);

            if (expresion != null)
                for (int i = 0; i < width; i++)
                    for (int j = 0; j < height; j++)
                        NoiseMap[i, j] = expresion(NoiseMap[i, j]);
        }

        public float this[int i, int j]
        {
            get => NoiseMap[i, j];
            set => NoiseMap[i, j] = value;
        }

        public static HeightMap Add(HeightMap h1, HeightMap h2)...
        public static HeightMap Subtract(HeightMap h1, HeightMap h2)...
        public static HeightMap MultSingle(HeightMap h, float val)...
        public static HeightMap operator +(HeightMap h1, HeightMap h2) => Add(h1,
h2);
        public static HeightMap operator -(HeightMap h1, HeightMap h2) =>
Subtract(h1, h2);
        public static HeightMap operator *(HeightMap h, float val) =>
MultSingle(h, val);
    }
}
```

```

        private (float, float) MinMax()...
        public void AddFilter(Filter f)...
        public void SaveToBmp(string path, string name)...
    }

    public class Filter
    {
        int width, height;

        public float[,] BitMap { get; private set; }

        public Filter(int width, int height)...

        public int Width...
        public int Height...

        public void Calculate(FilterExpresion exp)...
        public void Load(string fullpath)...
        public void SaveBmp(string path, string name)...
    }

    public class PerlinNoise: ILandGenerator, INoisable
    {
        int scale, octaves, seed;
        float lacunarity, persistence;

        public int Scale...
        public int Octaves...
        public float Lacunarity...
        public float Persistence...
        public Vector2[] Gradients { get; private set; }
        private int[] SeedNums { get; set; }
        public int Seed...

        public PerlinNoise(int scale, int octaves = 1, float lacunarity = 2, float
persistence = 0.5f, int seed = -1)...

        public float GenNoise(int x, int y)
        {
            Vector2 pos = new Vector2((float)x / Scale, (float)y / Scale);

            float x0 = (float)Math.Floor(pos.X);
            float x1 = x0 + 1;
            float y0 = (float)Math.Floor(pos.Y);
            float y1 = y0 + 1;

            Vector2 g0 = GetGradient(x0, y0);
            Vector2 g1 = GetGradient(x1, y0);
            Vector2 g2 = GetGradient(x0, y1);
            Vector2 g3 = GetGradient(x1, y1);

            Vector2 d0 = new Vector2(pos.X - x0, pos.Y - y0);
            Vector2 d1 = new Vector2(pos.X - x1, pos.Y - y0);
            Vector2 d2 = new Vector2(pos.X - x0, pos.Y - y1);
            Vector2 d3 = new Vector2(pos.X - x1, pos.Y - y1);

            float sd0 = Vector2.Dot(g0, d0);
            float sd1 = Vector2.Dot(g1, d1);
            float sd2 = Vector2.Dot(g2, d2);
            float sd3 = Vector2.Dot(g3, d3);

            float sx = SmootherStep(d0.X);
            float sy = SmootherStep(d0.Y);

```

```

        float blendx1 = sd0 + sx * (sd1 - sd0);
        float blendx2 = sd2 + sx * (sd3 - sd2);
        float blendy = blendx1 + sy * (blendx2 - blendx1);

        return blendy;
    }
    public float[,] GenMap(int width, int height)...
    private Vector2 GetGradient(float x, float y)...
    private float SmootherStep(float t) => t * t * t * (6 * t * t - 15 * t +
10);
    }
}

```

Заключение

Во время выполнения практики были рассмотрены и проанализированы основные способы представления модели ландшафта, информации о ландшафте, генерации карты высот, алгоритмы удаления невидимых линий, методы закрашивания, построения теней и технологии дополненной реальности и их реализации. Были проанализированы их достоинства и недостатки, выбраны наиболее подходящие решения поставленной задачи. Было спроектировано программное обеспечение для выполнения данной задачи, а также реализован модуль для генерации карт высот.

Спроектированный программный продукт позволит пользователю загружать или генерировать на основе параметров информацию о ландшафте, просматривать модель ландшафта в интерактивном режиме посредством технологии дополненной реальности. Также пользователю будет позволено изменять характеристики источника света и совершать преобразования над моделью ландшафта.

При проведении работы были получены знания в области компьютерной графики и закреплены навыки проектирования программного обеспечения, а поиск оптимальных решений для эффективной работы программного обеспечения позволил улучшить навыки поиска и анализа информации.

Следует отметить, что проделанная работа позволила лучше изучить как язык программирования C#, так и среду разработки Visual Studio и её расширения.

Литература

1. Дополненная реальность [Электронный ресурс]. – Режим доступа: <https://www.ptc.com/ru/technologies/augmented-reality> (дата обращения 02.07.2021)
2. Технология дополненной реальности AR [Электронный ресурс]. – Режим доступа: https://funreality.ru/technology/augmented_reality/ (дата обращения 02.07.2021)
3. Селянин Н.А., Система трёхмерного моделирования ландшафта: выпускная квалификационная работа — Екатеринбург: Уральский государственный педагогический университет, 2020. - 66 с.
4. Демин А.Ю., Основы компьютерной графики: учебное пособие – Томск: Изд-во Томского политехнического университета, 2011. – 191 с.
5. Набережнов Г. М. Трёхмерное моделирование полигональными сетками [Текст] / Г. М. Набережнов, Н. Н. Максимов // Казань: Казанский Государственный Технический университет им. А.Н.Туполева, 2008. – 14 с.
6. Генерация трехмерных ландшафтов [Электронный ресурс]. – Режим доступа: <https://www.ixbt.com/video/3dterrains-generation.shtml> (дата обращения 05.07.2021)
7. Снук Г. Создание 3D-ландшафтов с использованием C++ и DirectX 9 / Пер. С англ. - М.: КУДИЦ-ОБРАЗ, 2007. - 368 с.
8. Алгоритм «diamond-square» для построения фрактальных ландшафтов [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/post/111538/> (дата обращения 06.07.2021)
9. Libnoise: Glosarry [Электронный ресурс]. – Режим доступа: <http://libnoise.sourceforge.net/glossary/index.html#perlinnoise> (дата обращения 06.07.2021)

10. Terrain Generation Tutorial: Hill Algorithm [Электронный ресурс]. – Режим доступа: <https://www.stuffwithstuff.com/robot-frog/3d/hills/hill.html> (дата обращения 06.07.2021)
11. Роджерс Д. Алгоритмические основы машинной графики: Пер. с англ. - М.: Мир, 1989. - 512 с.
12. Методы закраски [Электронный ресурс]. – Режим доступа: <https://portal.tpu.ru/SHARED/j/JBOLOTOVA/academic/ComputerGraphics> (дата обращения 11.07.2021)
13. Технологии и алгоритмы дополненной реальности [Электронный ресурс]. – Режим доступа: <https://blog.arealidea.ru/articles/mobile/tekhnologii-i-algoritmy-dlya-sozdaniya-dopolnennoy-realnosti/> (дата обращения 12.07.2021)
14. ArUco Library Documentation [Электронный ресурс]. – Режим доступа: <https://docs.google.com/document/d/1QU9KoBtjSM2kF6ITQjQ76xqL7H0TEtXriJX5kwi9Kgc/edit> (дата обращения 12.07.2021)
15. AprilTag Documentation [Электронный ресурс]. – Режим доступа: <https://april.eecs.umich.edu/software/apriltag> (дата обращения 12.07.2021)
16. ARToolKit Documentation [Электронный ресурс]. – Режим доступа: <http://www.hitl.washington.edu/artoolkit/documentation/> (дата обращения 12.07.2021)
17. Реализация схемы наложения текстуры [Электронный ресурс]. – Режим доступа: <http://www.100byte.ru/mxscptxmpls/txtr/txtr.html> (дата обращения 13.07.2021)
18. Perspective Texture mapping [Электронный ресурс]. – Режим доступа: <http://www.lysator.liu.se/~mikaek/doc/perspectivetexture/> (дата обращения 13.07.2021)
19. Игровой движок. Программирование и внутреннее устройство. Третье издание. — СПб.: Питер, 2021. — 1136 с.: ил. — (Серия «Для профессионалов»).

20. Документация по C# [Электронный ресурс]. – Режим доступа: <https://docs.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/> (дата обращения 18.07.2021)
21. Введение в NuGet [Электронный ресурс]. – Режим доступа: <https://docs.microsoft.com/ru-ru/nuget/what-is-nuget> (дата обращения 18.07.2021)
22. Начало работы с WPF [Электронный ресурс]. – Режим доступа: <https://docs.microsoft.com/ru-ru/visualstudio/designers/getting-started-with-wpf?view=vs-2019> (дата обращения 18.07.2021)
23. Обзор XAML (WPF .NET) [Электронный ресурс]. – Режим доступа: <https://docs.microsoft.com/ru-ru/dotnet/desktop/wpf/xaml/?view=netdesktop-5.0&viewFallbackFrom=netdesktop-5.0> (дата обращения 18.07.2021)
24. OpenCV Documentation [Электронный ресурс]. – Режим доступа: <https://docs.opencv.org/master/d1/dfb/intro.html> (дата обращения 18.07.2021)
25. Emgu: Main Page [Электронный ресурс]. – Режим доступа: https://www.emgu.com/wiki/index.php/Main_Page (дата обращения 18.07.2021)
26. Введение в .NET [Электронный ресурс]. – Режим доступа: <https://docs.microsoft.com/ru-ru/dotnet/core/introduction> (дата обращения 18.07.2021)
27. ManagedCuda [Электронный ресурс]. – Режим доступа: <https://kunzmi.github.io/managedCuda/> (дата обращения 18.07.2021)
28. CUDA Zone [Электронный ресурс]. – Режим доступа: <https://developer.nvidia.com/cuda-zone> (дата обращения 18.07.2021)
29. Visual Studio 2019 [Электронный ресурс]. – Режим доступа: <https://visualstudio.microsoft.com/ru/vs/> (дата обращения 18.07.2021)
30. GitHub Extension [Электронный ресурс]. – Режим доступа: <https://visualstudio.github.com/> (дата обращения 18.07.2021)