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_____

ОТЧЕТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ

Студент_____Малышев Иван Алексеевич_____
фамилия, имя, отчество

Группа____ИУ7-41Б____

Тип практики _____Технологическая_____

Название предприятия_____кафедра_____

Студент _____ ____Малышев И. А.____
подпись, дата *фамилия, и.о.*

Руководитель практики _____ ____Куров А. В._____
подпись, дата *фамилия, и.о.*

Оценка _____

2021 г.

Индивидуальное задание

Разработать программу генерации трёхмерного ландшафта в дополненной реальности. Программа должна строить трёхмерную модель ландшафта на основе двумерной карты ландшафта и совмещать построенную модель с видеоизображением, получаемым с веб-камеры. Программа должна предоставлять возможность вращать модель вокруг оси, перпендикулярной плоскости основания модели, перемещать её параллельно этой плоскости, изменяя видимую часть модели, и масштабировать.

Оглавление

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]. 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].
  - Карта нормалей – растровое изображение в формате RGB, каждый пиксель которого несёт информацию о нормали. Нормали, полученные из карты, используются для искажения уже имеющихся нормалей, которые используются в расчётах освещения. Так, в результате получаются по-разному освещённые и затенённые участки. Как итог, абсолютно плоская поверхность визуально кажется неровной [3].
  - 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. Положение источника света задаётся относительно текущей точки наблюдения последовательными поворотами по осям X и Y.

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 определенного объекта [4].

В основном используются следующие три формы моделей:

1) Каркасная (проволочная) модель. В данной модели задается информация о вершинах и ребрах объекта. Это одна из простейших форм задания модели, но она имеет один существенный недостаток: модель не всегда однозначно передает представление о форме объекта;
2) Поверхностные модели. Этот тип модели часто используется в компьютерной графике. Поверхность может описываться аналитически, либо задаваться другим способом (например, отдельными участками поверхности, задаваемыми в качестве участков поверхности того или иного вида). При этом вложенные криволинейные поверхности можно представлять в упрощенном виде, выполняя, например, полигональную аппроксимацию: такая поверхность будет задаваться в виде поверхности

многогранника. Данная форма имеет свой недостаток: отсутствует информация о том, с какой стороны поверхности находится материал;

3) Твердотельные (объемные) модели. Отличие данной формы задания модели от поверхностной формы состоит в том, что в объемных моделях к информации о поверхностях добавляется информация о том, где расположен материал. Это можно сделать путем 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 твердотельное 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 триангулизированной 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 будет представлена цветом (чем выше значение,

прямая зависимость от яркости пикселя — тем больше значение высоты для этой точки). С помощью этого способа можно представить достаточно обширные пространства.

Преимущества:

1. Наглядность данных: в любой программе просмотра графических файлов можно сразу увидеть всю информацию о форме ландшафта;
2. Простота реализации: лёгкость нахождения координат вершины на карте;
3. Эффективность: данный способ позволяет эффективно описывать неровности поверхности, в отличие от аналогичных способов – расчёт освещённости остаётся неизменным.

Недостатки:

1. 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.

Преимущества:

1. Используется значительно меньше информации для построения ландшафта. Нам необходимо хранить только значения высот каждой вершины и связи эти вершины соединяющие. Это даёт нам выигрыш в скорости при передаче огромных массивов информации в видеокарту, в процессе визуализации ландшафта.

Недостатки:

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. При использовании карт высот вы пользуетесь достаточно простыми и "стандартными" средствами пиксельной графики. Хотя бы тот же MS Paint. Тут же вам потребуются более навороченные и "весомые" пакеты.

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].

Преимущества:

1. Кроме самих ландшафтов в таких блоках можно хранить и информацию о зданиях, строениях, растениях, специфических ландшафтных решениях (например, пещеры или скалы, нависающие друг над другом);
2. Возможность создания нескольких вариантов одного и того же сегмента, но при разной степени детализации. В зависимости от скорости или загруженности компьютера можно выбирать более или менее детализованные варианты (так называемые LOD ландшафты — LOD — Level Of Detail).

Недостатки:

1. Проблема стыковки разных сегментов: не понятно, как состыковывать регулярные и иррегулярные сетки;
2. Неочевидность данных: взглянув на картинку, вы не сможете моментально представить, как это должно будет выглядеть в игре;

3. 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»: вычисляется 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

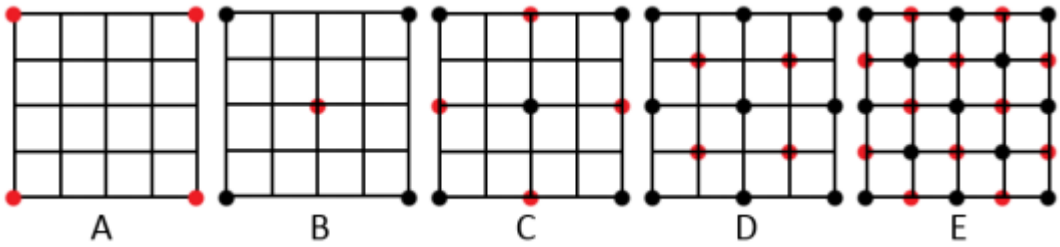


Рис. 1.1. Два этапа работы алгоритма diamond-square

Преимущества:

1. Простота: алгоритм довольно простой с точки зрения реализации;
2. Является довольно быстрым по сравнению с другими алгоритмами.

Недостатки:

1. Есть вероятность появления артефактов на ландшафте, напоминающие всплески;
2. Сложно 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:

1. *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];

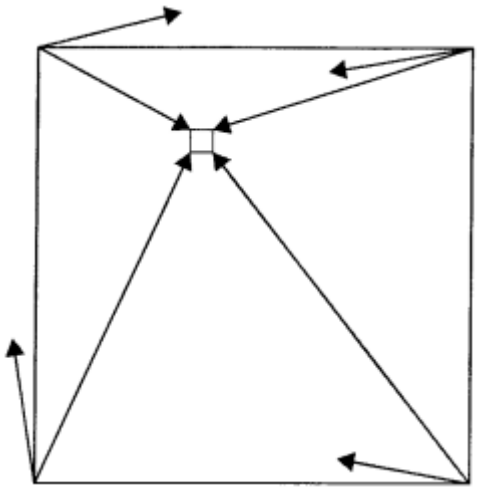


Рис. 1.2. Расчёт значения высоты пикселя при помощи градиентов и векторов, соединяющих точки сетки с расчётной позицией

2. *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). Дальше 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].

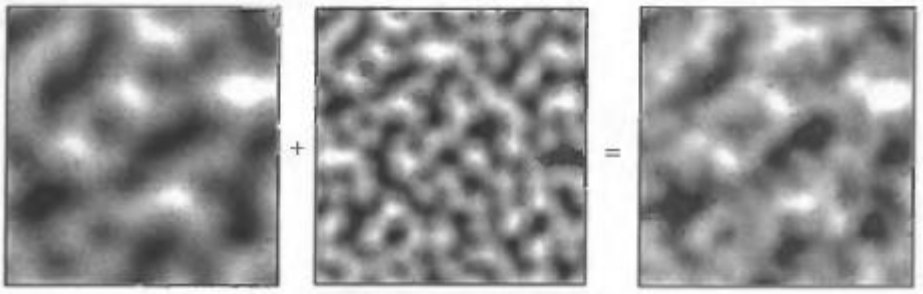


Рис. 1.3. Пример изображения шума Перлина, полученный сложением двух октав.

Преимущества:

1. Является относительно быстрым и детерминированным: зная лишь параметры генерации, можно получить значение высоты любой точки мира, не зная значений высоты соседних точек;
2. Благодаря механизму комбинаций октав, позволяет более тщательно настроить детали ландшафта.

Недостатки:

1. Без комбинаций разных 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:

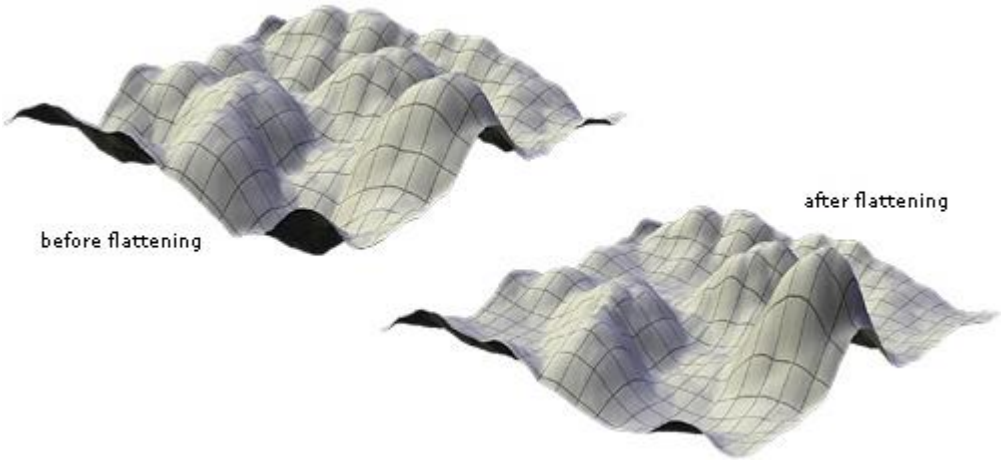


Рис. 1.4. Разница между ландшафтом до долинизации(слева) и после(справа)

Преимущества:

1. Есть возможность контролировать «гористость» ландшафта, получая простые горные и пологие ландшафты;
2. Легко модифицируется под другие типы ландшафта (одинокий остров, озёрные ландшафты и т. д.).

Недостатки:

1. В некоторых случаях требует 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 3 этапа:

1. Этап подготовки исходных данных.

На данном этапе должна быть задана информация о телах. Для каждого тела сцены должна быть сформирована матрица тела V. Размерность матрицы - $4 * n$, где n – количество граней тела.

Каждый столбец матрицы представляет собой четыре коэффициента уравнения плоскости $ax + by + cz + d = 0$, проходящей через очередную грань. Таким образом, матрица тела будет представлена в следующем виде:

$$V = \begin{pmatrix} a_1 & a_2 & & a_n \\ b_1 & b_2 & \cdots & b_n \\ c_1 & c_2 & & c_n \\ d_1 & d_2 & & d_n \end{pmatrix}$$

Матрица тела должна быть сформирована корректно, то есть любая точка, расположенная внутри тела, должна располагаться по положительную сторону от каждой грани тела. Для проведения проверки следует взять точку, расположенную внутри тела. Координаты такой точки можно получить среднее арифметическое координат всех вершин тела. В случае, если для очередной грани условие не выполняется, соответствующий столбец матрицы надо умножить на -1.

2. Этап удаления рёбер, экранируемых самим телом.

На данном этапе рассматривается вектор взгляда. Если зритель находится в бесконечности на положительной полуоси z, то его взгляд направлен в сторону отрицательной полуоси z. Вектор такого взгляда равен: $E = \{00 - 10\}$.

Для определения невидимых граней достаточно умножить вектор E на матрицу тела V. Отрицательные компоненты полученного вектора будут соответствовать невидимым граням.

3. Этап удаления невидимых рёбер, экранируемых другими телами сцены.

На данном этапе для определения невидимых точек ребра требуется построить луч, соединяющий точку наблюдения с точкой на ребре. Точка будет невидимой, если луч на своём пути встречает в качестве преграды рассматриваемое тело. Если тело является преградой, то луч должен пройти через тело. Если луч проходит через тело, то он находится по положительную сторону от каждой грани тела.

4. Этап удаление линий пересечения тел, экранируемых самими телами, связанными отношением протыкания и другими телами.

Решается протыкание при помощи запоминания всех точек протыкания и добавлении к сцене отрезков, связывающих эти точки. Отрезки образуются путём соединения каждой точки протыкания со всеми остальными точками протыкания для этой пары объектов. Затем проверяется экранирование этих отрезков данными и другими телами. Видимые отрезки образуют структуру протыкания.

Преимущества:

1. Высокая точностью.

Недостатки:

1. Квадратичная вычислительная сложность алгоритма от числа объектов;
2. Не выпуклые многогранники нужно делить на выпуклые для корректной работы алгоритма;
3. Нет 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.

Для решения задачи вычисления глубины z каждый многоугольник описывается уравнением $ax + by + cz + d = 0$. При $c = 0$ многоугольник для наблюдателя вырождается в линию.

Для некоторой сканирующей строки $y = const$, поэтому имеется возможность рекуррентно высчитывать z' для каждого $x' = x + dx$:

$$z' - z = \frac{-ax' + d}{c} + \frac{ax + d}{c} = \frac{a(x - x')}{c}$$

Получим: $z' = z - \frac{a}{c}$, *так как* $x - x' = dx = 1$

При этом стоит отметить, что для невыпуклых многогранников предварительно потребуется удалить нелицевые грани.

Преимущества:

1. Простота реализации;
2. Оценка вычислительной трудоёмкости алгоритма линейна;
3. Экономия вычислительного времени, так как элементы сцены не сортируются.

Недостатки:

1. Большой объем требуемой памяти;
2. Реализация эффектов прозрачности сложна.

1.6.3 Алгоритм обратной трассировки лучей

Работает в пространстве изображения.

Отслеживать ход лучей от источника света до объекта и он него до наблюдателя неэффективно, так как лишь малая часть таких лучей дойдёт до

наблюдателя, поэтому было предложено отслеживать лучи 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.

Если точка зрения не находится в бесконечности (перспективная проекция) [11]:

- Предполагается, что наблюдатель так же находится на OZ;
- Растр перпендикулярен OZ;
- Задача состоит в том, чтобы построить одноточечную центральную проекцию на картинную плоскость.

Определения пересечений происходит с помощью погружения объектов в некоторую выпуклую оболочку – например, сферическую. Поиск пересечения с такой оболочкой происходит проще: достаточно проверить превосходит ли радиус сферы-оболочки расстояние от центра этой сферы до луча.

Преимущества:

1. Высокая реалистичность синтезируемого изображения;
2. Работа с поверхностями в математической форме;
3. Вычислительная сложность слабо зависит от сложности сцены.

Недостатки:

1. Трудоёмкие вычисления.

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. Указанные факты говорят о том, что обратная трассировка лучей 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]:

$$I_\alpha = I_0 \cdot cos(\alpha)$$

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.

Так как для реалистичности конечного изображения будут использованы карты нормалей, то при расчёте освещения объекта будут использованы не только нормали на полигонах, но и нормали с карты, что увеличит качество изображения без изменения геометрии объекта.

Существует несколько методов закрашивания изображаемых поверхностей. Рассмотрим их.

1.7.1 Простая закраска

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).

Преимущества:

1. Простой и быстрый;
2. Используется для работы с многогранниками с диффузным отражением.

Минусы:

1. 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

$$I_a = I_1 \frac{y_s - y_2}{y_1 - y_2} + I_2 \frac{y_1 - y_s}{y_1 - y_2},$$
$$I_b = I_1 \frac{y_s - y_3}{y_1 - y_3} + I_3 \frac{y_1 - y_s}{y_1 - y_3},$$
$$I_p = I_a \frac{x_b - x_p}{x_b - x_a} + I_b \frac{x_p - x_a}{x_b - x_a}.$$

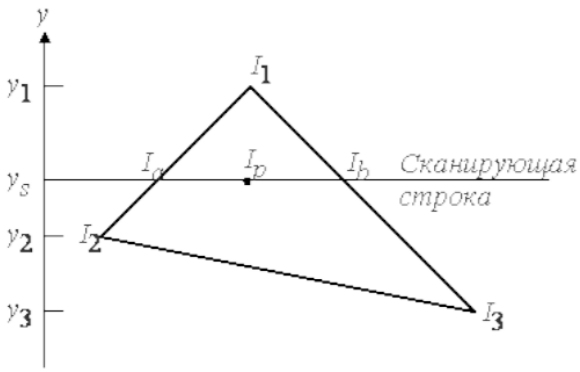


Рис. 1.5. Интерполяция интенсивностей

Для цветных объектов отдельно интерполируется каждая из компонент цвета.

Преимущества:

1. Используется с фигурами вращения, аппроксимированные полигонами;

Недостатки:

1. 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 нормалями, которые сами тоже являются результатами интерполяции вдоль рёбер многоугольника между нормалями 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.

Преимущества:

1. Используется с фигурами вращения, аппроксимированные полигонами, с зеркальным отражением.

Недостатки:

1. 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

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].

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 [2]:

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. Преимущество такой технологии в том, что объекты реального мира служат маркерами сами по себе и для них не нужно создавать специальных визуальных идентификаторов.
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.* Кроме маркерной и безмаркерной, существует технология дополненной реальности, основанная 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 распознания изображений. Вообще говоря, множество маркеров достаточно широко: ими могут быть и геометрические фигуры простой формы (например, круг, квадрат), и объекты в форме прямоугольного параллелепипеда, и даже глаза и лица людей.

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 фудуциальных маркеров на изображения [14]. При этом она может обнаруживать маркеры других библиотек, таких как: AprilTag и ARToolKit.

Преимущества:

1. Лёгкая настройка (с генератором маркеров ArUco);
2. Меньше ложных срабатываний

Недостатки:

1. Более восприимчивы 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.

Преимущества:

1. Неплохо работает даже на большом расстоянии;
2. Более гибкий дизайн маркеров (например, маркеры не обязательно квадратные);
3. Встроенная поддержка пакетов тегов, в которых несколько тегов объединяются в один тег (использование нескольких тегов с разной ориентацией эффективно устраняет проблему неоднозначности вращения)

Недостатки:

1. Менее прямолинейная настройка;
2. Больше ложных срабатываний.

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.

Преимущества:

1. Есть возможность генерации и отслеживания квадратных и естественных маркеров;
2. Позволяет использовать стереокамеры.

Недостатки:

1. Излишняя функциональность: 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.

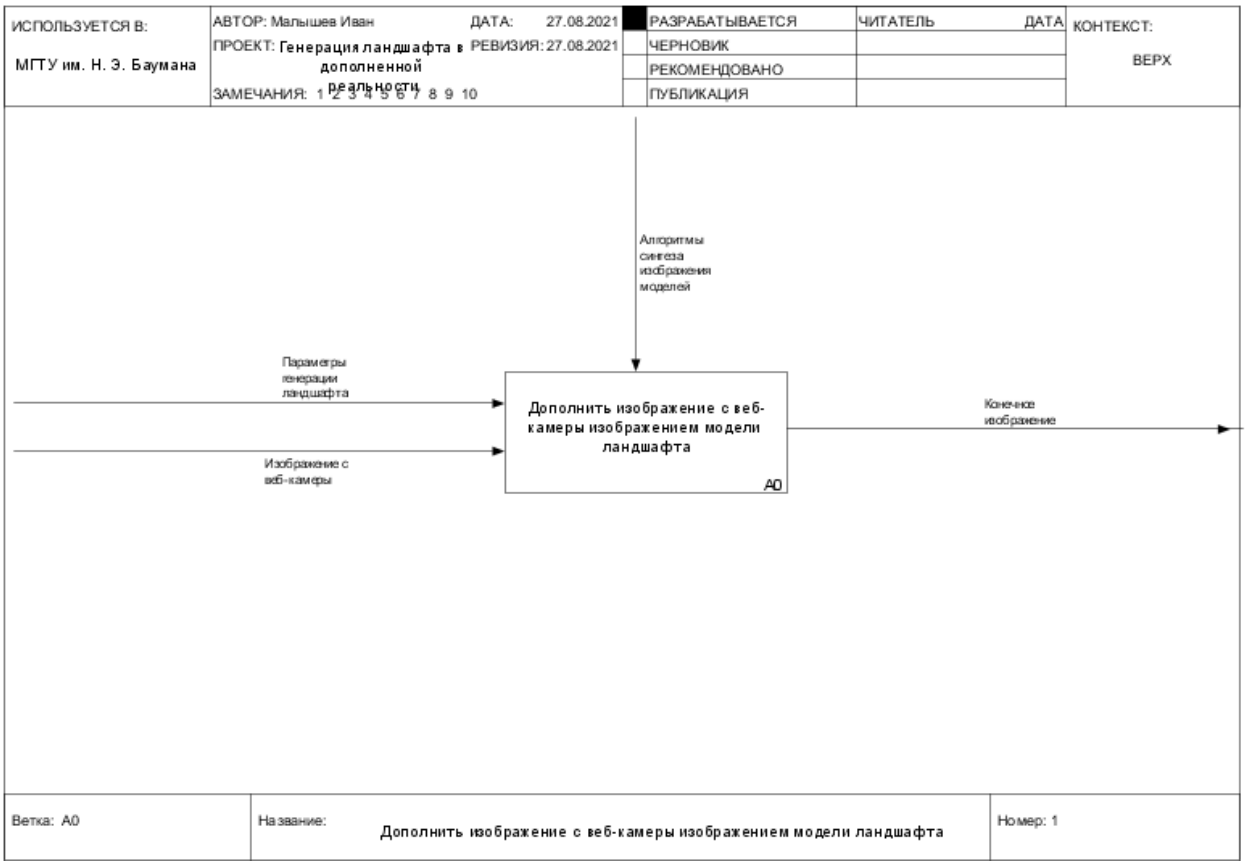


Рис. 2.1. Схема общей постановки задачи

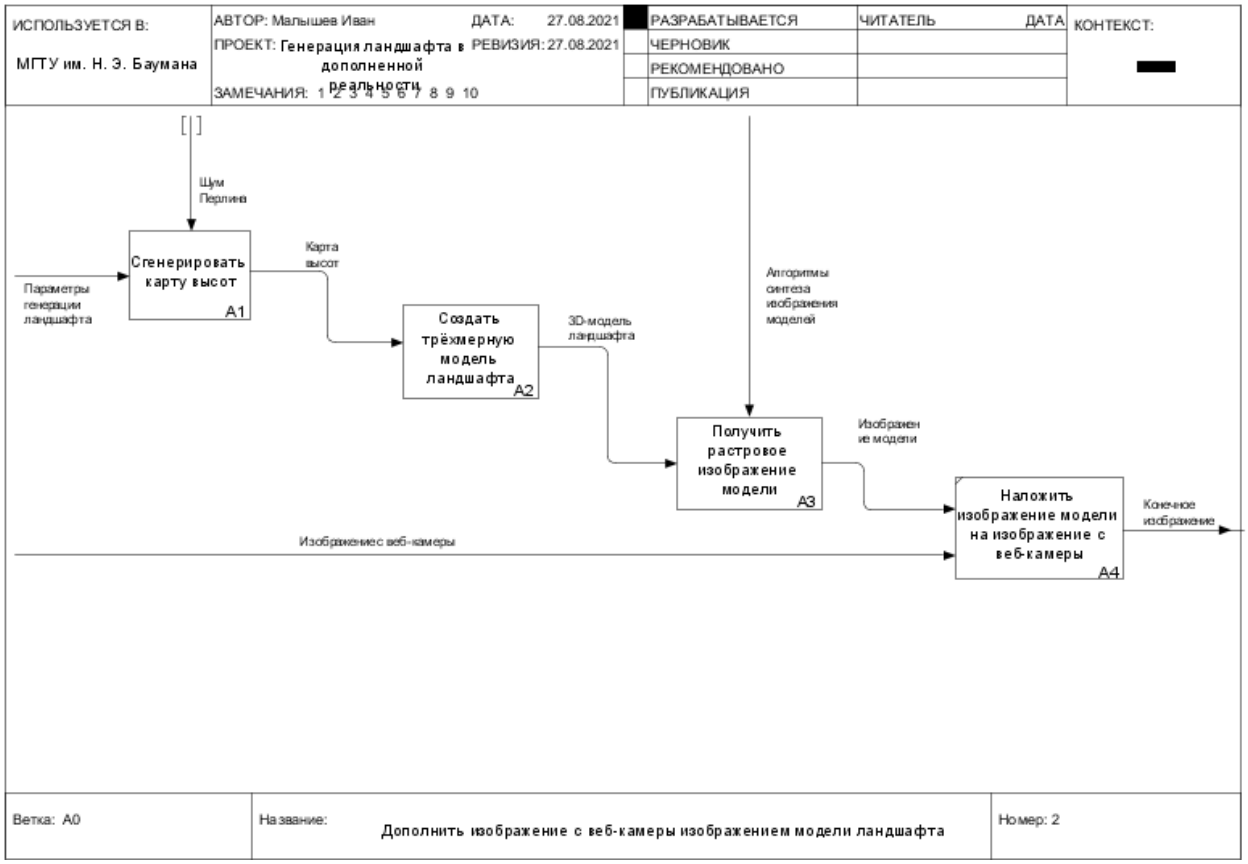


Рис. 2.2. Схема алгоритма синтеза виртуального и реального изображения

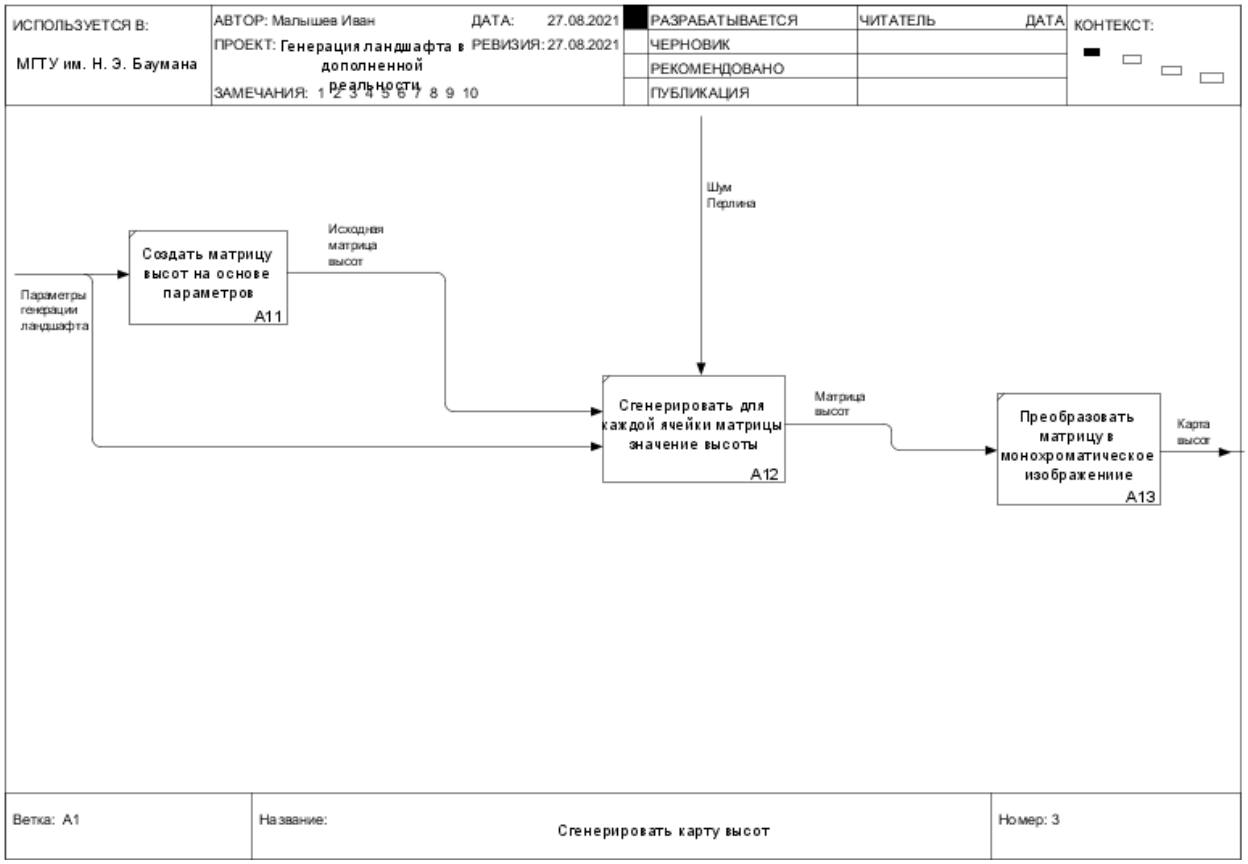


Рис. 2.3. Схема алгоритма генерации карты высот

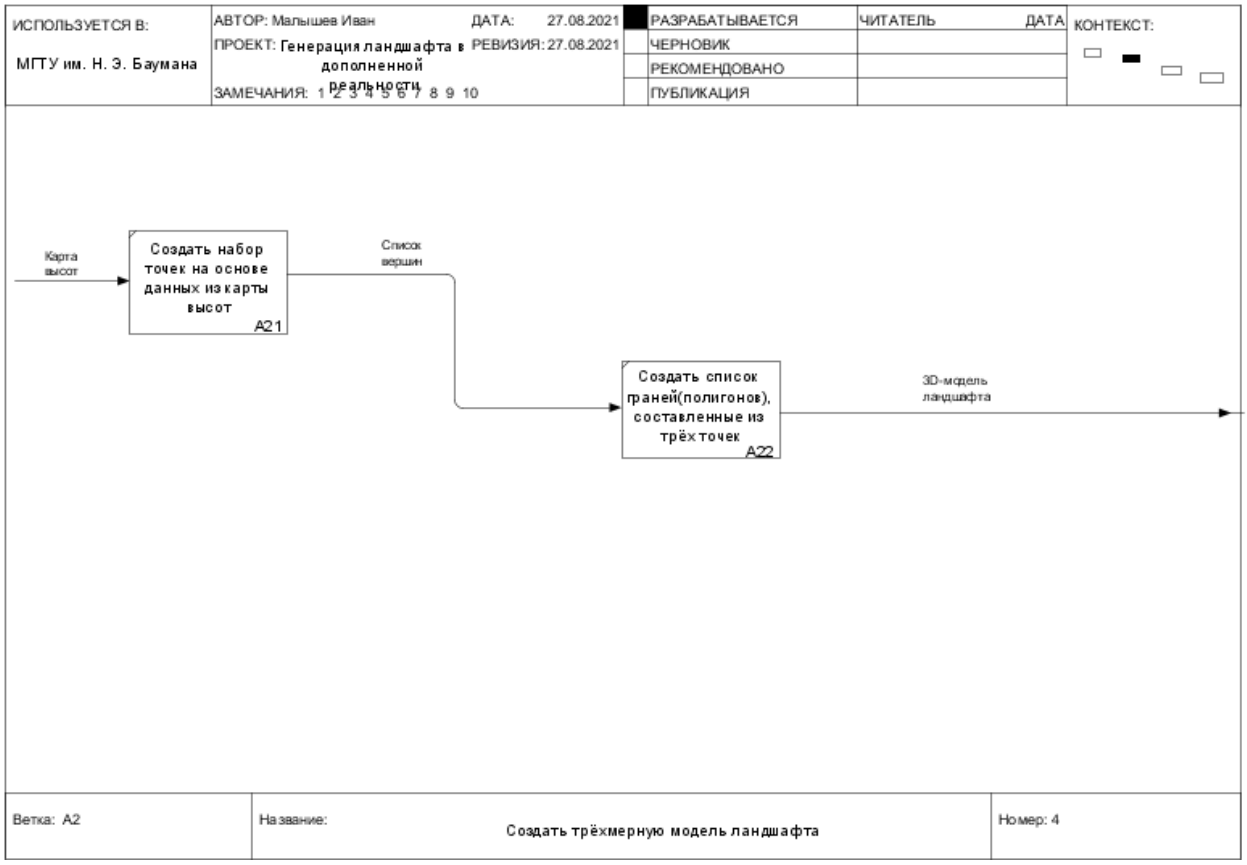


Рис. 2.4. Схема алгоритма создания 3D-модели ландшафта

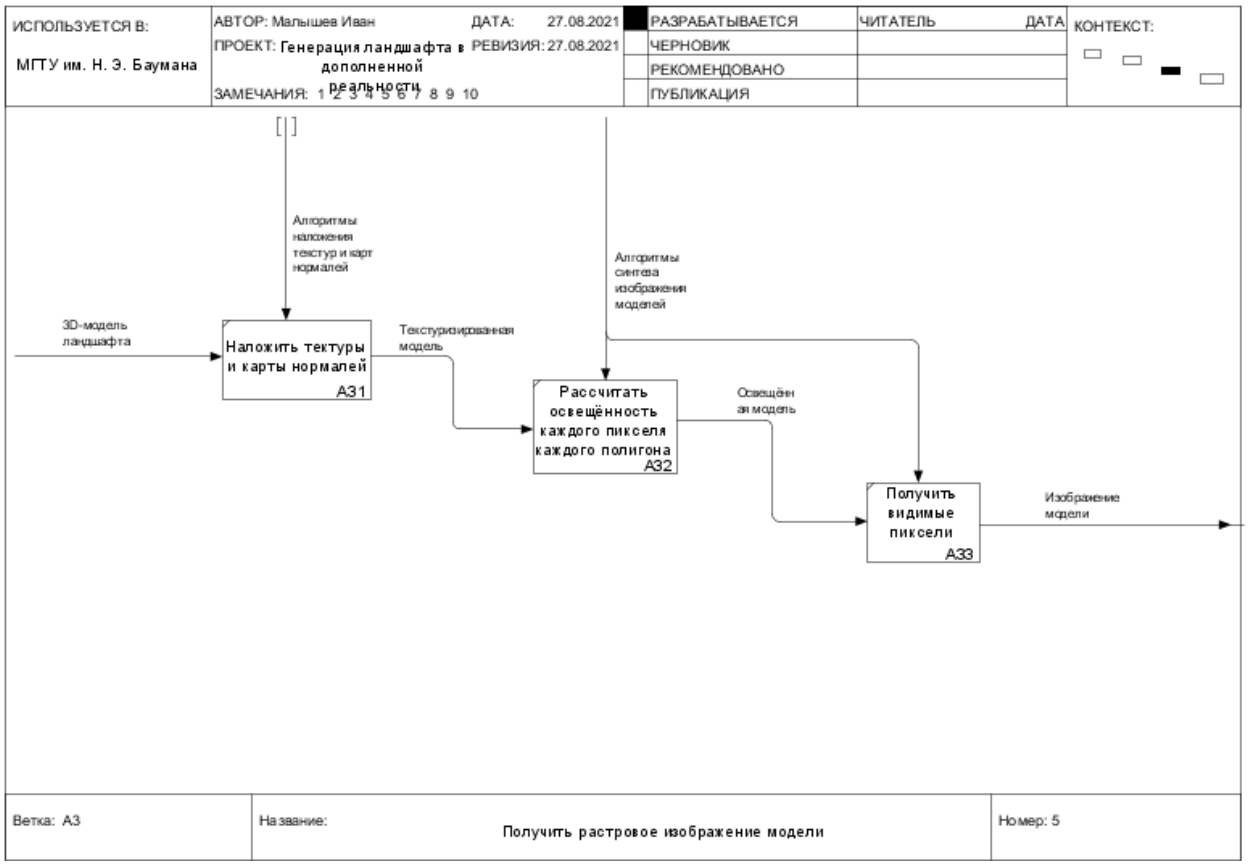


Рис. 2.5. Схема алгоритма растеризации модели

2.2.2 Наложение текстур и карт нормалей

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].

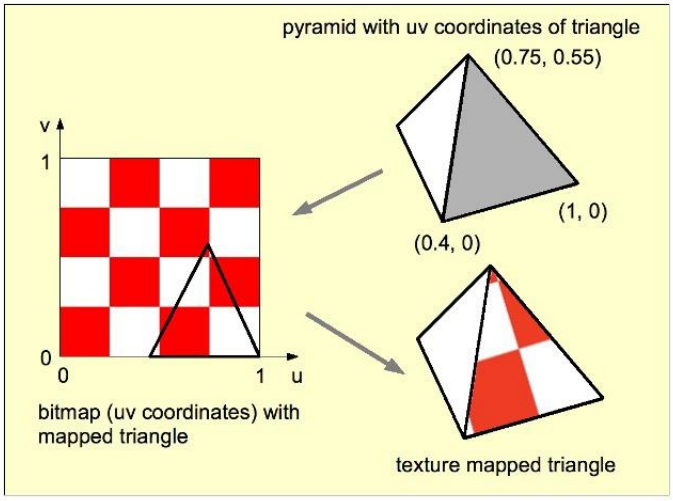


Рис. 2.6. Представление UV-координат

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:

$$u_\alpha = (1 - \alpha)u_0 + \alpha u_1, 0 \le \alpha \le 1$$

, 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

Наложение карт нормалей происходит точно таким же образом. Из полученной части карты нормалей извлекается информация о нормалях поверхности, которая потом используется в раскраске.

На рисунке 2.8 представлена схема алгоритма наложения.

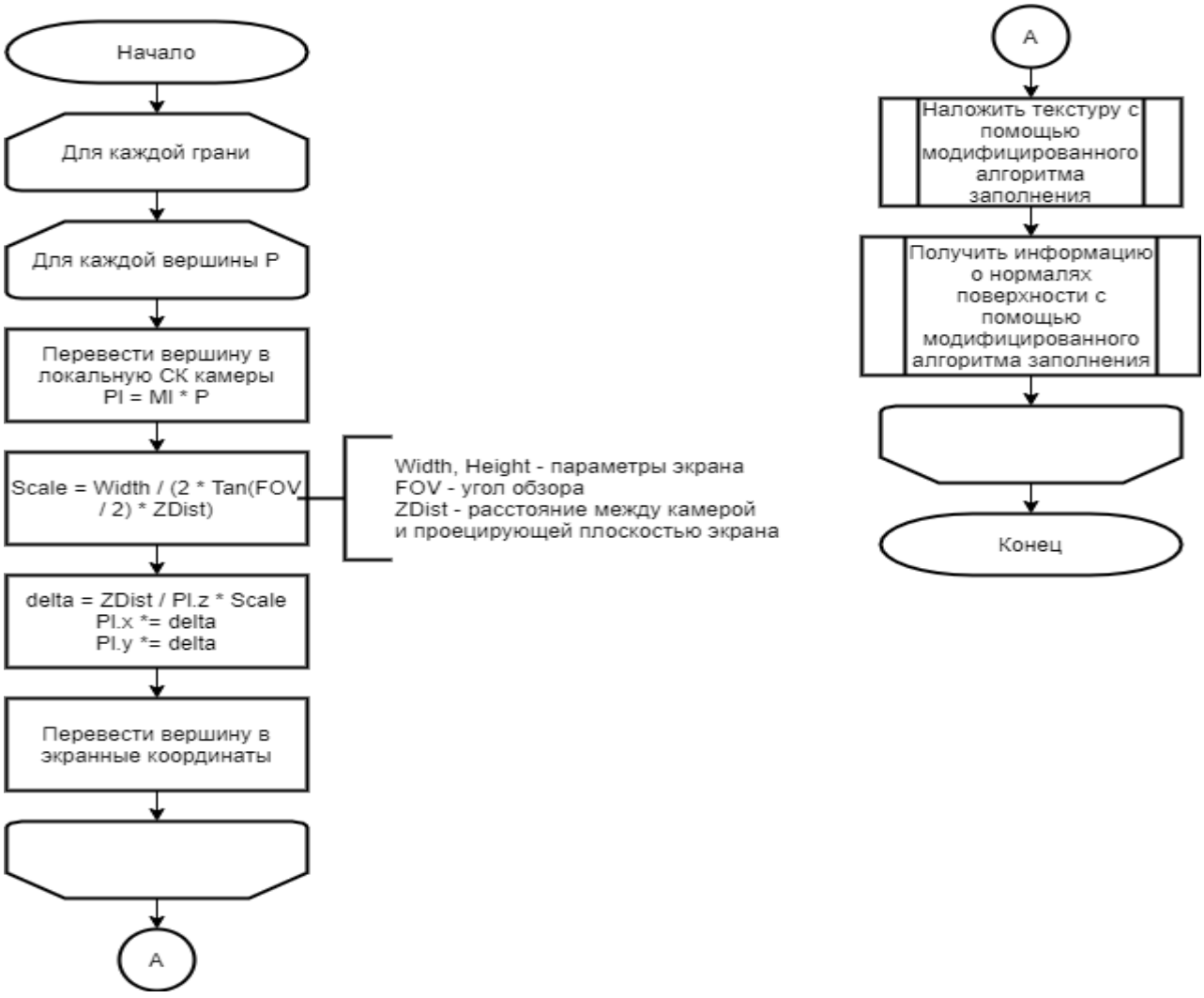


Рис. 2.8. Схема наложения текстур и нормалей на полигон

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. $u_\alpha = \frac{(1-\alpha)\frac{u_0}{z_0}+\alpha\frac{u_1}{z_1}}{(1-\alpha)\frac{1}{z_0}+\alpha\frac{1}{z_1}}, \alpha = \frac{x-x_0}{x_1-x_0}$
2. $v_\alpha = \frac{(1-\alpha)\frac{v_0}{z_0}+\alpha\frac{v_1}{z_1}}{(1-\alpha)\frac{1}{z_0}+\alpha\frac{1}{z_1}}, \alpha = \frac{y-y_0}{y_1-y_0}$
3. $x_{pixel} = floor(M_x * u_\alpha), y_{pixel} = floor(M_y * v_\alpha)$, где M_x, M_y – ширина и высота текстуры в растре

Часто у полигонов, располагающиеся слишком далеко, тексельная 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).

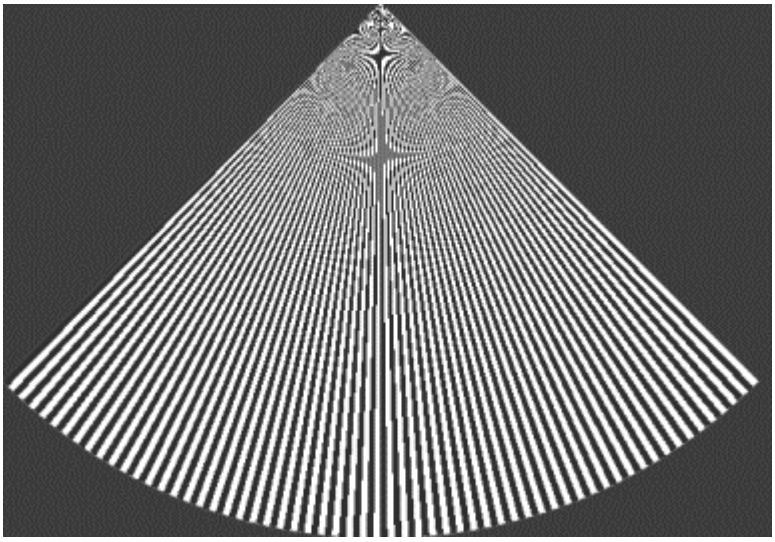


Рис. 2.9. Пример случая появления муара

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].

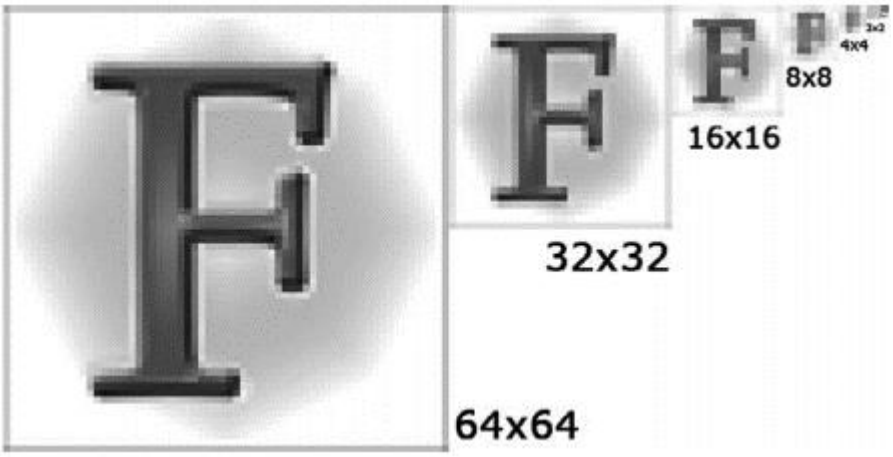


Рис. 2.10. MIP-уровни для текстуры размером 64 × 64

Для вычисления MIP-уровня используется следующая формула:

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

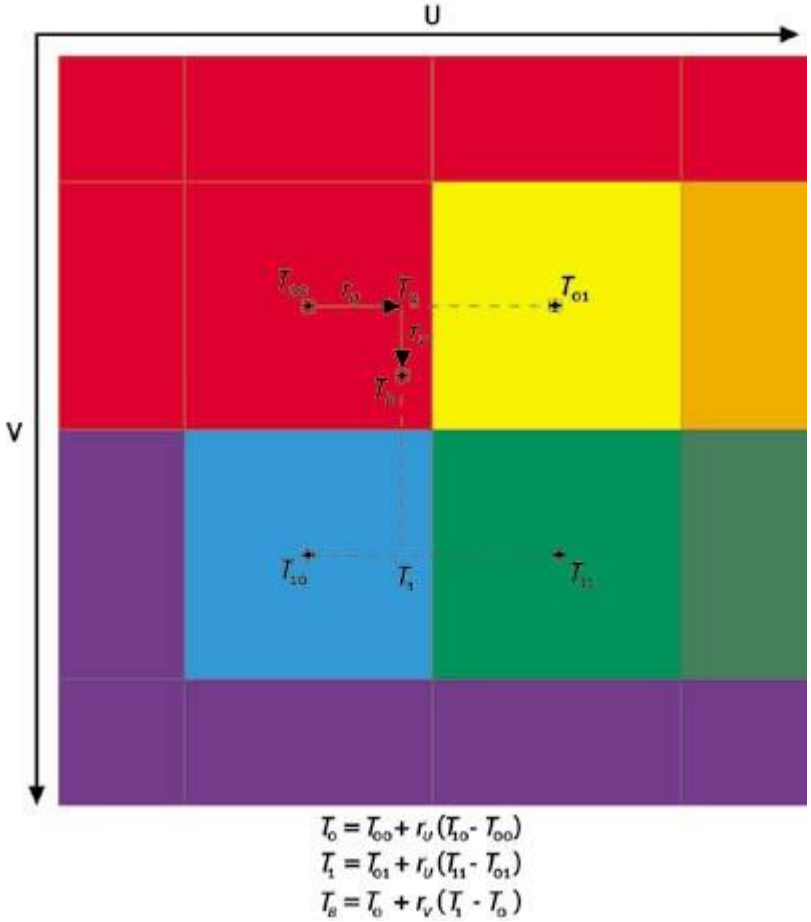
, где x_{min}, x_{max}, y_{min}, y_{max} – минимальные и максимальные координаты x и y полигона в растре; mipcount – общее число мип-уровней.

При рендеринге пиксела текстурируемого треугольника графическое аппаратное обеспечение сэмплирует растровое изображение текстуры, учитывая положение центра пиксела в её пространстве. Поскольку в большинстве случаев текселы не отображаются на пикселы один к одному, центры пикселов могут находиться в любом месте текстурного пространства, в том числе на разделяющей текселы границе. Поэтому для получения цвета фактически сэмплируемого тексела графическое аппаратное обеспечение обычно сэмплирует несколько текселов и смешивает 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.

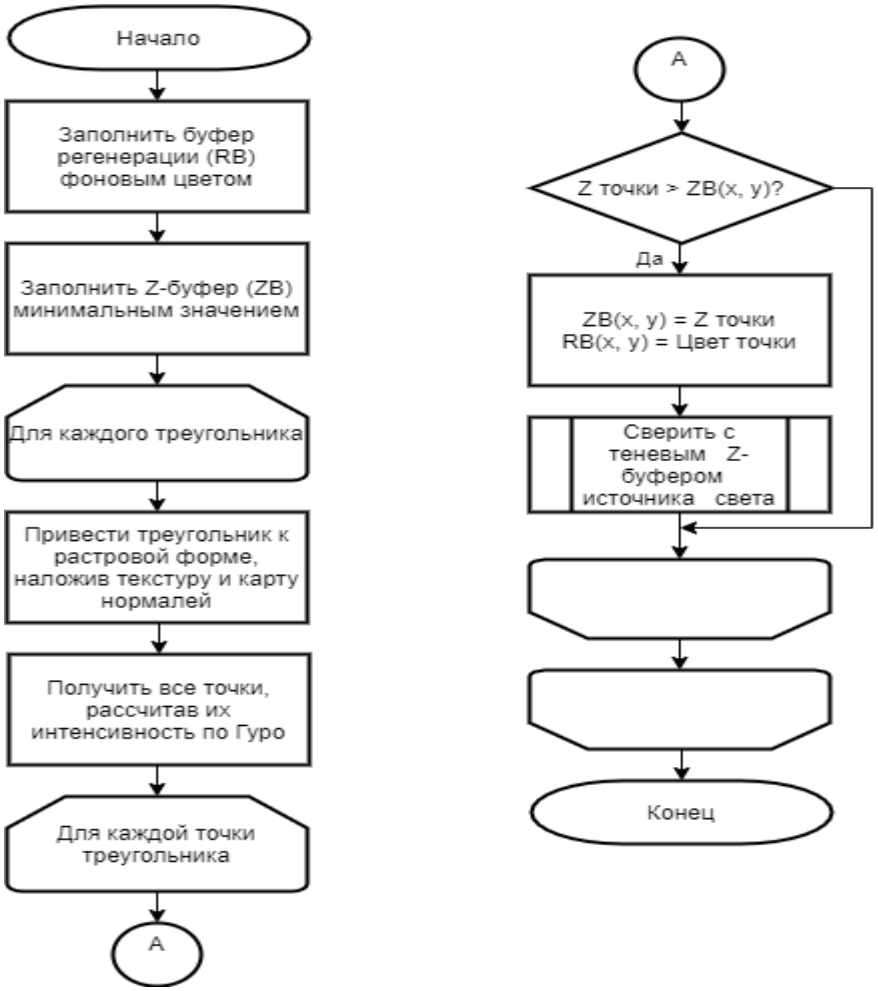


Рис. 2.12. Схема синтеза реалистичного изображения

2.2.4 Построение теней

На рисунке 2.13 представлена схема алгоритма построения теней.

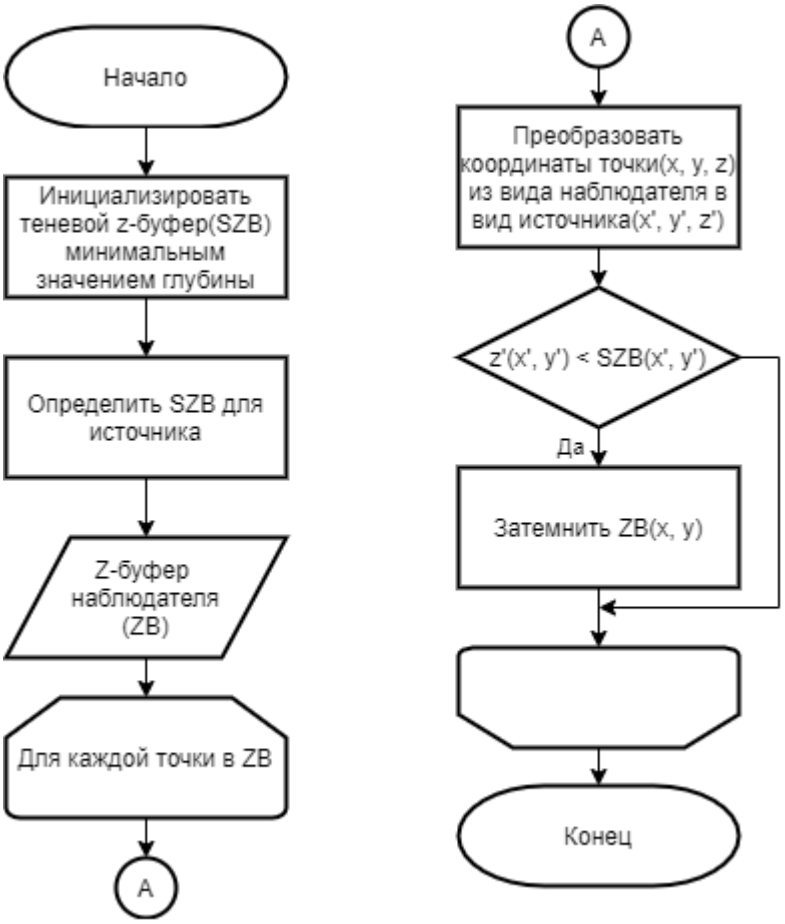


Рис. 2.13. Схема алгоритма построения теней

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
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; координаты нормали поверхности в вершине N_x, N_y, N_z
Полигон	Массив из трёх индексов списка вершин, список нормалей поверхности
Модель ландшафта	Список вершин, полигонов, список текстур и карт нормалей; пространство модели (ЛСК)
Текстура и карта нормалей	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	Интенсивность I, Цвет света C (по умолч. белый) и вектор направления **L**
Пользовательский интерфейс	Библиотечные классы
Дополненная реальность	Классы ArUco

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. Такой же принцип освещения соблюдается и у выбранного источника света.

Так как в качестве источника света был выбран направленный, то для построения теней используется ортогональная проекция, тогда как для определения видимости наблюдателем – перспективная [19].

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 и карт нормалей с последующей фильтрацией текстур, 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, 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 заботится 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 и аппаратным ускорением. Это значит, что приложение будет корректно масштабироваться под разные экраны с разным разрешением, при этом для обработки элементов пользовательского интерфейса используются мощности видеокарты, что уменьшит нагрузку на процессор в пользу расчётов конечного изображения, а также данный интерфейс не будет жёстко зависеть от логики программы.

Для реализации технологии дополненной реальности была выбрана обёртка библиотеки OpenCV [24] для .Net 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] для прямой работы с репозиториями. Плагины позволяют облегчить и ускорить разработку программного обеспечения.

3.2 Разработка и реализация программных модулей

3.3 Тестирование

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. Трехмерное 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/1QU9KoBtjSM2kF6ITOjQ76xqL7H0TEtXriJX5kwi9Kgc/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/mxscrptxmpls/txtr/txtr.html (дата обращения 13.07.2021)
18. Perspective Texture mapping [Электронный ресурс]. – Режим доступа: http://www.lysator.liu.se/~mikaelk/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)
31.