



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 «Робототехника и комплексная автоматизация»

КАФЕДРА «Системы автоматизированного проектирования (РК-6)»

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
НА ТЕМУ:**

***«Разработка реалистичных природных
ландшафтов на Unreal Engine 4»***

Студент РК6-81Б

(Группа)

А.В. Фёдоров

(И.О.Фамилия)

Руководитель ВКР

Ф.А. Витюков

(И.О.Фамилия)

Консультант

(И.О.Фамилия)

Консультант

(Подпись, дата)

(И.О.Фамилия)

Нормоконтролер

С.В. Грошев

(И.О.Фамилия)

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)**

УТВЕРЖДАЮ

Заведующий кафедрой РК6
(Индекс)
_____ Карпенко А.П.
(И.О.Фамилия)
«_____» 20 ____ г.

З А Д А Н И Е
на выполнение выпускной квалификационной работы

Студент группы РК6-81Б

_____ Фёдоров Артемий Владиславович

(фамилия, имя, отчество)

Тема квалификационной работы «Разработка реалистичных природных ландшафтов на Unreal Engine 4»

При выполнении ВКР:

Используются / Не используются	Да/Нет
1) Литературные источники и документы, имеющие гриф секретности	Нет
2) Литературные источники и документы, имеющие пометку «Для служебного пользования», иных пометок, запрещающих открытое опубликование	Нет
3) Служебные материалы других организаций	Нет
4) Результаты НИР (ОКР), выполняемой в МГТУ им. Н.Э.Баумана	Нет
5) Материалы по незавершенным исследованиям или материалы по завершенным исследованиям, но ещё не опубликованные в открытой печати	Нет

Тема квалификационной работы утверждена распоряжением по факультету
№ _____ от «____» 202_ г.

Часть 1. Аналитическая часть

Изучить полный цикл создания природных ландшафтов. Изучить средства работы с ландшафтами большого размера, предоставляемые трёхмерным движком Unreal Engine 4. Провести анализ различных методов рельефного текстурирования. Изучить средства создания и анимации игровых персонажей в Unreal Engine 4.

Часть 2. Практическая часть 1. Создание ландшафтов

Создать природную сцену небольшого размера с использованием готовых ассетов. Создать ландшафт большого размера, выполнить импорт в движок Unreal Engine 4, провести оптимизацию отображения.

Часть 3. Практическая часть 2. Разработка водоёмов

Разработать шейдеры речной воды. Разработать инструмент для создания рек произвольной формы. С помощью полученных инструментов создать и встроить реку в природный ландшафт.

Часть 4. Практическая часть 3. Разработка игрового персонажа

Разработать анимированного игрового персонажа. Создать искусственный интеллект, управляющий персонажем-врагом. Разработать боевую систему, заключающуюся в возможности выбирать и атаковать персонажа-врага.

Оформление квалификационной работы:

Расчетно-пояснительная записка на 69 листах формата А4.

Перечень графического (илюстративного) материала (чертежи, плакаты, слайды и т.п.)

Работа содержит 8 графических листов формата А4

Дата выдачи задания «10» февраля 2024 г.

В соответствии с учебным планом выпускную квалификационную работу выполнить в полном объеме в срок до «26» июня 2024 г.

Руководитель квалификационной работы

Ф.А.Витюков

(Подпись, дата)

(И.О.Фамилия)

Студент

А.В.Фёдоров

(Подпись, дата)

(И.О.Фамилия)

Примечание:

1. Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

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 РК
КАФЕДРА РК6
ГРУППА РК6-81Б

УТВЕРЖДАЮ
Заведующий кафедрой РК6
(Индекс)
Карпенко А.П.
(И.О.Фамилия)

« _____ » 20 ____ г.

КАЛЕНДАРНЫЙ ПЛАН
выполнения выпускной квалификационной работы

студента: Фёдорова Артемия Владиславовича
(фамилия, имя, отчество)

Тема квалификационной работы «Разработка реалистичных природных ландшафтов на Unreal Engine 4»

№ п/п	Наименование этапов выпускной квалификационной работы	Сроки выполнения этапов		Отметка о выполнении	
		план	факт	Должность	ФИО, подпись
1.	Задание на выполнение работы. Формулирование проблемы, цели и задач работы	10.02.2024 Планируемая дата		Руководитель ВКР	
2.	1 часть _____	Планируемая дата		Руководитель ВКР	
3.	Утверждение окончательных формулировок решаемой проблемы, цели работы и перечня задач	Планируемая дата		Заведующий кафедрой	
4.	2 часть _____	Планируемая дата		Руководитель ВКР	
5.	3 часть _____	Планируемая дата		Руководитель ВКР	
6.	1-я редакция работы	05.06.2024 Планируемая дата		Руководитель ВКР	
7.	Подготовка доклада и презентации	08.06.2024 Планируемая дата			
8.	Отзыв руководителя	10.06.2024 Планируемая дата		Руководитель ВКР	
9.	Нормоконтроль	15.06.2024 Планируемая дата		Нормоконтролер	
10.	Внешняя рецензия	18.06.2024 Планируемая дата			
11.	Защита работы на ГЭК	26.06.2024 Планируемая дата			

Студент _____
(подпись, дата)

Руководитель работы _____
(подпись, дата)

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)**

**НАПРАВЛЕНИЕ НА ГОСУДАРСТВЕННУЮ ИТОГОВУЮ
АТТЕСТАЦИЮ**

Председателю
Государственной Экзаменационной Комиссии № _____

факультета _____ МГТУ им. Н.Э. Баумана

Направляется студент _____ группы _____

на защиту выпускной квалификационной работы _____

Декан факультета _____ « ____ » 20 ____ г.

Справка об успеваемости

Студент _____ за время пребывания в МГТУ имени Н.Э. Баумана с 20 ____ г. по 20 ____ г. полностью выполнил учебный план со следующими оценками: отлично – _____ %, хорошо – _____ %, удовлетворительно – _____ %.

Инспектор деканата _____

Отзыв руководителя выпускной квалификационной работы

Студент _____

Руководитель ВКР _____ « ____ » 20 ____ г.

(ФИО студента)

(подпись)

(дата)

АННОТАЦИЯ

Работа посвящена различным способам и инструментам разработки реалистичных пейзажей и ландшафтов в Unreal Engine 4.

В данной работе рассмотрены и проиллюстрированы следующие этапы: прохождение полного цикла создания ландшафта, импорт в движок Unreal Engine 4, оптимизация отображения больших ландшафтов с помощью механизма тайлинга, создание материалов ландшафта, добавление растительности в сцены, настройка освещения и погодных условий, реализация интерактивного управления погодными условиями, создание водоёмов и интеграция их в природные сцены.

Тип работы: выпускная квалификационная работа.

Тема работы: «Разработка реалистичных природных ландшафтов на Unreal Engine 4».

Объекты исследований: 3d-моделирование, создание шейдеров, повышение производительности при рендеринге.

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

Game engine (игровой движок) – набор ключевых компонентов программного обеспечения, используемых для разработки игр и иных 3d-приложений. Как правило, инструменты движка абстрагированы от специфики конкретной игры, но могут учитывать некоторые особенности жанра – они предоставляют «базис» для разработки, «надстройку» над которым создает его пользователь.

Unreal Engine 4 (UE4) – игровой движок, разрабатываемый и поддерживаемый компанией Epic Games.

3d-model (3d-модель) – математическое представление объекта в трехмерном пространстве.

3d-modeling (3d-моделирование) – процесс создания 3d-модели объекта.

Текстурирование – процесс создания текстур объекта.

High-poly модель (высокополигональная модель) – максимально детализированная версия 3d-модели. Имеет большое количество полигонов (от нескольких сотен тысяч до миллионов), в основном используется для дальнейшего запекания текстур. Как правило, высокополигональные модели не используются при отрисовке в реальном времени, поскольку для их обработки требуется слишком много вычислительных ресурсов.

Low-poly модель (низкополигональная модель) – модель, содержащая относительно низкое количество полигонов (несколько тысяч), при этом сохраняющая основные геометрические свойства объекта. Низкополигональные модели широко используются при отрисовке в реальном времени, поскольку затраты на их обработку приемлемы для получения достаточно плавного изображения.

Actor (актёр) – в рамках движка UE4 любой объект, который может быть размещен на уровне. Базовый класс всех actor'ов – AActor.

Actor Component – специальный тип объекта, который может быть присоединен к выбранному actor'у как подобъект (subobject). Как правило,

используется для внедрения функциональности, общей для различных actor'ов.
Базовый класс – UActorComponent.

Transform (трансформ) – данные о местоположении, повороте и масштабе объекта. Представляются матрицей преобразований.

Scene Component (USceneComponent) – класс, производный от UActorComponent. Представляет собой компонент, который может иметь свой трансформ на сцене. Данный компонент используется для внедрения функциональности, не требующей геометрического представления.

Полигональная сетка – набор вершин, рёбер и граней, определяющий внешний вид многогранного объекта.

Полигон – многоугольник, являющийся гранью или набором граней 3d-сетки. Основные типы: треугольник (tri), четырёхугольник (quad) и n-gon (5 или более вершин).

Sculpting (скульптинг) – процесс создания и детализации 3d-моделей с помощью 3d-кистей.

Rendering (рендеринг, отрисовка) – процесс получения 2d-изображения по имеющейся 3d-модели.

Polycount – количество полигонов модели.

Static Mesh (UStaticMesh) – класс, представляющий собой статический геометрический объект. Хранит данные о полигональной сетке модели.

Текстура – изображение, накладываемое на поверхность 3d-модели. Может содержать одно или несколько свойств поверхности, например: цвет, жёсткость (roughness), смещение (displacement), направление нормалей (normal map), и т.д.

Шейдер (Shader) – компьютерная программа, выполняющаяся параллельно на графическом процессоре, и служащая для различных графических вычислений, таких как отрисовка 3д моделей, расчёт освещения и так далее.

Материал – набор свойств, определяющих поведение света при отражении от поверхности. Материалы также могут использовать одну или несколько текстур.

Material (UMaterial) – класс, позволяющий хранить и модифицировать информацию о материале.

Static Mesh Component (UStaticMeshComponent) – компонент, который может иметь статический меш и набор материалов, применимых к нему.

Geometry instancing (инстансинг, дублирование геометрии) – техника, позволяющая отрисовывать множество однотипных элементов за один проход.

Instanced Static Mesh Component (UInstancedStaticMeshComponent, ISMC) – класс, производный от UStaticMeshComponent, позволяющий использовать механизм инстансинга геометрии.

LOD (Level of Detail, уровни детализации) – техника, позволяющая подменять разные по детализации версии модели в зависимости от дистанции между камерой и объектом, либо в зависимости от процента площади экрана, занимаемой моделью.

Hierarchical Instanced Static Mesh Component (HISMC, UHierarchicalInstancedStaticMeshComponent) – класс, производный от UInstancedStaticMeshComponent, позволяющий использовать LOD.

Central Processing Unit (CPU) – центральный процессор.

Graphics Processing Unit (GPU) – графический процессор (видеокарта).

Frames per second (FPS) – количество кадров в секунду.

Rendering Hardware Interface (RHI) – в UE4 надстройка над множеством графических API (например: DirectX, OpenGL, Vulkan), позволяющая писать независимый от платформы код.

Asset (ассет) – в контексте компьютерных игр, объект, представляющий собой единицу контента. Игровыми ассетами являются 3d-модели, текстуры, материалы, аудиофайлы, и т.д. Как правило, созданием ассетов занимаются художники, дизайнеры, музыканты.

Reference (референс) – изображение, используемое художником в процессе 3d-моделирования для получения дополнительной информации о моделируемом объекте.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	13
1. Создание природной сцены со статичной камерой.....	17
1.1. Создание ландшафта.....	18
1.2. Заполнение сцены объектами	19
1.3. Настройка освещения и погодных условий	20
2. Создание большой сцены с природным ландшафтом.....	23
2.1 Создание карты местности.....	23
2.2 Создание материала ландшафта	24
2.3 Добавление растительности в сцену	26
2.4 Добавление деталей	28
2.5 Обзор техник рельефного текстурирования	29
2.5.1 Измерение производительности различных методов рельефного текстурирования	34
2.6 Измерение производительности сцен с ландшафтом	36
3. Разработка инструментов для создания водоёмов.....	38
3.1 Разработка шейдера речной воды	38
3.1.1 Предварительное создание материала воды	39
3.1.2 Реализация ряби на поверхности воды	40
3.1.3 Реализация изменения цвета в зависимости от глубины.....	42
3.1.4 Реализация эффекта полного отражения света при низких углах падения.	44
3.1.5 Создание пены на поверхности воды	45
3.2 Разработка инструмента для создания рек	46
3.3 Внедрение водоёмов в готовую природную сцену	47
4. Разработка игрового персонажа	48
4.1. Решения, предоставляемые движком Unreal Engine 4	48
4.2. Создание системы перемещения персонажа.....	49
4.3. Настройка анимаций персонажа.....	52
4.3.1 Настройка анимаций перемещения.....	52
6.3.2 Создание анимаций атак	53
4.4. Разработка боевой системы	55

4.5. Разработка искусственного интеллекта.....	60
4.6. Настройка многопользовательской игры	62
4.7. Внедрение игровых персонажей в природную сцену	64
ЗАКЛЮЧЕНИЕ	66
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	67
ПРИЛОЖЕНИЕ	69

ВВЕДЕНИЕ

В современном мире трехмерные ландшафты находят применение во многих сферах. Большие сцены с участием природных пейзажей, созданных с помощью компьютерной графики, повсеместно используются в телевидении, кинематографе, видеоиграх, в качестве демонстрационного материала для различных архитектурных проектов.

В данной работе рассматривается создание природных ландшафтов с использованием трёхмерного движка Unreal Engine 4. Среди всех инструментов, Unreal Engine выделяет ориентированность на реалистичность получаемого изображения и возможность работы «в реальном времени», не требуются длительные расчёты как при рендеринге 3д графики с помощью таких инструментов как 3ds Max, Blender и др. Это позволяет быстро привносить изменения в проекты и создавать интерактивные сцены, способные меняться в зависимости, например, от ввода пользователя.

Работа делится на четыре основные части:

1. Создание сцены малого размера со статичной камерой;
2. Создание большой ландшафтной сцены;
3. Разработка инструментов для создания водоёмов;
4. Разработка игровых персонажей;

Часть первая. Создание сцены со статичной камерой.

Статичные природные сцены могут использоваться как фоны для ведущих телевидения, интерьерные интерактивные картины, рекламные фоны. В таких сценах большую роль играет уровень детализации, ведь в случае, если зритель будет приглядываться к изображению, которое не меняется, он быстрее заметит отсутствие мелких деталей и других погрешностей.

Другим важным аспектом является освещение, зачастую именно оно определяет уровень реалистичности изображения. Грамотное использование прямого и отраженного света, атмосферной перспективы в виде тумана и других природных эффектов может позволить получить картинку, местами почти неотличимую от фотографии.

При создании таких сцен важно ориентироваться на реальные фотографии природы и анализировать их составляющие.

В рамках первой части данной работы рассмотрено создание небольшой природной сцены, а также настроено интерактивное управление погодой в сцене.

Часть вторая. Создание большой ландшафтной сцены.

Большие ландшафтные сцены могут использоваться для кинематографа и видеоигр. При создании сцен, по которым будет перемещаться камера, помимо детализации и освещения важную роль играют «наполненность сцены» и оптимизация.

Под «наполненностью» подразумевается равномерное распределение объектов, привлекающих внимание, таких как деревья, растения и камни. Потенциальному зрителю или игроку не должны встречаться области локации, выглядящие пустыми или переполненными. Такие недочёты могут заметно понизить уровень реализма.

Оптимизация – достижение приемлемой производительности на всех целевых платформах при разработке видеоигры. Для сохранения стабильно высокого уровня производительности важно использовать качественные ассеты: 3д модели, использующие оптимальное количество полигонов для нужного уровня детализации. Существует набор способов оптимизации, достигаемых программно:

Использование LOD (Level Of Detail) - Для этого необходимо на стадии моделирования создать одну или несколько дополнительных версий модели с пониженной детализацией, а затем, после импорта в движок, настроить зависимость уровня детализации от дистанции между объектом и камерой, либо от процента площади экрана, занимаемой объектом.

Тайлинг (Tiling) ландшафта – Техника, похожая на LOD, но относящаяся к 3д моделям ландшафта. Ландшафт делится на квадратные «куски», для каждого из которых создается набор LOD-ов. Правильный LOD отображается в

зависимости от расстояния между камерой и куском ландшафта. Такая техника позволяет отображать огромные ландшафты с сохранением производительности.

Использование технологий микрорельефного текстурирования – создание рельефа поверхностей без создания большого количества полигонов, а с помощью текстур. Повсеместное использование таких техник сильно повышает детализированность сцены без большого ущерба производительности.

В рамках второй части данной работы рассмотрено создание большой ландшафтной сцены с использованием различных технологий оптимизации.

Часть третья. Разработка инструментов для создания водоёмов.

Водоёмы в различных проявлениях являются неотъемлемой частью природных ландшафтов. Из-за присущей жидкостям сложности изображения как в статичном виде, так и в реальном времени, убедительно выглядящая вода может заметно повысить реалистичность всей сцены. «Убедительность» складывается как из небольших деталей – конкретных шейдеров воды, так из общих форм водоёмов и того, как они взаимодействуют с окружением.

В рамках третьей части данной работы рассмотрены различные аспекты, влияющие на правдоподобность изображения воды, созданы реалистичные шейдеры воды, а также разработаны инструменты для удобного добавления водоёмов в природные сцены.

Часть четвертая. Разработка игровых персонажей.

Создание игровых персонажей является важным аспектом разработки видеоигр, который оказывает значительное влияние на погружение пользователя и общую реалистичность продукта. Посредством игровых персонажей, пользователь может исследовать виртуальный мир. Реалистичность игровых персонажей зависит от использованных 3д моделей и анимаций, что, как правило является задачей художников, работающих над продуктом. Однако то, как анимации будут восприниматься пользователем, зависит от выборов разработчика игрового процесса.

При разработке игрового процесса необходимо интегрировать анимации с логикой игры, чтобы движения персонажа были плавными и отзывчивыми. Важно, чтобы все анимации, были синхронизированы с игровыми событиями и могли быстро реагировать на действия игрока.

В рамках четвертой части работы рассмотрен процесс настройки и создания анимаций игрового персонажа, разработка игрового процесса в виде боевой системы, демонстрирующей различные полученные анимации, создание базового искусственного интеллекта неигровых персонажей, а также настройка базовой многопользовательской игры.

Цели работы:

1. создать природную сцену со статичной камерой;
2. разработать систему управлению погодой по нажатию клавиши;
3. создать и оптимизировать природную сцену большого размера;
4. разработать инструменты для создания водоёмов;
5. создать анимированного игрового персонажа;
6. создать игровой процесс в виде боевой системы и способы её демонстрации в виде неигровых;

Для выполнения работы были использованы средства следующих программ:

- Unreal Engine 4 – создание сцен и разработка внутриигровых систем;
- Blender – создание 3d-модели ландшафта небольшой сцены;
- Adobe Substance 3D Designer – создание текстур;
- World Machine – создание ландшафтов больших размеров.

1. Создание природной сцены со статичной камерой

Процесс создания статичных сцен начинается с изучения референсов для определения общей композиции и конкретных элементов, которые будут заполнять сцену.



Рисунок 1. Фотографии-референсы, взятые за основу при создании сцены.

Для поддержания хорошего баланса детализации сцена должна содержать большие, средние и малые элементы. Основными элементами композиции будут являться деревья, валуны и лесная тропа. В качестве средних элементов будут выступать ветки и камешки, лежащие на земле, а также различные кусты, а в качестве мелких элементов будет выступать трава[14].

1.1. Создание ландшафта

Ландшафт сцены был создан с помощью программы Blender 3D [3] и импортирован в проект виде Static Mesh.

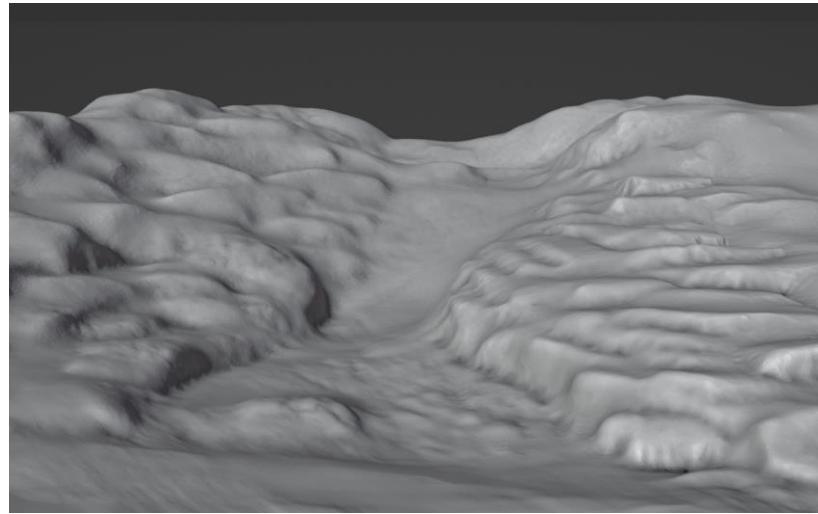


Рисунок 2. Базовая 3д модель лесной тропы

Был создан материал, состоящий из трёх «слоёв»: травы, камня и песка. Распределение слоёв было реализовано с помощью Vertex Colors: материал каждой вершины модели выбирается в соответствии с её цветом, состоящим из трёх каналов: красного, зеленого и синего. Высокие значения в красном канале отвечают за наличие травы, Высокие значения в зеленом канале отвечает за наличие материала камней, а значения, близкие к нулю в зеленом и красном канале, отвечают за наличие песка. Благодаря этому возможно распределение материалов ландшафта с помощью “раскрашивания” модели внутри редактора Unreal Engine [1].

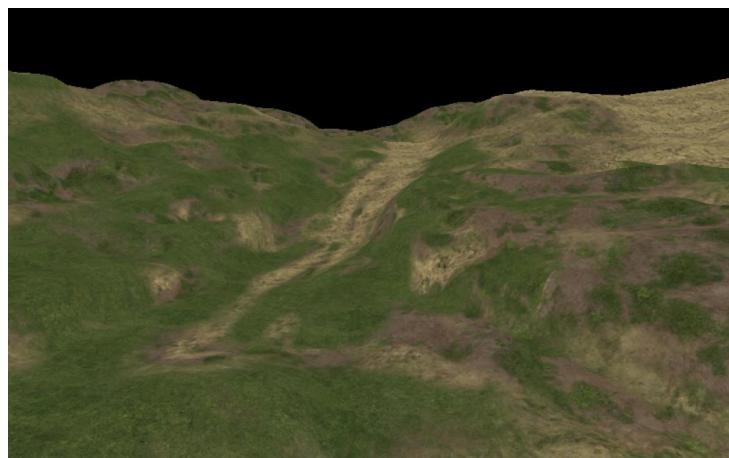


Рисунок 3. Многослойный материал примененный к модели

1.2. Заполнение сцены объектами

Основные элементы композиции – камни и деревья – были расставлены вручную. Были использованы высококачественные модели, созданные с помощью сканирования реальных объектов - Megascans.



Рисунок 4. Основные элементы композиции

Инструмент Foliage Tool позволяет размещать в сцене большое количество одинаковых моделей, таких как пучки травы, деревья и камни, при этом производительность сцены гораздо выше, чем в случае, если бы все модели были размещены вручную. Это достигается за счёт технологии Static Mesh Instancing [2][7]. С помощью инструмента Foliage Tool были расставлены модели травы, упавших веток, маленьких камней и различных ростков[13].



Рисунок 5. Озеленение сцены

1.3. Настройка освещения и погодных условий

Освещение сцены состоит из Directional Light – источника прямого света, имитирующего солнце, Sky Light – источника рассеянного света, и Exponential Height Fog – объекта, создающего эффект тумана.



Рисунок 6. Сцена при ясной погоде

Эффект дождя был создан при помощи повышения плотности тумана и увеличения отражающей способности материалов (Specular) для создания эффекта влажной поверхности. Частицы дождя были созданы с помощью системы частиц Niagara FX. Визуальный эффект капель, стекающих по стеклу, был реализован с помощью Post-Process Material.

Для возможности переключения погоды по нажатию клавиши был создан класс *AWeatherHandler*. Функции *AWeatherHandler::SetClearWeather()* и *AWeatherHandler::SetRainWeather()* отвечают за установку ясной и дождливой погоды соответственно. Данные функции представлены в листингах 1 и 2.



Рисунок 7. Сцена при дождливой погоде

Листинг 1. Функция *SetClearWeather()*, отвечающая за переключение погоды на ясную.

```
void AWeatherHandler::SetClearWeather()
{
    UE_LOG(LogTemp, Warning, TEXT("Set Clear Weather"));
    for (int Index = 0; Index < DirectionalLights.Num(); ++Index)
    {
        ULightComponent* DirectionalLightComponent = DirectionalLights[Index]-
>FindComponentByClass<ULightComponent>();
        DirectionalLightComponent-
>SetIntensity(DirectinalLightIntensityClear);
        DirectionalLights[Index]->SetActorRotation(DirectinalLightAngleClear);
    }
    for (UNiagaraComponent* Component : RainFX)
    {
        Component->Deactivate();
    }
    SkyLight->SetIntensity(SkyLightIntensityClear);

    for (const AActor* it : WetActors)
    {
        UStaticMeshComponent* SMC = it-
>FindComponentByClass<UStaticMeshComponent>();
        UMaterialInstanceDynamic* Material = (UMaterialInstanceDynamic*)SMC-
>GetMaterial(0);
        Material->SetScalarParameterValue(FName(TEXT("Wetness")), 0);
    }
    Fog->SetFogDensity(FogDensityClear);
    FPostProcessSettings& PostProcessSettings = PPV->Settings;
    if (TestMatIns) {
        TestMatInsDyna =
UKismetMaterialLibrary::CreateDynamicMaterialInstance(this, TestMatIns);
        FWeightedBlendable WeightedBlendable;
        WeightedBlendable.Object = TestMatInsDyna;
        WeightedBlendable.Weight = 0;
        PostProcessSettings.WeightedBlendables.Array.Empty();
        PostProcessSettings.WeightedBlendables.Array.Add(WeightedBlendable);
    }
}
```

Листинг 2. Функция *SetRainWeather()*, отвечающая за переключение погоды на дождливую.

```
void AWeatherHandler::SetRainWeather()
{
    UE_LOG(LogTemp, Warning, TEXT("Set Rain Weather"));
    for (int Index = 0; Index < DirectionalLights.Num(); Index++) {
        ULightComponent* DirectionalLightComponent =
DirectionalLights[Index]->FindComponentByClass<ULightComponent>();
        DirectionalLightComponent-
>SetIntensity(DirectinalLightIntensityRain);
        DirectionalLights[Index]-
>SetActorRotation(DirectinalLightAngleRain);

        for (AActor* it : WetActors) {
            UStaticMeshComponent* SMC = it-
>FindComponentByClass<UStaticMeshComponent>();
            UMaterialInstanceDynamic* Material =
(UMaterialInstanceDynamic*)SMC->GetMaterial(0);
            Material->SetScalarParameterValue(FName(TEXT("Wetness")), 1);
        }
    }

    for (UNiagaraComponent* Component : RainFX) {
        Component->Activate();
    }

    SkyLight->SetIntensity(SkyLightIntensityRain);
    Fog->SetFogDensity(FogDensityRain);

    FPostProcessSettings& PostProcessSettings = PPV->Settings;

    if (TestMatIns) {
        TestMatInsDyna =
UKismetMaterialLibrary::CreateDynamicMaterialInstance(this, TestMatIns);
        FWeightedBlendable WeightedBlendable;
        WeightedBlendable.Object = TestMatInsDyna;
        WeightedBlendable.Weight = 1;
        PostProcessSettings.WeightedBlendables.Array.Empty();

        PostProcessSettings.WeightedBlendables.Array.Add(WeightedBlendable);
    }
}
```

2. Создание большой сцены с природным ландшафтом

Сцена будет состоять из скалистого острова, окруженного водой. Перед созданием ландшафта были изучены фото-референсы.



Рисунок 8. Фотографии-референсы, взятые за основу при создании сцены.

2.1 Создание карты местности

Ландшафт сцены с островом был создан в программе World Machine, позволяющей создавать реалистичные пейзажи с помощью таких техник как наложение различных шумов (шум Перлина, Симплексный шум) друг на друга и эрозия грунта ветром и водой [12].

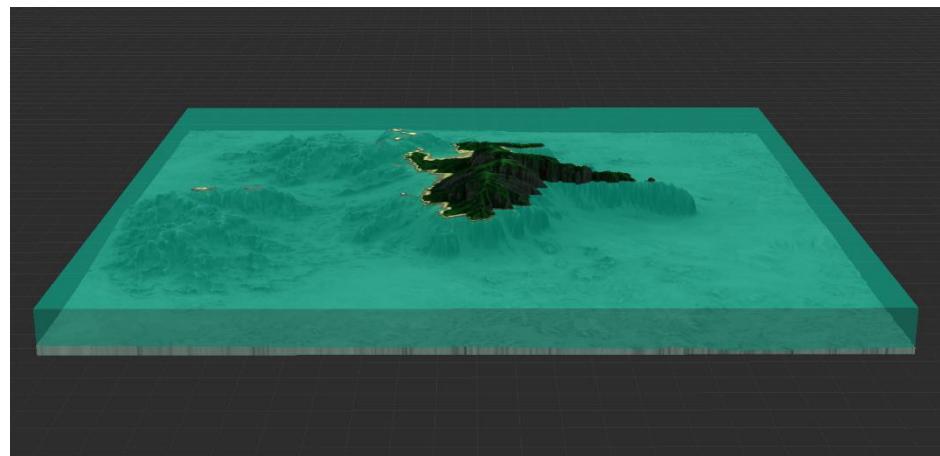


Рисунок 9. Ландшафт, созданный в World Machine

Ландшафт был разбит на 16x16 карт высот 253x253 пикселей каждая для представления в виде тайлов (квадратных “кусков” ландшафта) в World Composition. World Composition позволяет работать с большими сценами, показывая близкие к наблюдателю тайлы в высоком качестве и далёкие от него

в низком качестве [5][6]. Это позволяет увеличивать размеры сцены без потери производительности.

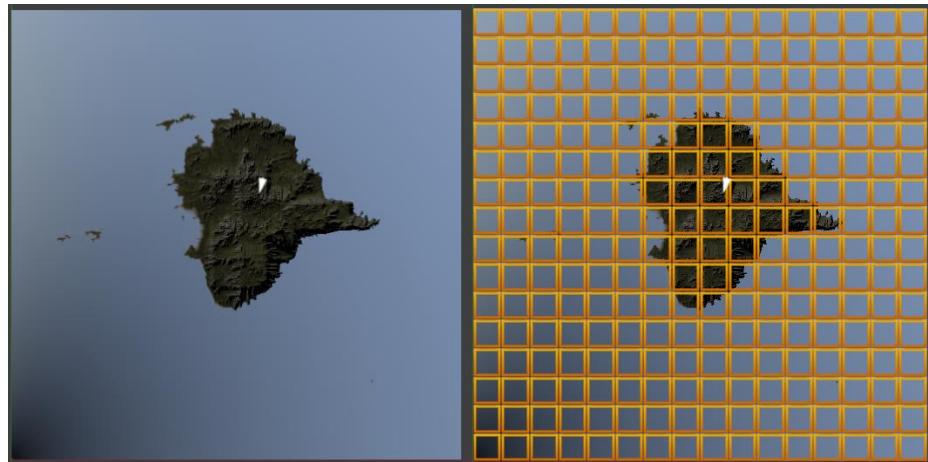


Рисунок 10. Представление ландшафта в World Composition

2.2 Создание материала ландшафта

Материал ландшафта был создан с использованием технологий Landscape Layers и Triplanar Mapping (box mapping) [4].

Landscape Layers – технология, позволяющая хранить информацию о различных «слоях» ландшафта, что даёт возможность «раскрашивать» местность разными типами поверхности. В данном случае были использованы следующие слои:

- Скалы
- Песок
- Зеленая трава
- Выцветшая трава
- Каменистое дно водоёмов

Техника Triplanar Mapping позволяет избегать «растягивания» текстур на отвесных гранях и заключается в интерполяции между проекциями текстуры в зависимости от направления поверхности. Разработанная для этого Material Function представлена на Рисунке 11.

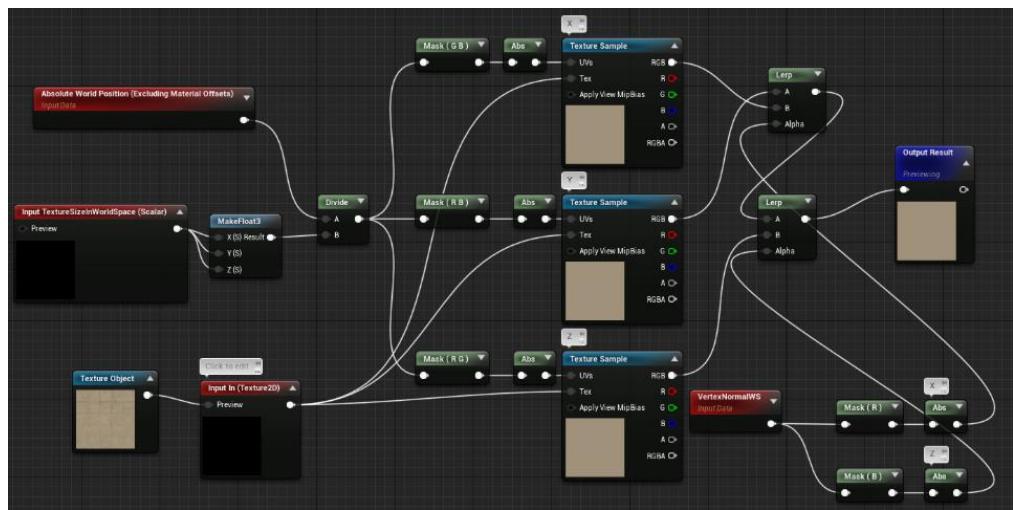


Рисунок 11. Material function для Triplanar Mapping



а)

б)

Рисунок 12. Материал ландшафта а) без использования triplanar mapping
б) с использованием triplanar mapping



Рисунок 13. Ландшафт, импортированный в Unreal Engine

2.3 Добавление растительности в сцену

Технология Landscape Grass позволяет размещать различные типы травы на пейзаж и управлять ими из материала ландшафта. На сцену были добавлены несколько видов полевой травы с помощью Landscape Grass [2][8].

Использованные материалы травы используют Vertex Animation, анимации, заключающейся в изменении позиций индивидуальных вершин модели внутри шейдера. Это позволяет получить эффект растительности, раскаивающейся на ветру. Скорость, турбулентность и другие параметры ветра настраиваются в материалах растений.



Рисунок 14. Трава, созданная с помощью Landscape Grass

Технология Procedural Foliage Spawner позволяет управлять «посадкой» растительности: для каждого вида растений задается плотность размещения, время роста, допустимая близость к другим объектам и другие параметры, позволяющие создать натуральный вид. С помощью Procedural Foliage Spawner по всей карте были размещены четыре разновидности елей. Важно использовать отличающиеся модели деревьев для создания эффекта разнообразия растительности и избежать однообразности ландшафта.



Рисунок 15. Различные модели деревьев, использованные в сцене



Рисунок 16. Результат работы Procedural Foliage Spawner

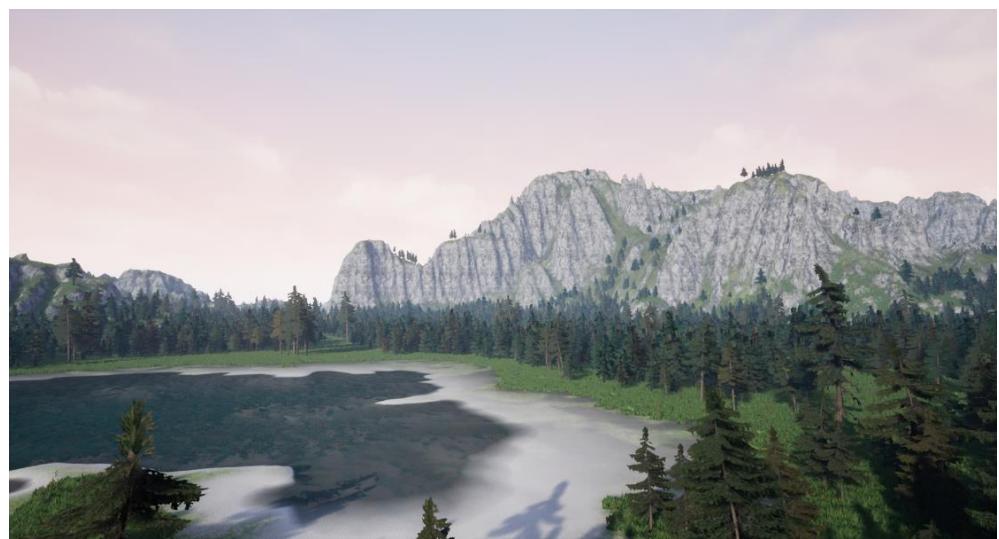


Рисунок 17. Сцена в финальном виде

2.4 Добавление деталей

На горе была создана поляна с цветами и тропинкой, размещена модель деревянного дома. Эффект «вытоптанной» тропинки был реализован с помощью раскрашивания травы в цвет грунта и уменьшения её размеров. Для передачи информации о положении тропинки использована технология Virtual Texture. Она заключается в создании текстуры, доступ к которой имеют различные объекты сцены: материал ландшафта записывает в текстуру данные о расположении тропинки, а материал растительности считывает их для определения высоты и цвета травы.



Рисунок 18. «Вытоптанная» тропинка, ведущая к дому

Конечный вариант сцены с настроенным освещением представлен на рисунке 19.



Рисунок 19. Сцена с домом на горе.

2.5 Обзор техник рельефного текстурирования

Большие элементы 3д моделей отображаются с помощью набора полигонов, однако использовать полигоны для отображения рельефа может быть очень невыгодно с точки зрения производительности программы.

Мелкие детали такие как морщины на коже или трещины в камне могут быть изображены с помощью 2д текстуры, наложенной на модель. Но такой подход может выглядеть недостаточно убедительно, при приближении к поверхности будет заметно что все детали на самом деле плоские, а при изменении угла освещения тени на деталях не будут изменяться. Для достижения лучшего результата существует ряд технологий **рельефного текстурирования**.

Для демонстрации различных технологий в программе Substance 3D Designer был разработан материал речных камней. Данный материал содержит большое количество мелких деталей, которые необходимо убедительно отобразить.

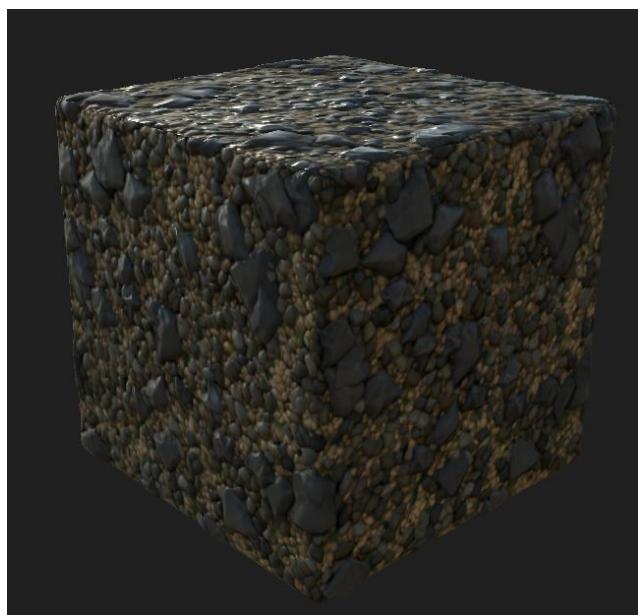


Рисунок 20. Демонстрация материала, разработанного в Substance 3D Designer

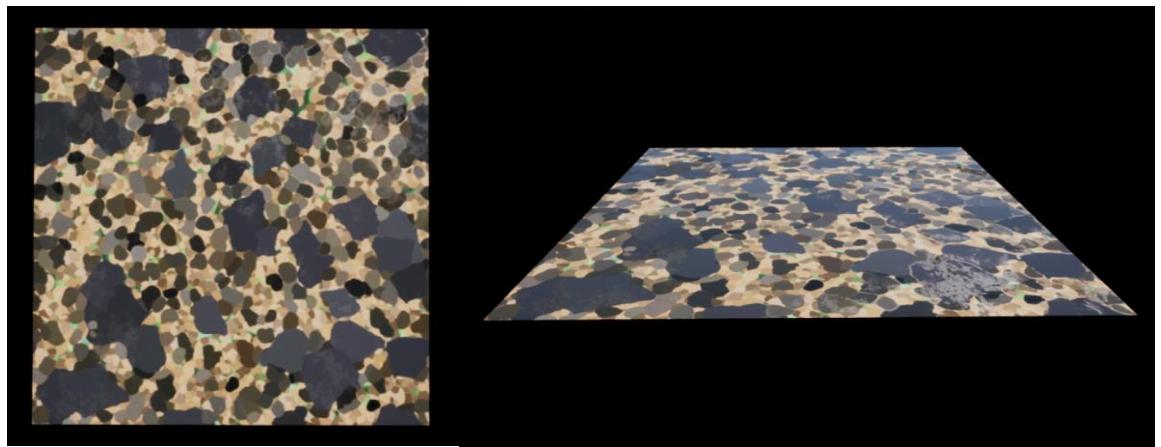


Рисунок 21. Базовое наложение текстуры без использования технологий рельефного текстурирования

Bump Mapping

Технология bump mapping заключается в использовании текстуры, в которую закодирована карта высот микрорельефа, для затенения определенных элементов модели, таких как трещины и щели.

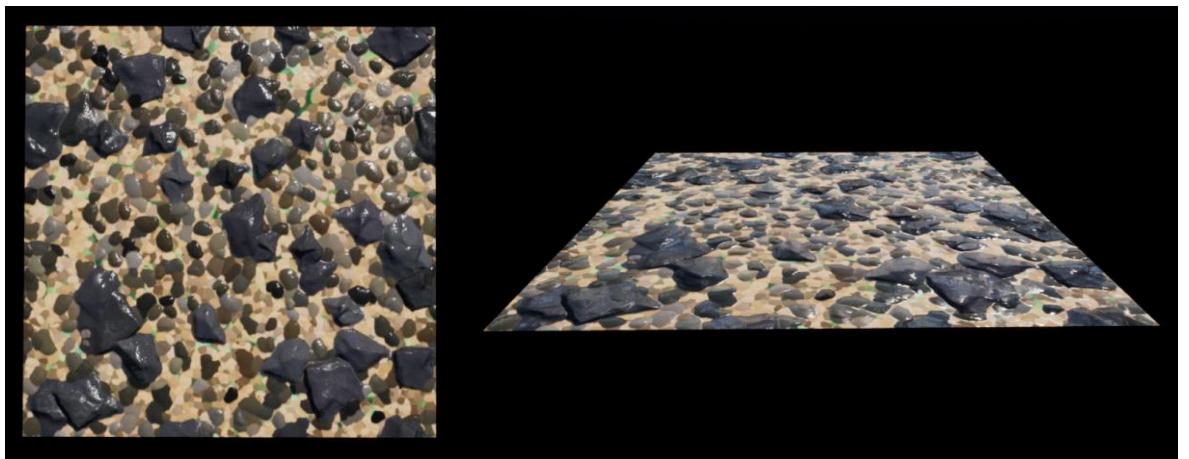


Рисунок 22. Материал, использующий bump mapping.

Normal Mapping

Normal Mapping – технология, дающая результат похожий на bump mapping, но гораздо более распространённая. Вместо информации о высоте рельефа используется информации об нормалях поверхности, позволяющая управлять отражением падающего света. Normal mapping имеет заметный эффект на блестящих материалах с сильным рефлексом.

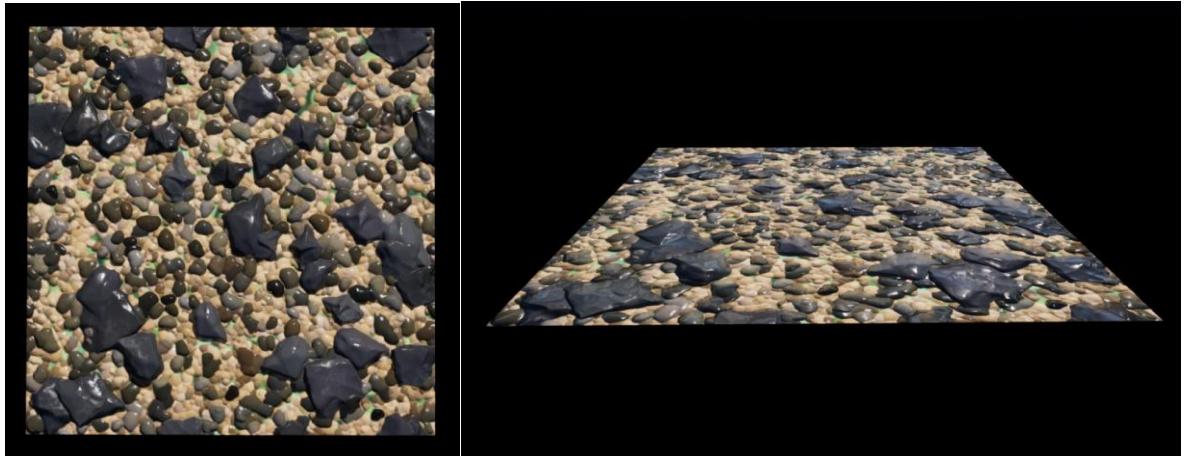


Рисунок 23. Материал, использующий normal mapping

Bump offset

Bump offset позволяет в некоторой степени передать перекрытие выступающими элементами материала более плоских участков, добавляя глубины.

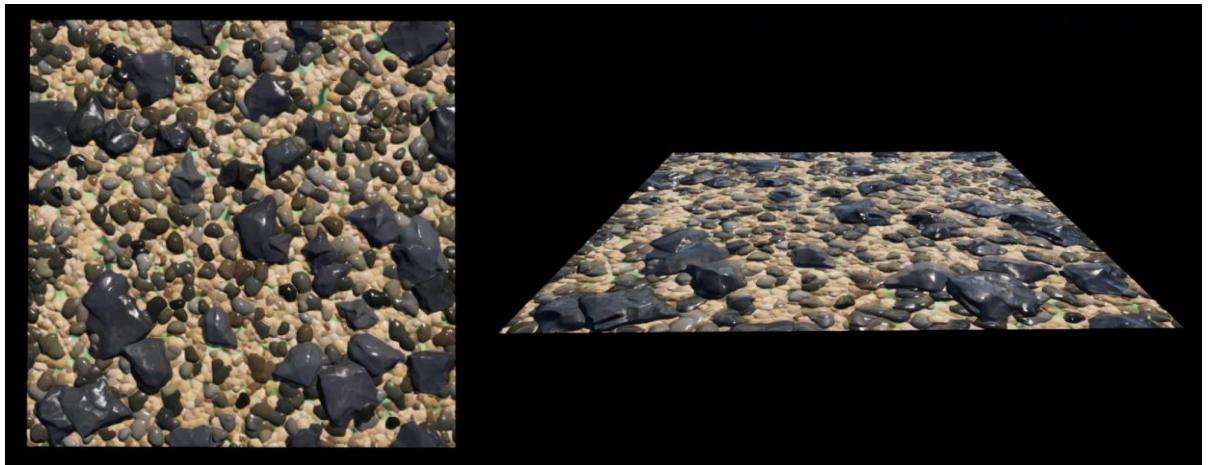


Рисунок 24. Материал, использующий normal mapping и bump offset

Parallax occlusion mapping

Фактически представляет собой форму локальной трассировки лучей в пикельном шейдере. Трассировка лучей используется для определения высот и учёта видимости пикселей. Иными словами, данный метод позволяет создавать ещё большую глубину рельефа при небольших затратах полигонов и применении сложной геометрии. Недостаток метода — невысокая детализация силуэтов и граней.

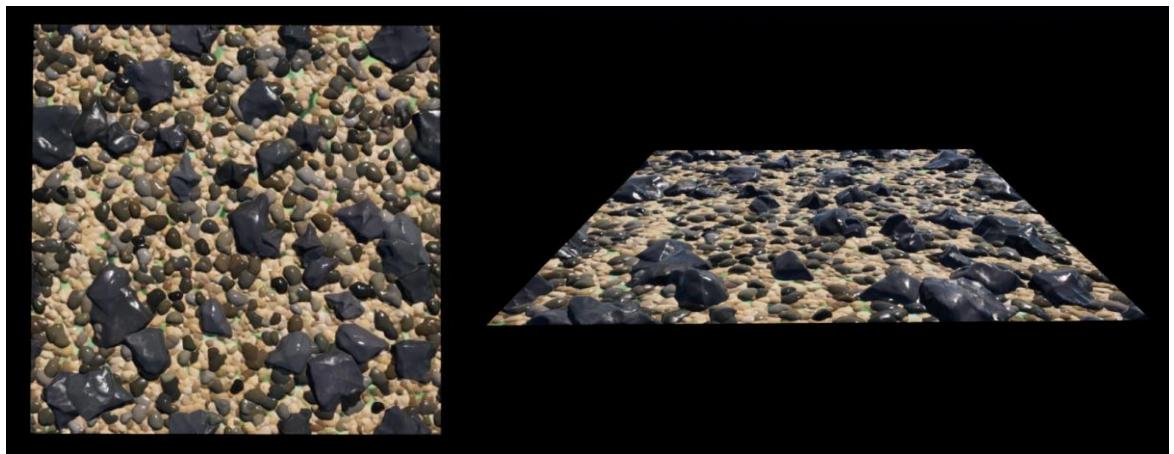


Рисунок 25. Материал, использующий normal mapping и parallax occlusion mapping

Displacement mapping

Displacement mapping заключается в изменении геометрии поверхности по карте высот. В отличии от вышеупомянутых технологий результат не «плоский» и микрорельеф имеет реальный объёмный вид.

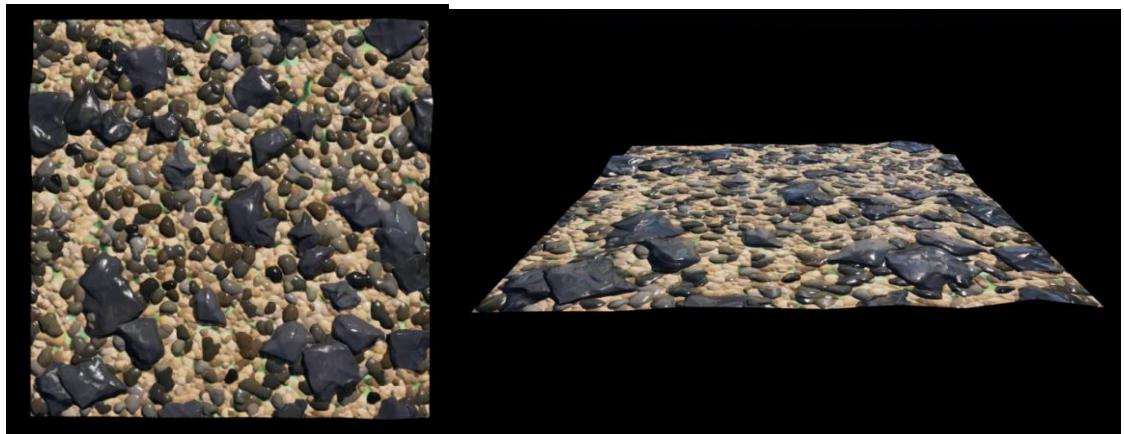
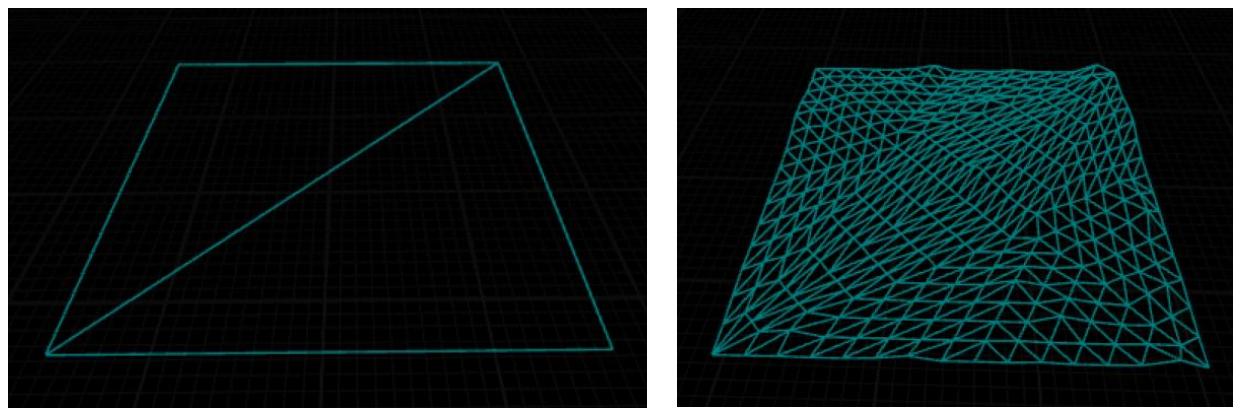


Рисунок 26. Материал, использующий normal mapping и displacement mapping.

Tessellation

Tessellation (замощение, тесселяция) – техника автоматизированного добавления новой геометрии с целью повышения детализации 3д модели. На рисунке 5 демонстрируется тесселяция модели, состоящей из двух полигонов.



а)

б)

Рисунок 27. а) Материал, не использующий Tessellation

б) Материал, использующий Tessellation

Тесселяция позволяет использовать displacement mapping с более заметным эффектом: при приближении камеры к объекту увеличивается детализация сетки, тем самым создавая «реальный» объёмный рельеф.

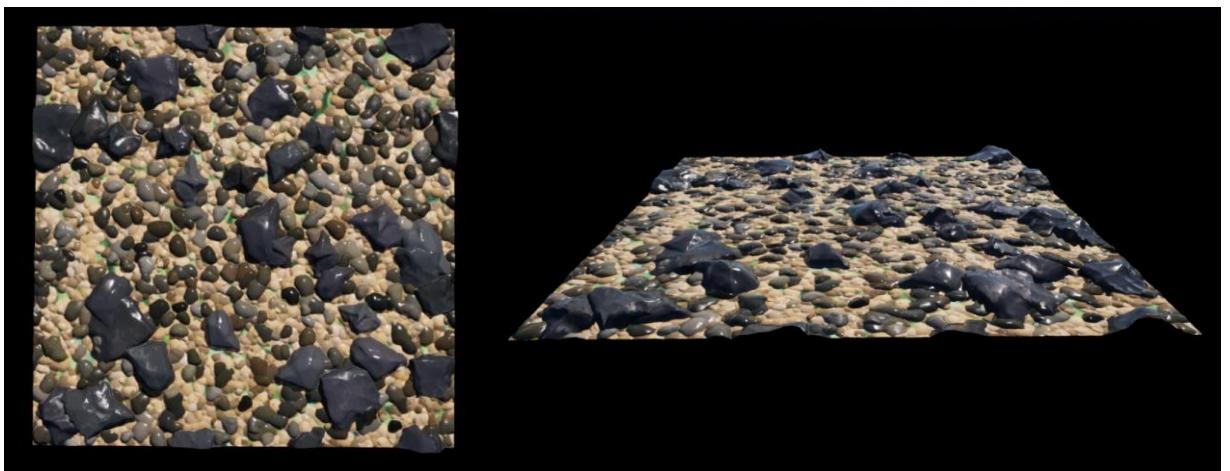


Рисунок 28. Материал, использующий normal mapping, displacement mapping

и tessellation.

2.5.1 Измерение производительности различных методов рельефного текстурирования

Был проведен тест по измерению зависимости производительности программы от выбранного метода рельефного текстурирования.

В рамках теста была создана пустая сцена содержащая в себе модель дизайнерского стула, состоящую из около 200 тысяч полигонов.

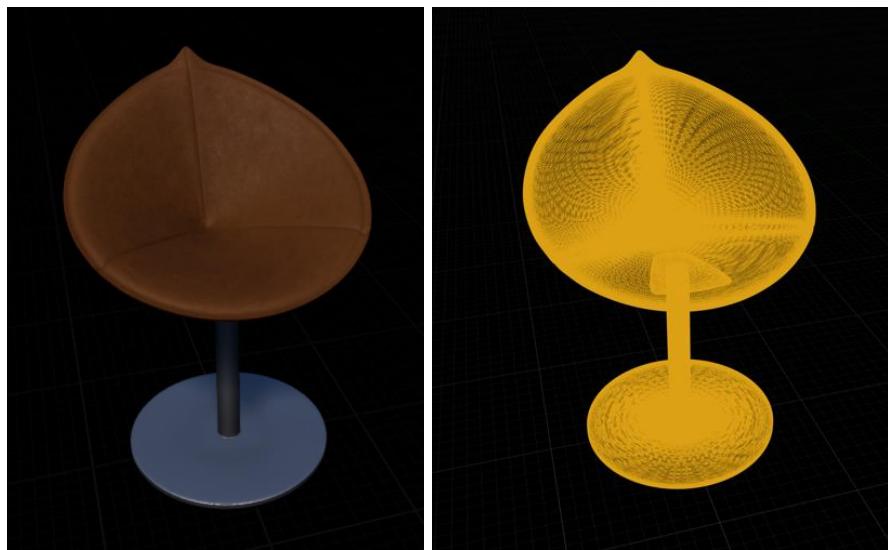


Рисунок 29. Высокополигональная модель стула

Для каждого из методов рельефного текстурирования был создан соответствующий материал обивки кресла и был произведен замер производительности. На рисунке 20 представлен график зависимости производительности сцены от плотности сетки.

Метод	FPS	FPS, %
Без метода	146	100
Bump map	140	95,89041
Normal map	140	95,89041
Bump offset	133	91,09589
Parallax occlusion	130	89,0411
Displacement	125	85,61644
Displacement+Tessellation	60	41,09589

Таблица 1. Измерения производительности

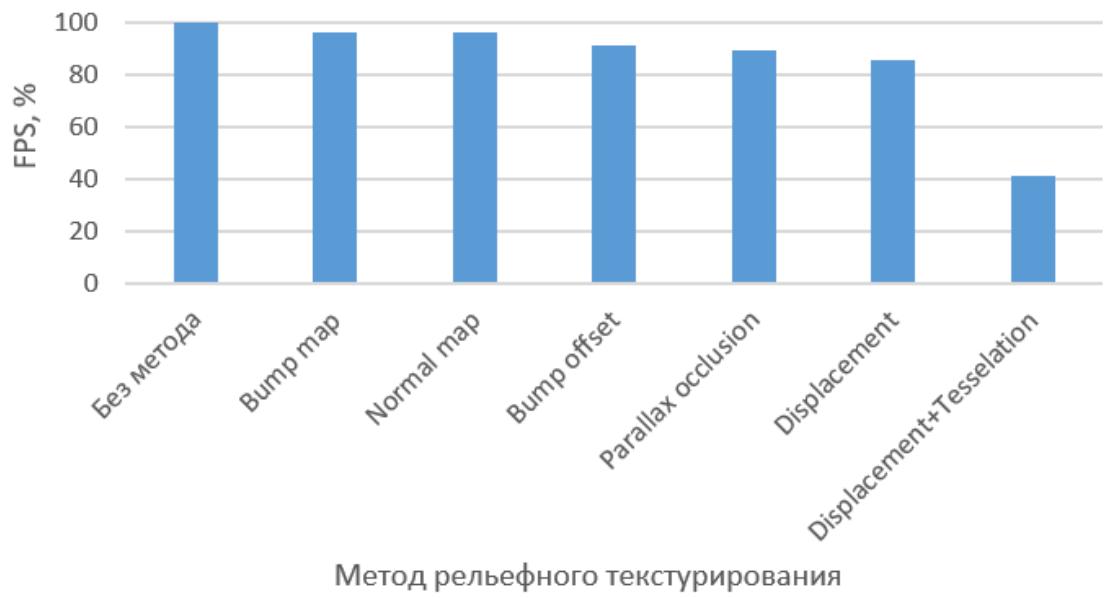


Рисунок 30. Зависимость количества кадров в секунду от выбранного метода рельефного текстурирования.

По итогам теста был сделан вывод: при использовании методов Bump Mapping и Normal Mapping нет значительной потери производительности. Bump Offset и Parallax Occlusion Mapping, так же приводят к небольшим потерям производительности. При сравнивать потерей производительности и визуальных результатов, которые достигается за их счёт, можно прийти к выводу что использование методов рельефного текстурирования эффективно.

Характеристики компьютера, на котором проводились замеры:

Процессор 12th Gen Intel Core i5-12600;

Графический процессор NVIDIA GeForce GTX 1050 Ti, 32 Гб ОЗУ.

2.6 Измерение производительности сцен с ландшафтом

Был проведен тест по измерению зависимости производительности программы от плотности сетки ландшафта.

В рамках теста в программе World Machine был создан несложный ландшафт размером 4x4 км и экспортирован в виде карты высот в следующих разрешениях: 8129x8129, 4033x4033, 2017x2017, 1009x1009, 505x505, 253x253 и 127x127 пикселей.

Для каждой из плотностей сетки был произведен замер производительности. На рисунке 20 представлен график зависимости производительности сцены от плотности сетки.

Количество вершин	Плотность сетки, пиксель/метр	FPS
66 080 641 (8129)	2.03	68
16 265 089	1.01	126
4 068 289	0.50	134
1 1018 081	0.25	135
255 025	0.06	140
16 129	0.03	141

Таблица 1. Измерения производительности

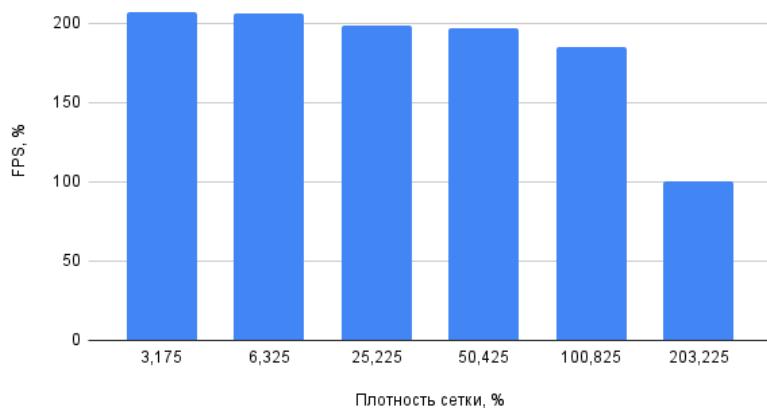


Рисунок 31. Зависимость количества кадров в секунду от плотности сетки ландшафта.

Внешний вид ландшафтов с различными плотностями карты высот представлен на рисунке 31.

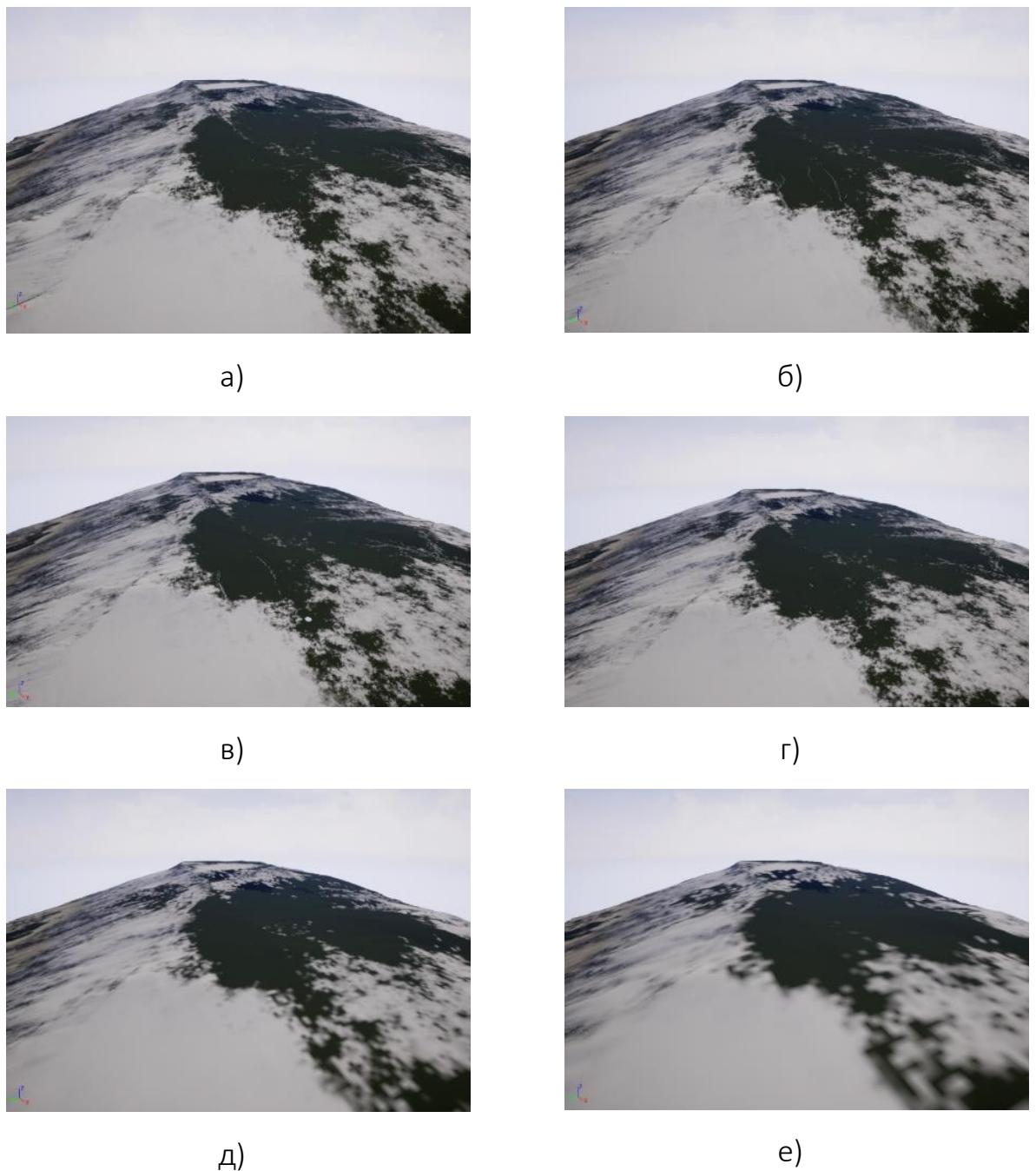


Рисунок 32. Внешний вид ландшафта при плотностях карты высот
 а) 2.03225 б) 1.00825 в) 0.50425 г) 0.25225 д) 0.06325 е) 0.03175
 пикселей на метр.

По итогам теста был сделан вывод: при использовании ландшафта с плотностью карты высот 0.5 пикселя на метр нет значительной потери ни производительности, ни качества изображения.

Характеристики компьютера, на котором проводились замеры:

Процессор 12th Gen Intel Core i5-12600;

Графический процессор NVIDIA GeForce GTX 1050 Ti, 32 Гб ОЗУ.

3. Разработка инструментов для создания водоёмов

Различные водоёмы, такие как реки, озёра, моря, являются неотъемлемой частью многих пейзажей, поэтому важно понимать техники создания и внедрения водоёмов в природные сцены.

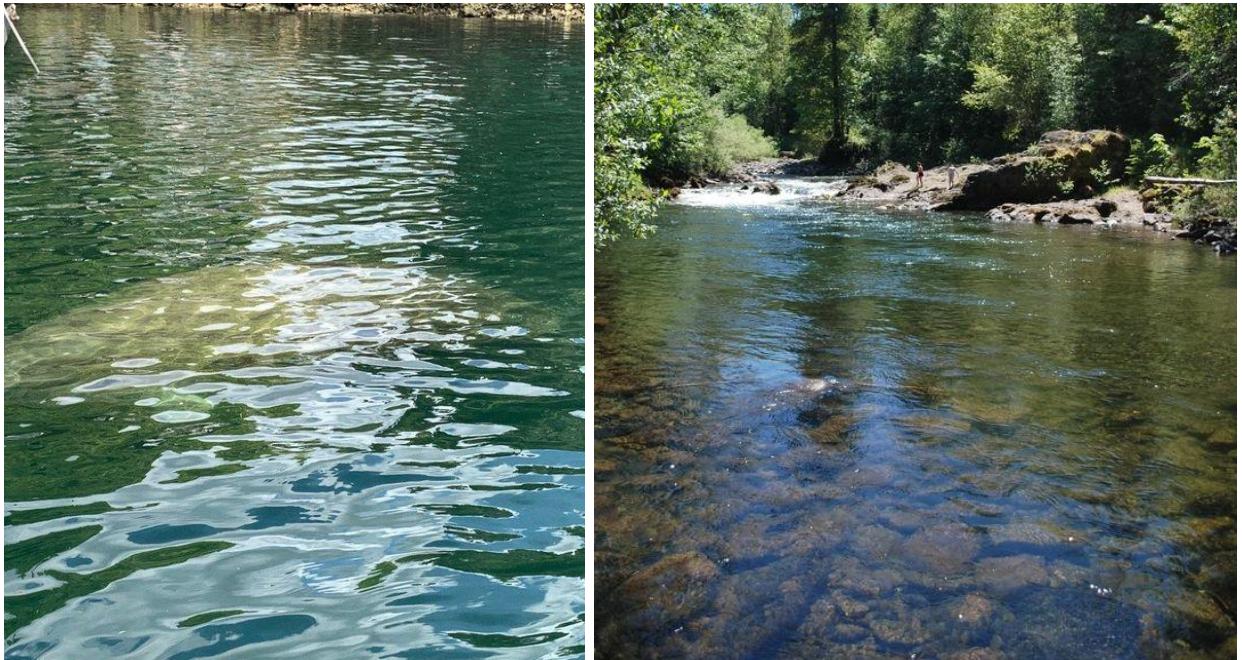


Рисунок 33. Фотографии-референсы, взятые за основу при создании материалов воды.

3.1 Разработка шейдера речной воды

После анализа некоторого количества референсов водоёмов был выделен ряд важных деталей:

- Двигающаяся рябь на поверхности воды
- Рефракция света
- Поглощение света, зависящее от глубины
- Отражение света в зависимости от угла, под которым наблюдается вода
- Пена, появляющаяся на границах воды и в местах резкого изменения направления течения

3.1.1 Предварительное создание материала воды

Был создан материал с параметрами *Blend Mode: Translucent* и *Refraction Mode: Pixel Normal Offset*[18]. Первый параметр позволяет управлять прозрачностью материала с помощью свойства *Opacity*, второй отвечает за тип рефракции.

В Unreal Engine 4 присутствуют два типа рефракции в прозрачных материалах: *Index Of Refraction* и *Pixel Normal Offset*.

Index Of Refraction вычисляет отклонение лучей по физическим законам преломления в зависимости от индекса отражения материала (IOR), эта техника подходит для небольших объектов, таких как посуды, стекол и других изогнутых поверхностей. Однако при использовании с большими объектами, такими как водоёмы, может иметь непредсказуемые результаты и вызывать визуальные артефакты.

Pixel Normal Offset создаёт иллюзию физического преломления, используя разницу между реальной нормалью поверхности, и нормалью поверхности, вычисленной, например, с помощью карт нормалей (см. *Normal Mapping*). Результат не обладает большой реалистичностью, но хорошо подходит для больших объектов, и объектов, чьи нормали быстро изменяются, таких как водоёмы с волнами. Именно этот тип рефракции использован в данном материале.

Для создания отражений на воде был использован параметр *Screen Space Reflections: On* и *Lighting Mode: Surface Translucency Volume*.

Screen Space Reflections (SSR) – это метод повторного использования данных экрана для расчета отражений. SSR — дорогостоящий метод с точки зрения вычислений, но при правильном использовании он может дать отличные результаты.

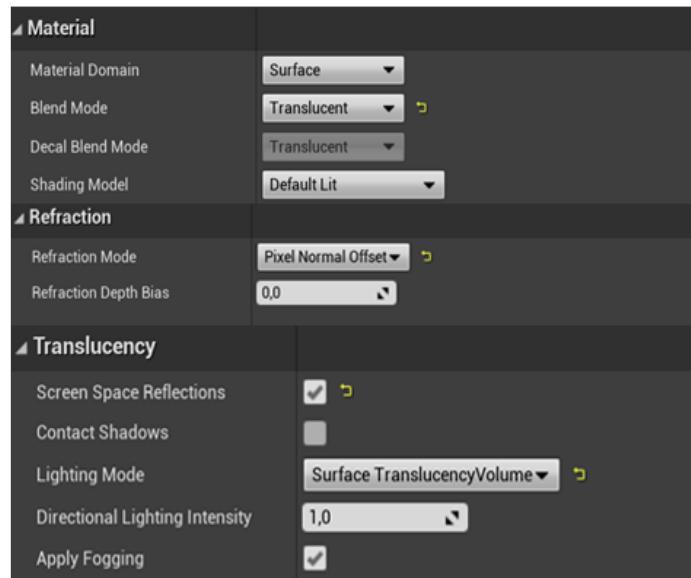


Рисунок 34. Параметры материала воды.

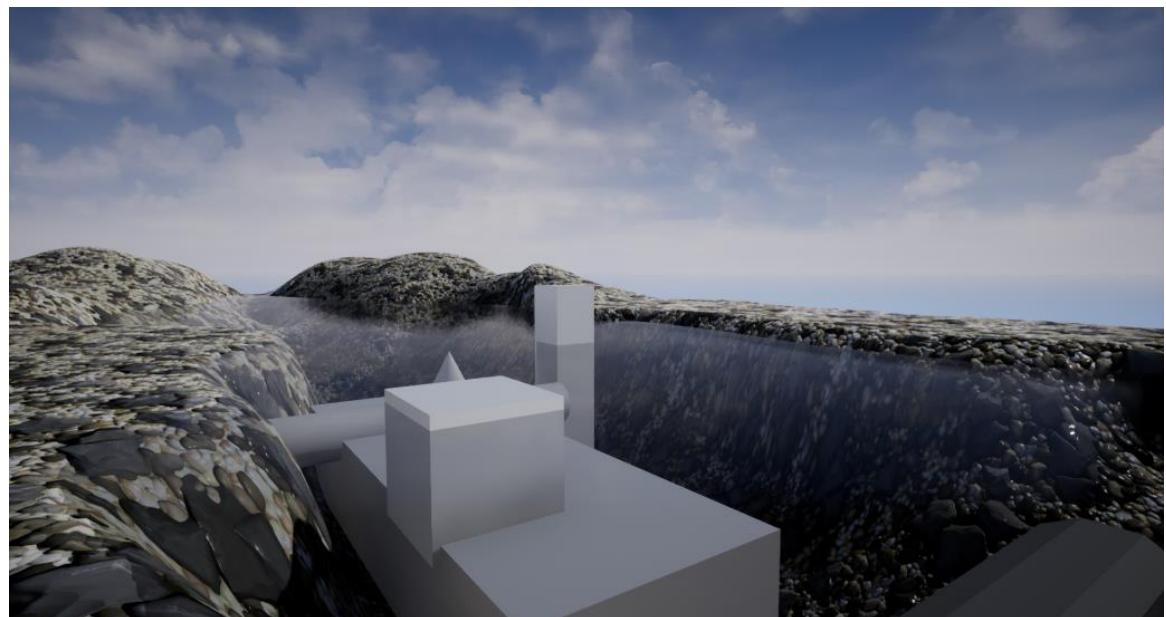


Рисунок 35. Заготовка для материала воды в тестовой сцене

3.1.2 Реализация ряби на поверхности воды

Для создания ряби была выбрана техника Scrolling Textures – «прокрутка текстур», заключающаяся, в данном случае, в использовании движущихся карт нормалей. Для удобного управления прокруткой текстур была создана функция (material function) MF_DirectionalPanner.

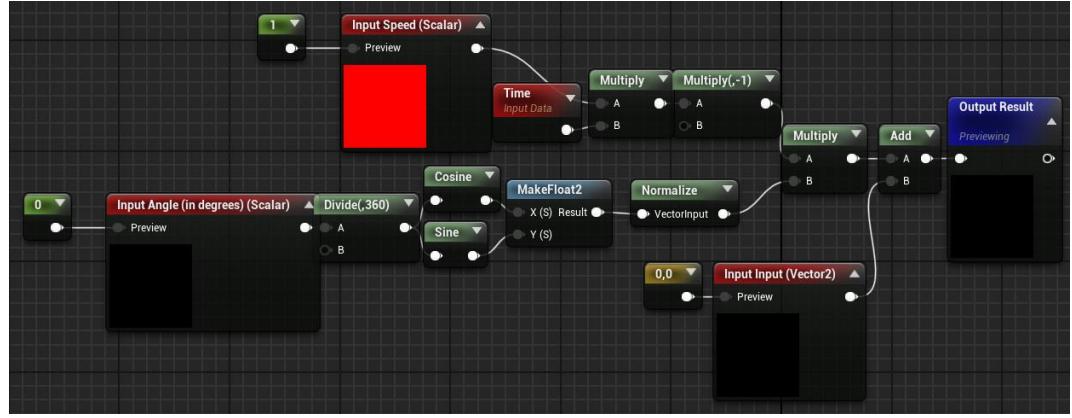


Рисунок 36. Функция MF_DirectionalPanner.

Для более реалистичного и «хаотичного» внешнего вида ряби были использованы три карты нормалей,двигающиеся с разными скоростями в разных направлениях.

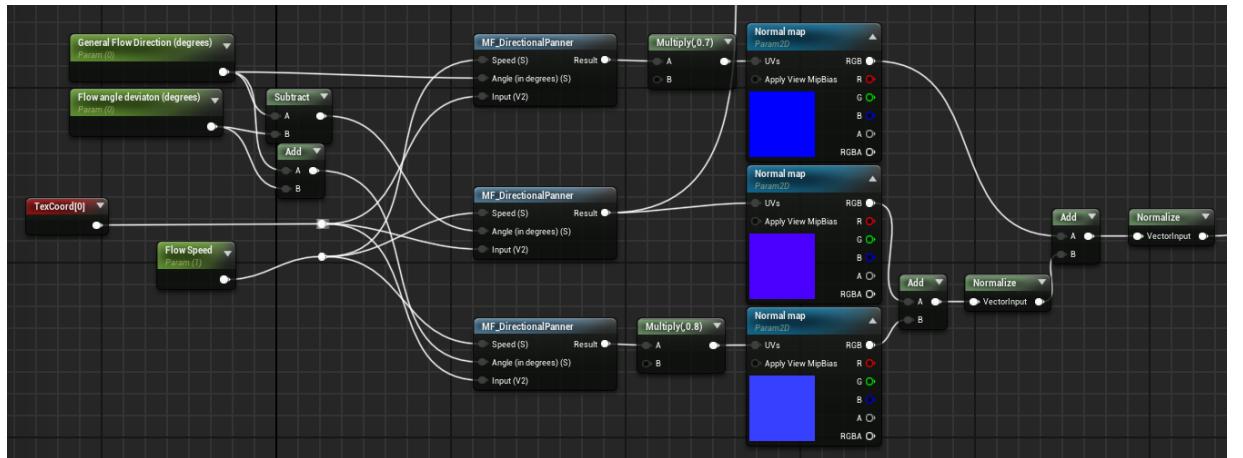


Рисунок 37. Часть шейдера, отвечающая за прокрутку карт нормалей.

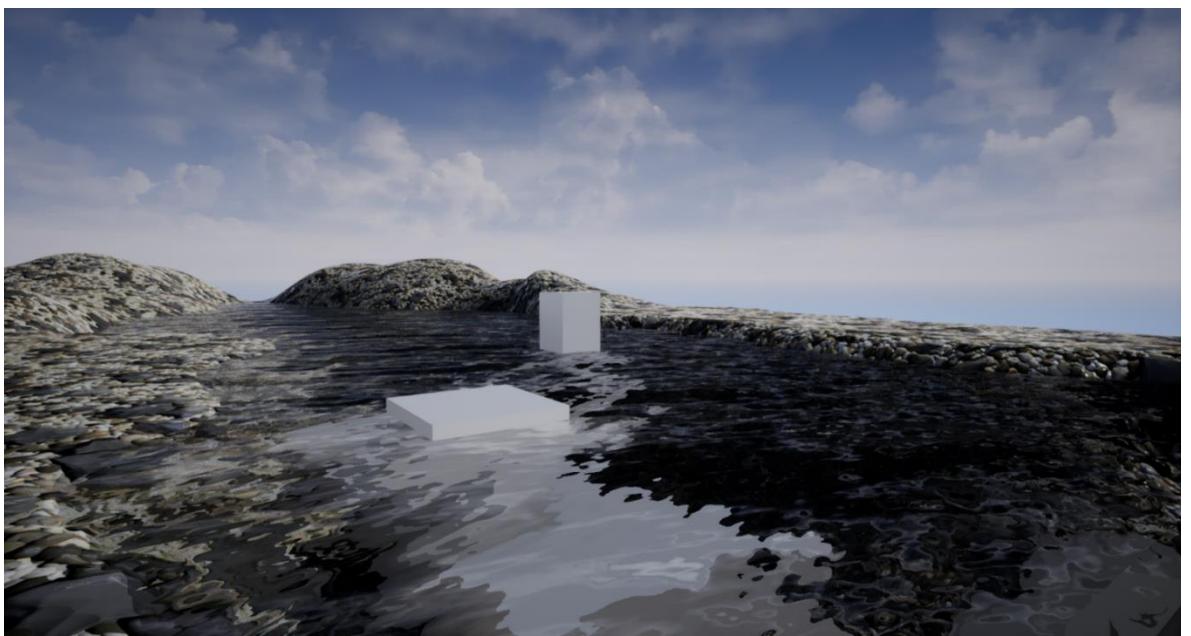


Рисунок 38. Рябь на поверхности воды.

3.1.3 Реализация изменения цвета в зависимости от глубины

Одним из способов определения глубины воды может быть сравнение расстояний от камеры до дна с расстоянием от камеры до поверхности воды. В таком случае глубина находится по формуле:

$Depth = Scene\ Depth - Pixel\ Depth$, где Scene Depth – расстояние от камеры до дна, а Pixel Depth – расстояние от камеры до поверхности.

Это «дешёвая» техника, но не такая реалистичная, ведь как правило освещенность воды меняется в зависимости от расстояния от дна до поверхности воды а не до наблюдателя.

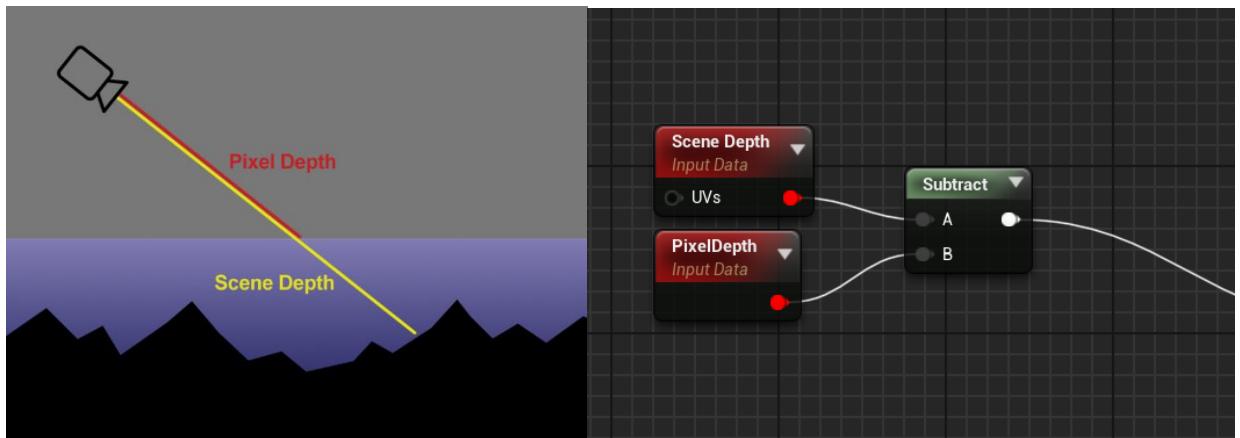


Рисунок 39. Схема и реализация базового определения глубины воды

Для более реалистичного результата можно находить вертикальное расстояние от дна до поверхности воды, тогда формула для глубины будет следующей:

$Depth = (Absolute\ World\ Position - Camera\ Position) * (Scene\ Depth / Pixel\ Depth) - Water\ Level$, где Absolute World Position – положение отрисовываемого пикселя в абсолютной системе координат, Camera Position – положение камеры в абсолютной системе координат, Scene Depth – расстояние от камеры до дна, Pixel Depth – расстояние от камеры до поверхности, и Water Level – положение поверхности воды в абсолютной системе координат.

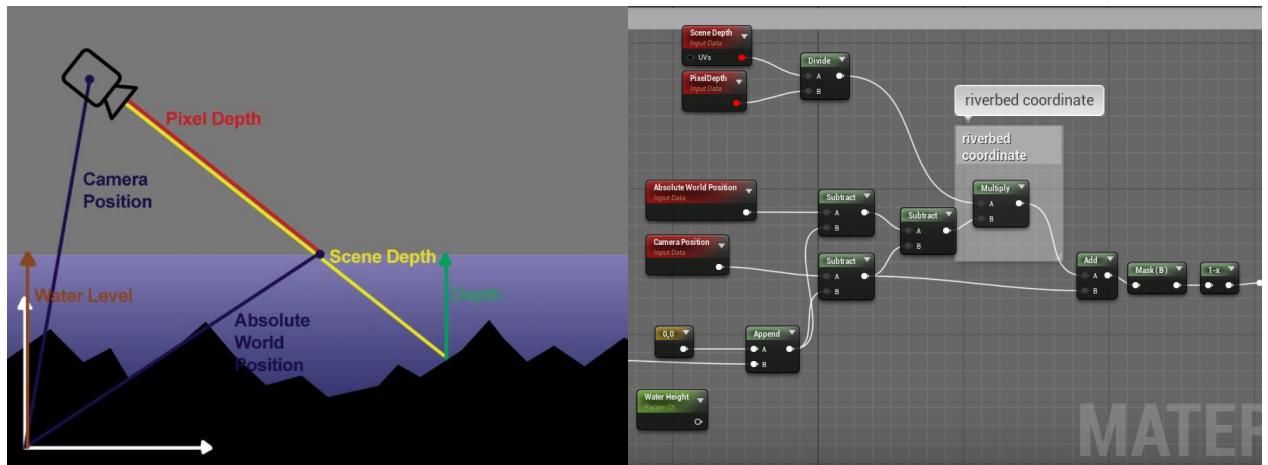


Рисунок 40. Схема определения расстояния от поверхности воды до дна

Такая схема приводит к гораздо более реалистичному результату, но может быть использована только на относительно горизонтальных поверхностях, так как необходимо знать высоту поверхности в каждой её точке.

Теперь, с использованием одной из двух схем можно реализовать изменение цвета и прозрачности в зависимости от глубины. В данном случае цвет будет терять яркость по мере увеличения, вместе с уменьшением прозрачности.

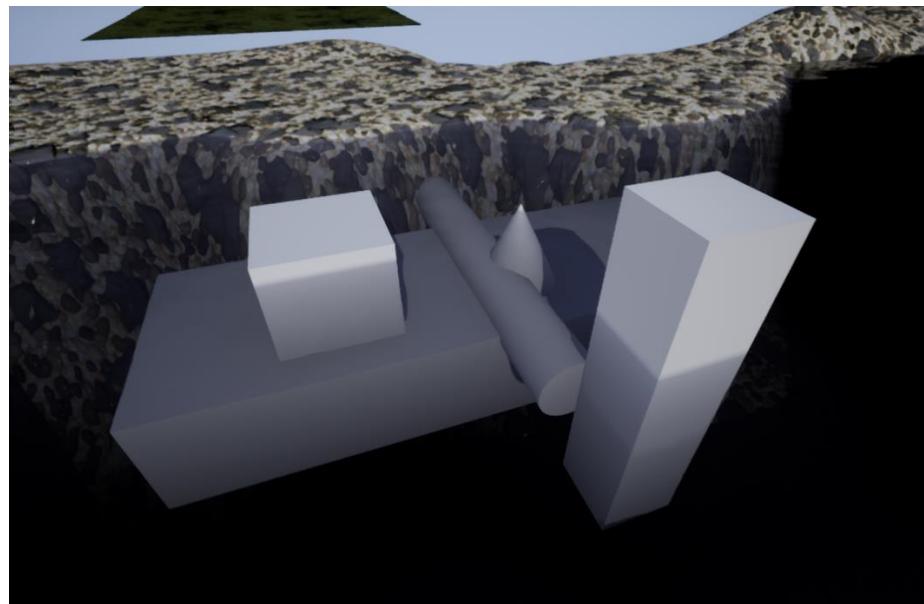


Рисунок 41. Зависимость цвета и прозрачности воды от глубины

3.1.4 Реализация эффекта полного отражения света при низких углах падения.

С помощью функции Fresnel реализуется полное отражение света при низких углах падения: материал становится полностью непрозрачным и полностью отражающим.

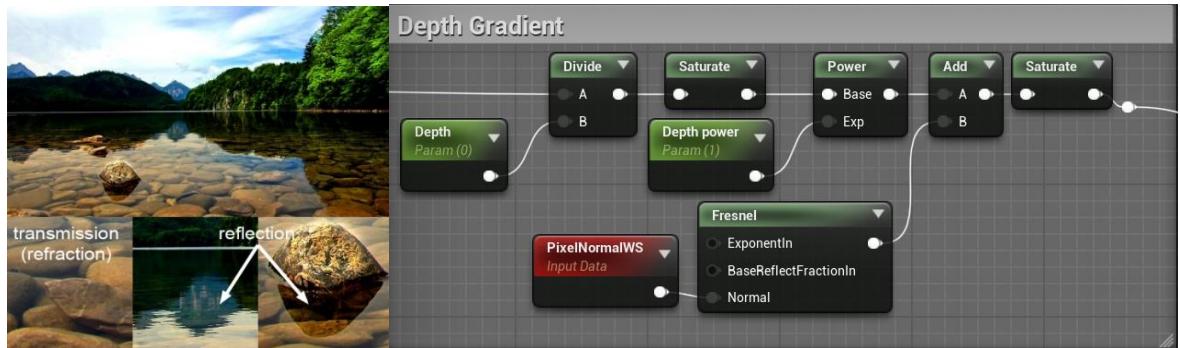


Рисунок 42. Часть шейдера, отвечающая за полное отражение лучей света при низких углах обзора.

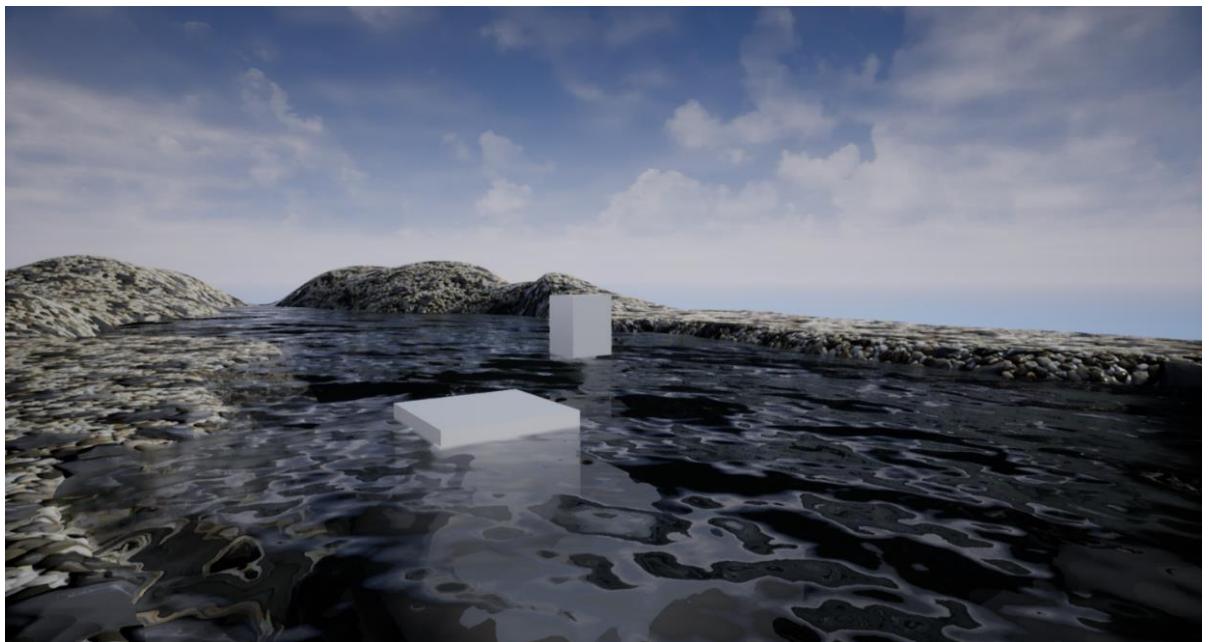


Рисунок 43. Материал воды с поглощением света на глубине и полном отражении при низких углах падения света.

3.1.5 Создание пены на поверхности воды

В потоке пена появляется на контакте на границах воды, поэтому для определения положения пены можно использовать функцию *DistanceToNearestSurface*, возвращающую расстояние до ближайшей поверхности. Более того, для того чтобы иметь некоторый художественный контроль над размещением пены, она также зависит от цвета вершин модели (Vertex Coloring), что позволяет вручную «раскрашивать» водоём пеной.

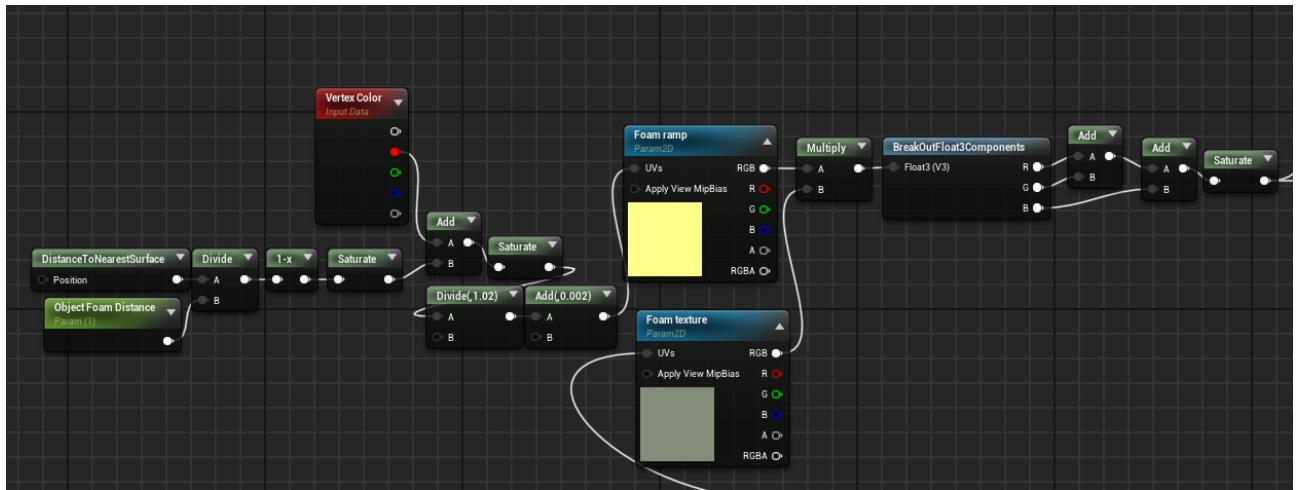


Рисунок 44. Часть шейдера, отвечающая за размещение пены.

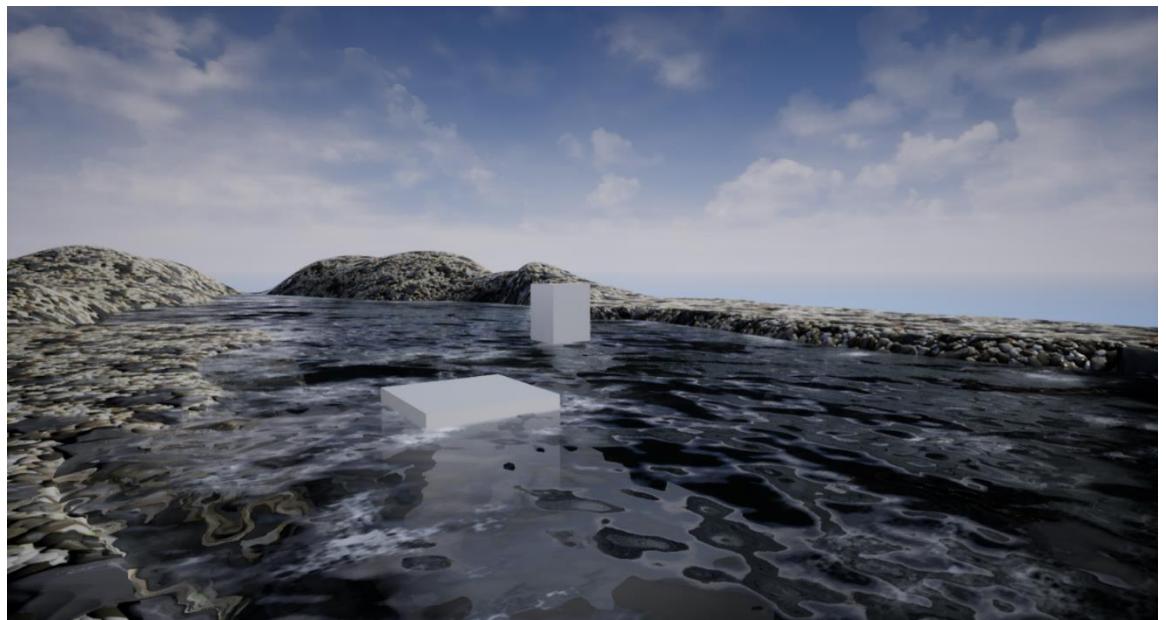


Рисунок 45. Материал воды с пеной на границах сушей.

3.2 Разработка инструмента для создания рек

С помощью системы **Blueprints** был разработан инструмент, позволяющий создавать реки по заданному пользователем объемному сплайну. На рисунке 17 представлена демонстрация работы инструмента. Белым цветом выделен модифицируемый сплайн, по которому создается набор Сплайн-Мешей (Spline-Mesh), 3д моделей, чья геометрия «изогнута» по форме кривой.

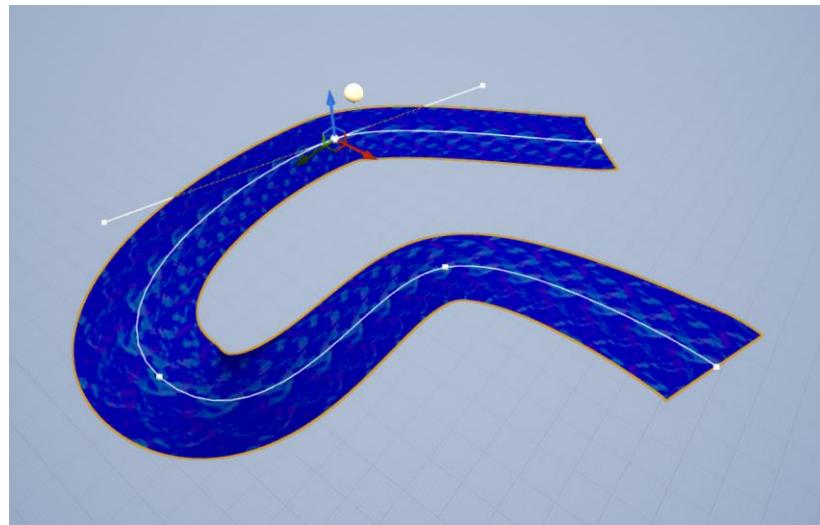


Рисунок 46. Демонстрация инструмента для создания рек.

Для того чтобы направление течения всегда было параллельно руслу реки UV координаты были организованы как на рисунке 18. Таким образом была достигнута видимая неразрывность прокручивающихся текстур волны.

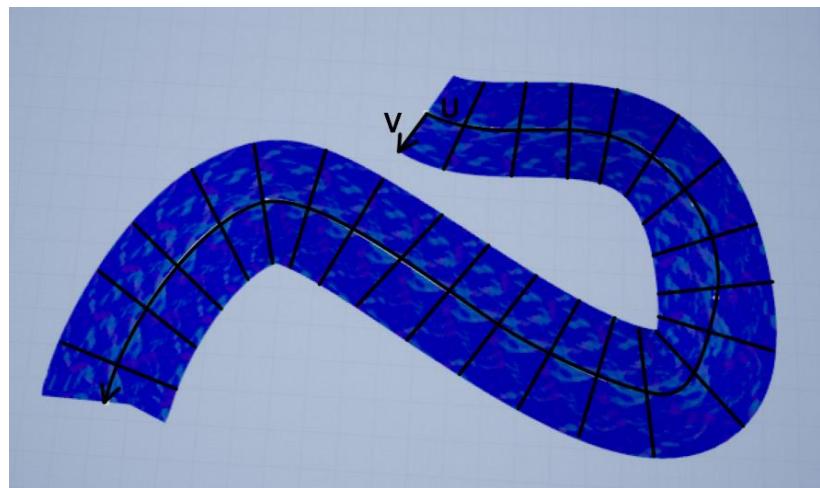


Рисунок 47. Схематичные UV координаты сплайн-меша реки.

3.3 Внедрение водоёмов в готовую природную сцену

В природную сцену, созданную во время эксплуатационной практики, добавлена река с помощью разработанных материалов воды и инструмента для создания рек.



Рисунок 48. Река в контексте природного ландшафта.

С использованием того-же инструмента для создания рек и пены на воде был реализован водопад. С помощью системы частиц *Niagara* были создан эффект брызг водопада.

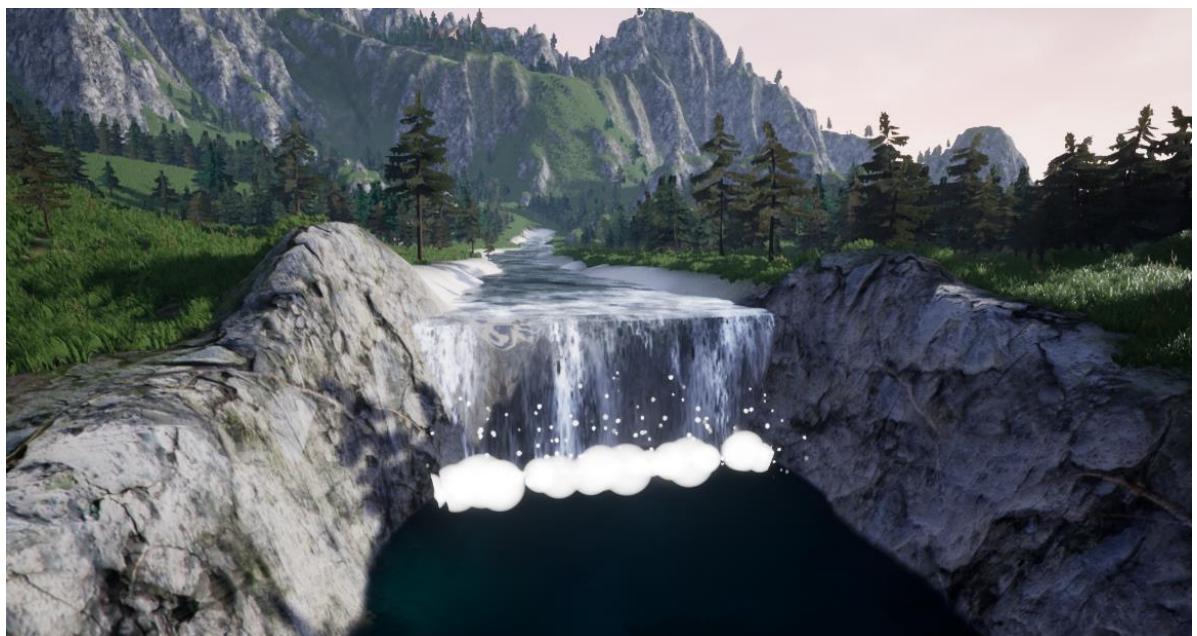


Рисунок 49. Водопад.

4. Разработка игрового персонажа

4.1. Решения, предоставляемые движком Unreal Engine 4

Движок Unreal Engine 4 предоставляет базовый функционал для создания управляемых игровых персонажей. В частности, присутствует набор классов, которые могут быть расширены с помощью языка C++:

AActor – «актёр», базовый класс для любого объекта, который может быть размещен в виртуальном 3д пространстве[1].

APawn – «пешка», базовый класс для любого «актёра», который может находиться под управлением игрока или компьютера.

ACharacter – «пешка», которая обладает «телом», коллайдером (компонентом, отвечающим за столкновение с другими объектами) и базовой логикой перемещения, состоящей из ходьбы и прыжков.

ACharacterMovementComponent – класс, отвечающий за логику перемещения персонажа и различные режимы перемещения.

При анимации персонажей речь всегда идёт о скелетной анимации (Skeletal Animations). Для модели персонажа создаётся скелет, и анимация состоит из вращения костей друг относительно друга. В базовых ассетах Unreal Engine 4 присутствует минималистичная модель человека и небольшой набор анимаций, связанных с ней. Все созданные далее анимации будут применяться к данной модели.



Рисунок 50. Базовая модель человека, поставляемая с Unreal Engine 4.

Технология *Animation Blueprints* позволяет управлять логикой анимаций конкретной модели с помощью визуального программирования. Возможно выполнять плавные переходы между анимациями, напрямую управлять костями скелета или настраивать логику, которая в конечном итоге будет определять окончательную позу модели для каждого кадра.

Технология *State Machines*, встроенная в *Animation Blueprints*, предоставляет способ разбить анимацию персонажа на набор различных состояний. Эти состояния затем регулируются *Transition Rules* (Правилами перехода), которые контролируют, как переходить из одного состояния в другое. *State Machines* значительно упрощает анимацию персонажей, которые могут выполнять большое количество несовместных действий, таких как ходьба, бег и прыжки.

Blendspaces – функции, предоставляемые движком Unreal Engine 4, позволяющие плавно интерполировать позу персонажа между рядами заданных анимаций.

4.2. Создание системы перемещения персонажа

Был разработан класс *ACustomCharacter*, наследующий от класса *ACharacter*. Данный класс служит основой для всех последующих разработанных классов и содержит в себе логику обработки вводов игрока и поля для хранения разнообразных переменных состояния[9].

Был разработан класс *ACustomCharacterMovementComponent*, наследующий от класса *ACharacterMovementComponent*. Данный класс отвечает за перемещение объекта класса *ACustomCharacter*. В частности функция *ACharacterMovementComponent::OnMovementUpdated()* были перегружена для возможности управление максимальной скоростью персонажа в зависимости от режима перемещения и была разработана функция *ACustomCharacterMovementComponent::PhysSlide()* для расчёта «физического» перемещения персонажа при скольжении по наклонной поверхности[10].

**Листинг 3. Функция *ACustomCharacterMovementComponent::PhysSlide()*,
отвечающая за логику перемещения персонажа в состоянии скольжения.**

```
void UCustomCharacterMovementComponent::PhysSlide(float deltaTime, int32 Iterations)
{
    if (deltaTime < MIN_TICK_TIME) { return; }

    FHitResult SurfaceHit;
    if
    (!GetSlideSurface(SurfaceHit) || Velocity.SizeSquared()<pow(Slide_MinSpeed, 2))
    {
        ExitSlide();
        StartNewPhysics(deltaTime, Iterations);
        return;
    }

    Velocity += Slide_GravityForce * FVector::DownVector * deltaTime;

    if (FMath::Abs(FVector::DotProduct(Acceleration.GetSafeNormal(),
UpdatedComponent->GetRightVector())) > 0.5)
    {
        Acceleration = Acceleration.ProjectOnTo(UpdatedComponent-
>GetRightVector());
    }
    else
    {
        Acceleration = FVector::ZeroVector;
    }

    if (!HasAnimRootMotion() && !CurrentRootMotion.HasOverrideVelocity())
    {
        CalcVelocity(deltaTime, Slide_Friction, true,
GetMaxBrakingDeceleration());
    }
    ApplyRootMotionToVelocity(deltaTime);

    Iterations++;
    bJustTeleported = false;

    FVector OldLocation = UpdatedComponent->GetComponentLocation();
    FQuat OldRotation = UpdatedComponent-
>GetComponentRotation().Quaternion();
    FHitResult Hit(1.f);
    FVector Adjusted = Velocity * deltaTime;
    FVector VelPlaneDir = FVector::VectorPlaneProject(Velocity,
SurfaceHit.Normal).GetSafeNormal();
    FQuat NewRotation = FRotationMatrix::MakeFromXZ(VelPlaneDir,
SurfaceHit.Normal).ToQuat();

    SafeMoveUpdatedComponent(Adjusted, NewRotation, true, Hit);

    if (Hit.Time < 1.f)
    {
        HandleImpact(Hit, deltaTime, Adjusted);
        SlideAlongSurface(Adjusted, (1.f - Hit.Time), Hit.Normal, Hit,
true);
    }

    FHitResult NewSurfaceHit;
    if (!GetSlideSurface(NewSurfaceHit) || Velocity.SizeSquared() <
pow(Slide_MinSpeed, 2))
```

```

    {
        ExitSlide();
    }

    if (!bJustTeleported && !HasAnimRootMotion() &&
!CurrentRootMotion.HasOverrideVelocity())
    {
        Velocity = (UpdatedComponent->GetComponentLocation() - OldLocation) / deltaTime;
    }
}

```

Листинг 4. Функция *ACustomCharacterMovementComponent::*

***OnMovementUpdated()*, отвечающая за изменение максимальной скорости персонажа в зависимости от режима перемещения.**

```

void UCustomCharacterMovementComponent::OnMovementUpdated(float DeltaSeconds, const FVector& OldLocation,
                                                               const FVector& OldVelocity)
{
    Super::OnMovementUpdated(DeltaSeconds, OldLocation, OldVelocity);

    if (MovementMode == MOVE_Walking && !IsCrouching())
    {
        if(CustomCharacterOwner->IsDoingAnAttack())
        {
            MaxWalkSpeed=Walk_MaxAttackWalkSpeed;
        }
        else if (Safe_bWantsToSprint) // &&
!GetCurrentAcceleration().IsZero()
        {
            MaxWalkRunSpeed = FMath::FInterpTo(MaxWalkRunSpeed,
Sprint_MaxWalkSpeed, DeltaSeconds, Walk_Sprint_InterpSpeedUp);
            MaxWalkSpeed = MaxWalkRunSpeed;
        }
        else
        {
            MaxWalkRunSpeed= FMath::FInterpTo(MaxWalkRunSpeed,
Walk_MaxWalkSpeed, DeltaSeconds, Walk_Sprint_InterpSpeedDown);
            MaxWalkSpeed = MaxWalkRunSpeed;
        }
    }
    if (IsCrouching() && !IsSliding())
    {
        MaxWalkRunSpeed = FMath::FInterpTo(MaxWalkRunSpeed,
Walk_MaxWalkSpeed, DeltaSeconds, Walk_Sprint_InterpSpeedDown);
    }else
    {

    }
    bPrevWantsToCrouch = bWantsToCrouch;
}

```

4.3. Настройка анимаций персонажа

4.3.1 Настройка анимаций перемещения

С помощью технологии *Blendspaces* был создан континуум различных анимаций ходьбы, позволяющий персонажу плавно переходить из стационарного положения в ходьбу, затем в легкую трусцу и наконец в полноценный бег.

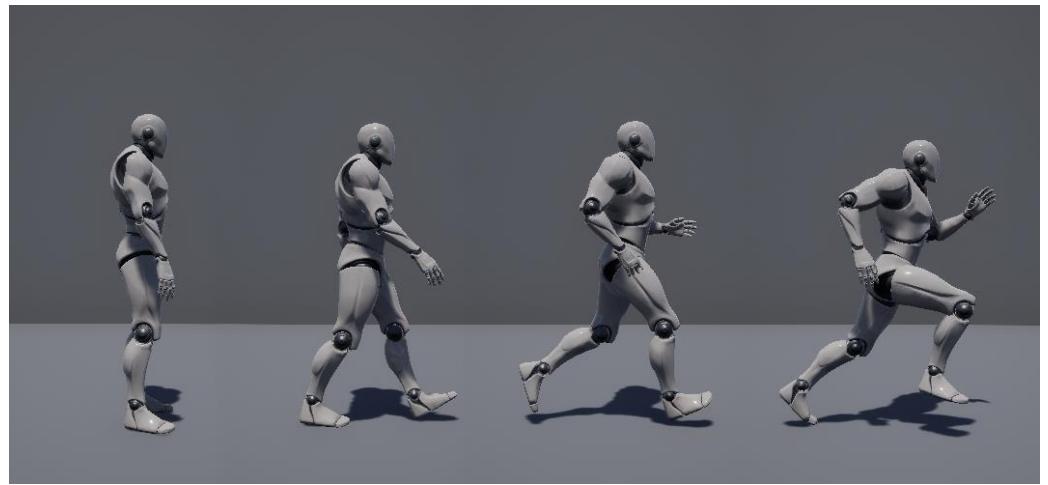


Рисунок 51. Выбор анимации в зависимости от скорости персонажа.

С помощью *State Machines* был создан набор состояний, позволяющих управлять анимацией скольжения персонажа по наклонной поверхности. Это процесс поделен на пять этапов: бег, входжение в скольжение, скольжение, выход из скольжения и снова бег. Персонаж переходит между этими состояниями в зависимости от вводов игрока и времени, проведенного в скольжении[11].

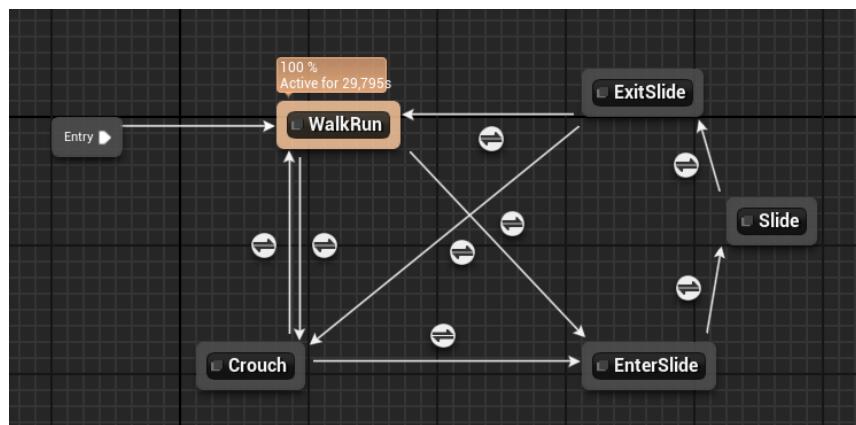


Рисунок 52. Набор состояний, определяющих анимацию персонажа во время скольжения по наклонной поверхности.

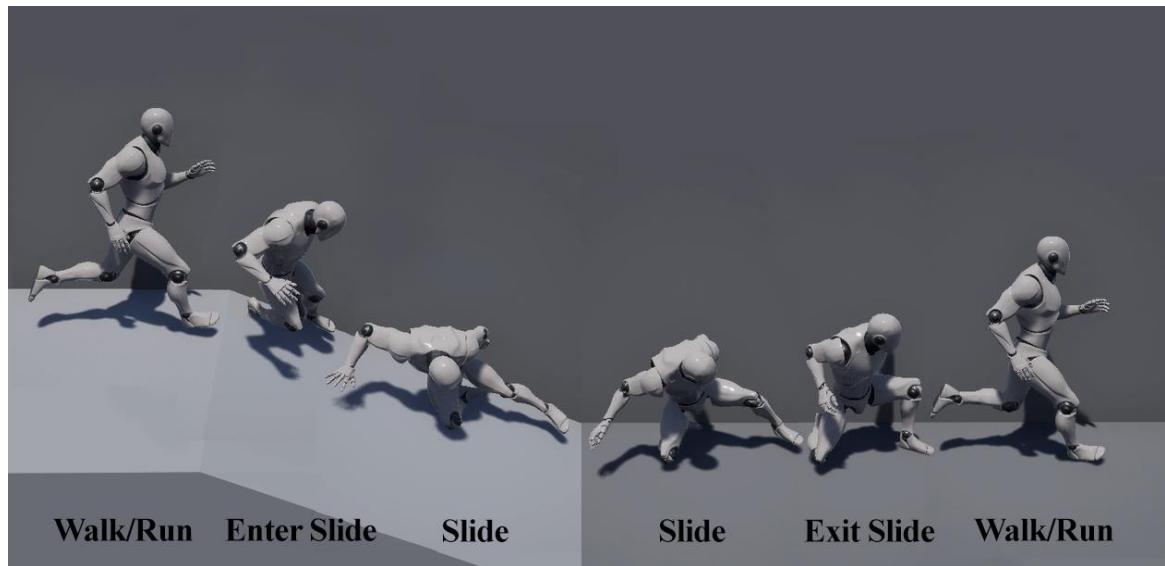


Рисунок 53. Демонстрация работы машины состояний.

6.3.2 Создание анимаций атак

Для возможности выбирать игроком произвольный угол атаки было принято решение сделать набор анимаций атак, между которыми будет проводится интерполяция[20].

С помощью программы Blender 3D были разработаны анимации горизонтального удара слева направо, вертикального удара сверху вниз и горизонтального удара справа налево. Такой набор позволяет получить анимацию любого удара, который проводится сверху вниз с помощью интерполяции.

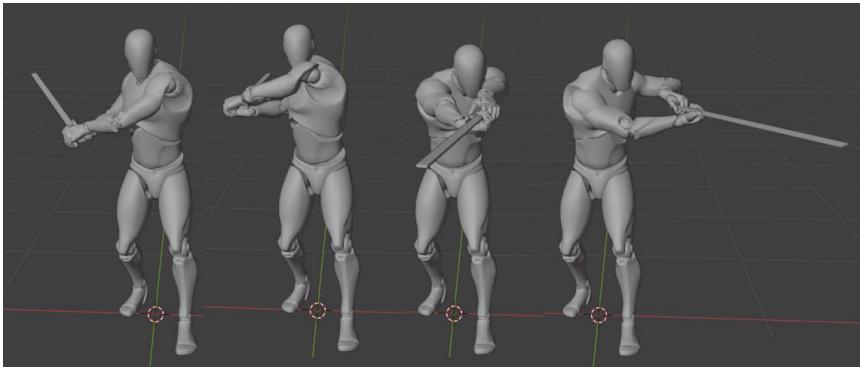
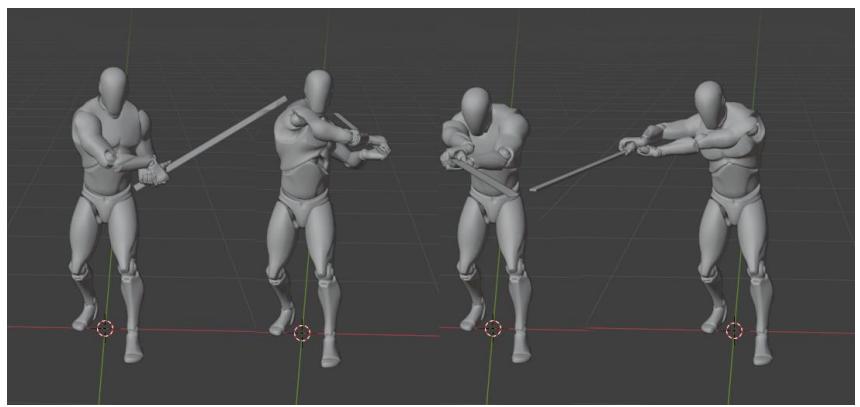


Рисунок 54. Наборы анимаций ударов, созданных в программе Blender 3D.

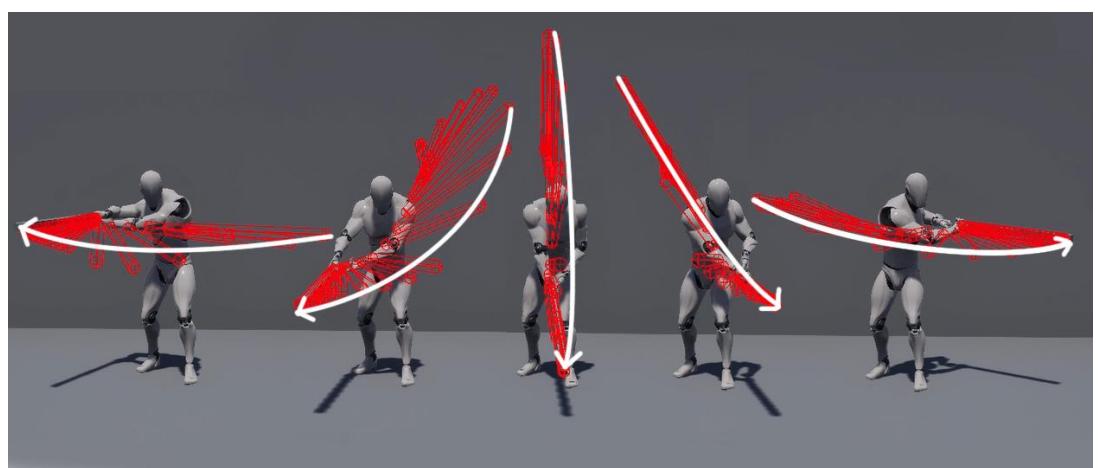


Рисунок 55. Атаки в произвольных направлениях, полученные с помощью интерполяции между наборами анимаций.

4.4. Разработка боевой системы

За основу боевой системы были взяты боевые системы таких видеоигр как For Honor, Dark Souls и Witcher, позволяющие персонажу игрока выбирать одиночную цель в виде вражеского персонажа и наносить ему различные атаки.

Для удобства управления состоянием игрового персонажа был разработан список состояний. В каждый момент времени персонаж может находиться только в одном из этих состояний. Перечень состояний выглядит следующим образом:

- Idle (покой) – персонаж не выполняет никаких действий.
- Winding Up (замах) – персонаж замахивается оружием.
- Attacking (атака) – персонаж наносит атаку и проводит трассировку удара.
- Recovering (восстановление) – персонаж восстанавливается после атаки, возвращая оружие в исходное положение.
- Blocking (блок) – персонаж защищается от входящей атаки.
- Stunned by Attack (Оглушение атакой) – персонаж оглушен входящей атакой и временно не может совершать действий.
- Stunned by Block (Оглушение блоком) – персонаж оглушен после того как его атака была успешно заблокирована и временно не может совершать действий.

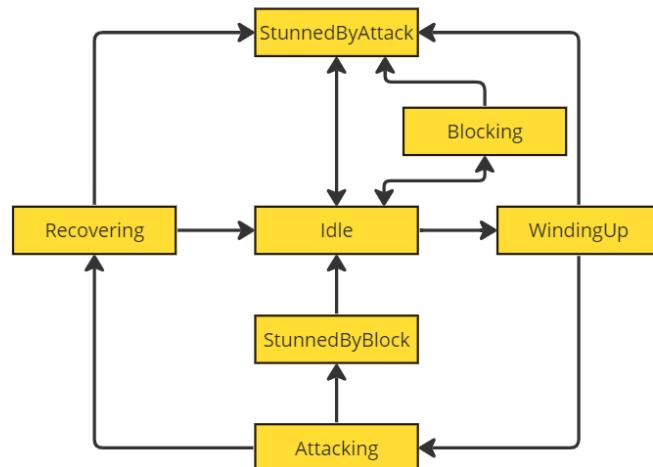


Рисунок 56. Граф состояний персонажа и возможных переходов между ними

Также в список состояний были добавлены переходные состояния, нужные для обеспечения синхронизации между игроками при действиях, которые внезапно меняют состояние персонажа.

- Interrupt Winding Up (прерывание замаха) – персонаж заканчивает замах раньше для нанесения более быстрой но более слабой атаки.
- Block Impact (Успешный блок) – персонаж успешно блокирует входящую атаку.

Листинг 5. Список возможных состояний персонажа.

```
UENUM(BlueprintType)
enum class ECharacterState : uint8
{
    Idle UMETA(DisplayName="Idle"),
    WindingUp UMETA(DisplayName="Winding Up"),
    InterruptWindingUp UMETA(DisplayName="Interrupt Winding Up"),
    Attacking UMETA(DisplayName="Attacking"),
    Recovering UMETA(DisplayName="Recovering"),
    Blocking UMETA(DisplayName="Blocking"),
    BlockImpact UMETA(DisplayName="BlockImpact"),
    StunnedByAttack UMETA(DisplayName="StunnedByAttack"),
    StunnedByBlock UMETA(DisplayName="StunnedByBlock")
};
```

Для обработки действий, которые нужно выполнить при переключении состояний персонажа, таких как начало анимаций (атаки, оглушения), сброс флагов и сброс таймеров была разработана функция ACustomCharacter::HandleCharacterStateChange.

Листинг 6. Функция ACustomCharacter::HandleCharacterStateChange, обрабатывающая переключение персонажа между разными состояниями.

```
void ACustomCharacter::HandleCharacterStateChange()
{
    switch (CharacterState)
    {
        case ECharacterState::Idle:
            bLockSwordPosition = false;
            break;
        case ECharacterState::WindingUp:
```

```

    bWantsToAttack = true;
    CurrentWindupTime = 0;

    for (int i = 0; i < AttackAnimations.Num(); ++i)
    {
        PlayAnimMontage(AttackAnimations[i], 1, FName("Default"));
    }
    break;

case ECharacterState::InterruptWindingUp:
    for (int i = 0; i < AttackAnimations.Num(); ++i)
    {
        PlayAnimMontage(AttackAnimations[i], 1, FName("Attack"));
    }

    break;

case ECharacterState::Attacking:
    bLockSwordPosition = true;
    bTraceHasAHit = false;
    break;

case ECharacterState::Recovering:
    bLockSwordPosition = true;
    break;

case ECharacterState::BlockImpact:

    for (int i = 0; i < BlockImpactAnimations.Num(); ++i)
    {
        PlayAnimMontage(BlockImpactAnimations[i]);
    }

    CharacterState=ECharacterState::Blocking;
    break;

case ECharacterState::Blocking:
    //Start Block Animation
    bLockSwordPosition = true;
    CurrentBlockTime = BlockTime;
    break;

case ECharacterState::StunnedByAttack:
    for (int i = 0; i < StunAnimations.Num(); ++i)
    {
        PlayAnimMontage(StunAnimations[i]);
    }

    CurrentStunTime = AttackStunTime;
    break;

```

```

        case ECharacterState::StunnedByBlock:
            //Start Stun Animation
            for (int i = 0; i < StunAfterAttackAnimations.Num(); ++i)
            {
                PlayAnimMontage(StunAfterAttackAnimations[i]);
            }
            StunDirection=FVector2D(1,0);
            CurrentStunTime = BlockStunTime;
            break;
        default:
            break;
    }
}

```

Для наглядного выбора цели была реализовано перемещение камеры «за плечо» персонажа и по направлению к цели. Для этого был разработан класс ACustomPlayerManager, наследующий от класса APlayerManager, отвечающего за управление камерой. В разработанном классе была перегружена функция APlayerManager::UpdateViewTarget()

Листинг 6. Функция ACustomPlayerManager::UpdateViewTarget(), отвечающая за положение камеры за плечом персонажа.

```

void ACustomPlayerCameraManager::UpdateViewTarget(FTViewTarget& OutVT,
float DeltaTime)
{
    Super::UpdateViewTarget(OutVT, DeltaTime);
    if(ACustomCharacter* CustomCharacter =
Cast<ACustomCharacter>(GetOwningPlayerController()->GetPawn()))
    {
        UCustomCharacterMovementComponent* CMC = CustomCharacter-
>GetCustomCharacterMovementComponent();

        FVector TargetCrouchOffset = FVector(0,
            0,
            CMC->CrouchedHalfHeight - CustomCharacter->GetClass()-
>GetDefaultObject<ACharacter>()->GetCapsuleComponent()-
>GetScaledCapsuleHalfHeight()
        );
        FVector Offset =
FMath::InterpEaseInOut(FVector::ZeroVector,TargetCrouchOffset,FMath::Clamp(Cro-
uchBlendTime/CrouchBlendDuration,0.f,1.f),TransitionEaseInOutExponent);
        if(CustomCharacter->IsLockedOn())
        {
            AActor* Target=CustomCharacter->GetTargetedActor();
            checkf(Target!=nullptr,TEXT("CustomPlayerCameraManager:
bIsLockedOn is true, but targeted actor is missing"));
            FVector toTarget=Target->GetActorLocation()-CustomCharacter-
>GetActorLocation();
            FVector absOffset=lockOnOffset;
            if(CustomCharacter->IsShoulderCameraFlipped())absOffset.Y=-
absOffset.Y;
        }
    }
}

```

```

        FVector
relativeShoulderCamOffset=toTarget.Rotation().RotateVector(absOffset);
        FVector lookAtPosition=CustomCharacter-
>GetActorLocation()+relativeShoulderCamOffset;
        FRotator
lookAtRotation=UKismetMathLibrary::FindLookAtRotation(GetCameraLocation(),((Ta-
rget->GetActorLocation())+CustomCharacter->GetActorLocation())/2));

        OutVT.POView.Location=FMath::VInterpTo(GetCameraLocation(),lookAtPosition,D-
eltaTime,LockOnFollowSpeed);

        OutVT.POView.Rotation=FMath::RInterpTo(GetCameraRotation(),lookAtRotation,D-
eltaTime,LockOnFollowSpeed);
        GetOwningPlayerController()->SetControlRotation(OutVT.POView.Rotation);
    }
    if(CMC->IsCrouching())
    {
        CrouchBlendTime =
FMath::Clamp(CrouchBlendTime+DeltaTime,0.f,CrouchBlendDuration);
        Offset -= TargetCrouchOffset; //?? TEST THIS
    }
    else
    {
        CrouchBlendTime = FMath::Clamp(CrouchBlendTime-
DeltaTime,0.f,CrouchBlendDuration);
    }

    if(CMC->IsMovingOnGround())
    {
        OutVT.POView.Location+=Offset;
    }
}
}

```



Рисунок 57. Расположение камеры за плечом персонажа при выборе цели.

С помощью технологии Notify States, позволяющей выполнять логику в определенный промежуток анимации реализуется удар. Каждый кадр атаки происходит трассировка цилиндром по всей длине меча. При попадании по другому персонажу срабатывает логика, отвечающая за получение урона.

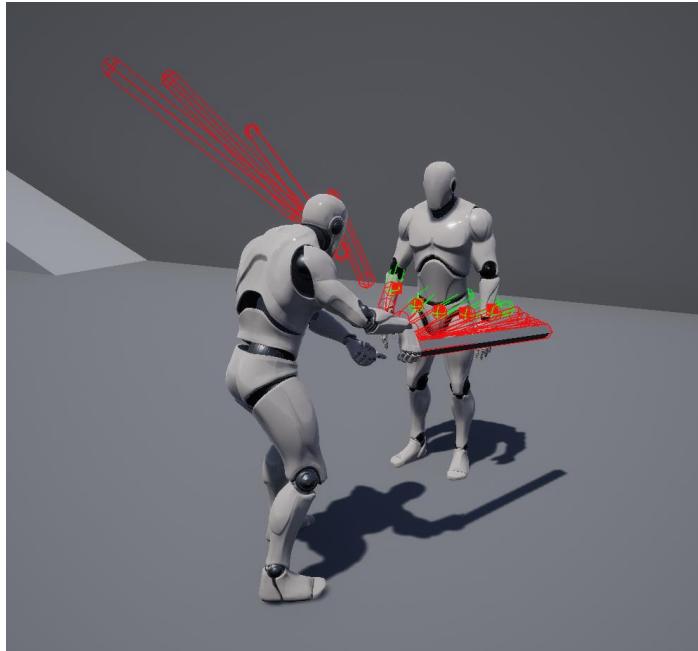


Рисунок 58. Трассировка атаки. Жёлтые фигуры показывают успешное попадание по выбранному персонажу.

4.5. Разработка искусственного интеллекта

Unreal Engine 4 предоставляет широкий набор инструментов для создания искусственного интеллекта (ИИ) неигровых персонажей (NPC):

- **Behavior Trees** (Деревья поведения): Структуры данных, позволяющие описывать и организовывать поведение NPC в иерархическом виде
- **Blackboards** (Чёрные доски): Работают совместно с Behavior Trees и служат для хранения и обмена данными между различными частями ИИ системы.
- **Perception System** (Система восприятия): Система восприятия позволяет NPC "чувствовать" окружение, реагировать на звуки, видеть других персонажей и объекты.

Для демонстрации боевой системы в однопользовательском режиме игры был разработан базовый алгоритм поведения неигрового персонажа, позволяющий ему следовать за игроком, защищаться от атак и наносить удары.

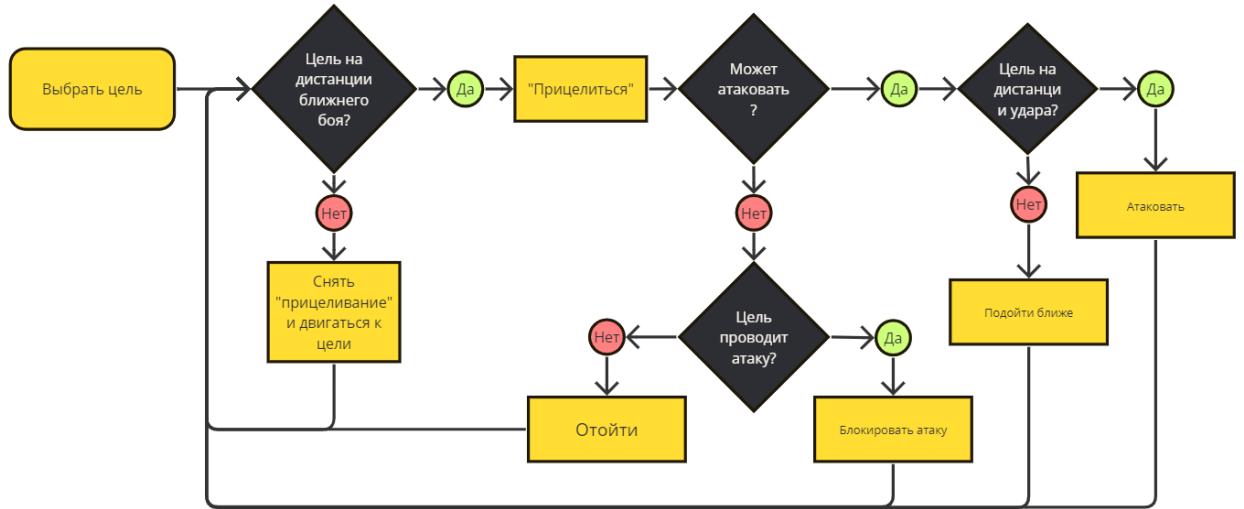


Рисунок 59. Логика поведения неигрового персонажа.

С помощью технологии Behavior Trees и визуального программирования Blueprints разработанный алгоритм поведения был перенесен в Unreal Engine 4.

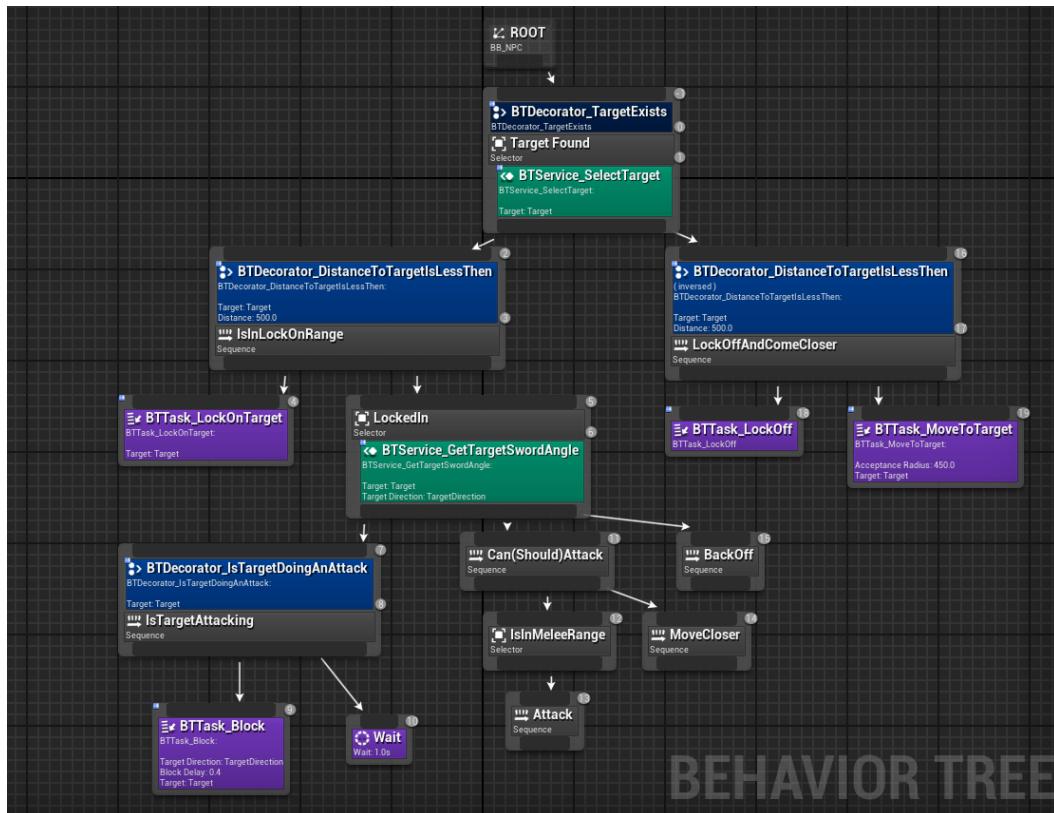


Рисунок 60. Логика поведения ИИ, представленная в виде дерева Behavior Tree.

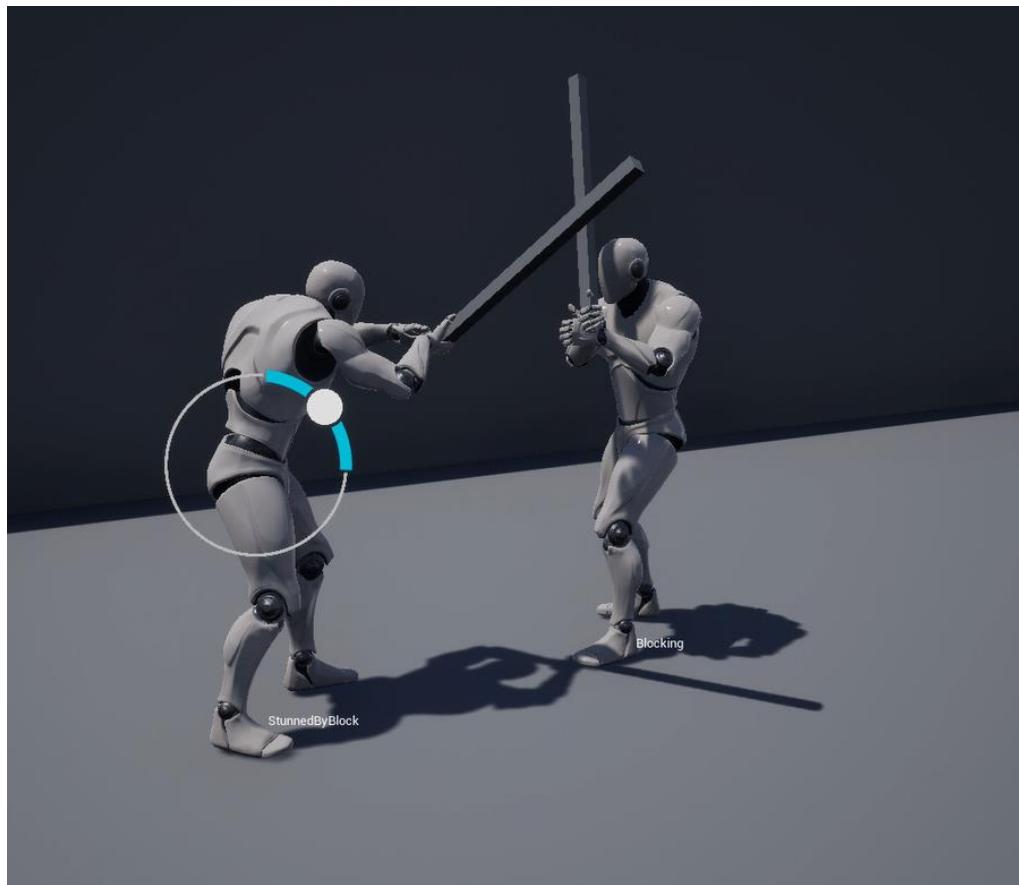


Рисунок 61. Демонстрация работы ИИ. Игрок (слева) наносит удар. Неигровой персонаж (справа) успешно блокирует удар.

4.6. Настройка многопользовательской игры

Двигок Unreal Engine 4 предоставляет широкий набор инструментов технологий для создания многопользовательских приложений. Технологии, которые были использованы в ходе данной работы:

- **Replication** (Репликация) – технология, позволяющая синхронизировать состояния объектов между сервером и клиентами. UE4 позволяет автоматически реплицировать переменные, функции и события[17].
- **Remote Procedure Calls** (удаленные вызовы процедур) или **RPC** – технология, позволяющий программе вызвать процедуру (функцию) на удаленном сервере так, как если бы она выполнялась локально.

В рамках данной практики была выбрана модель, в которой сервер обрабатывает всю важную для игрового процесса логику для избежание рассинхронизации игровых процессов разных клиентов. Так, сервер вычисляет,

нанесет ли один игрок другому атаку или она будет заблокирована и имеет последнее слово за всеми переходами персонажей между различными состояниями.

С помощью технологии RPC была разработана функция *ACustomCharacter::SetCharacterState*, позволяющая обрабатывать переключение состояний персонажа на сервере либо выполнять удаленный вызов процедуры если функция была вызвана на клиенте[16][19].

Листинг 7. Функции *ACustomCharacter::SetCharacterState*, позволяющая клиенту менять состояние персонажа в синхронизации с сервером.

```
void ACustomCharacter::SetCharacterState(ECharacterState NewState)
{
    CharacterState = NewState;
    HandleCharacterStateChange();
    if (!HasAuthority())
    {
        Server_SetCharacterState(NewState);
    }
}
```

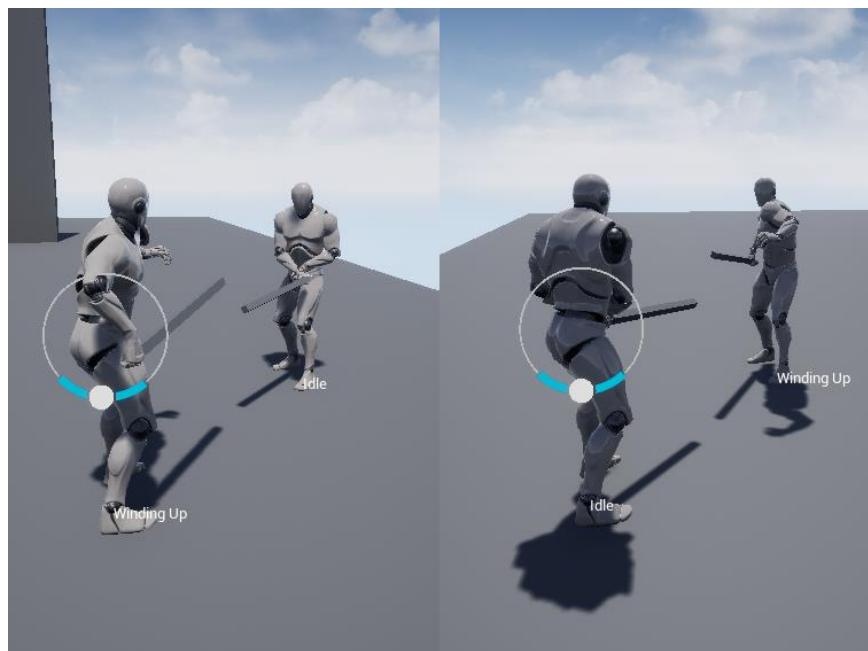


Рисунок 62. Демонстрация работы игры в многопользовательском режиме. Представлены изображения с двух клиентов, подключенных к одному серверу.

4.7. Внедрение игровых персонажей в природную сцену

Из Epic Games Store были взяты 3д модели персонажей - пиратов и различного холодного оружия. С помощью технологии Retargeting [15] – перенесения анимаций между похожими скелетами (обладающими примерно одинаковыми наборами костей), к взятым 3д моделям были применены разработанные анимации. Наделенные моделями персонажи были перенесены в разработанный природный ландшафт.



Рисунок 63. Персонаж с наложенной на него 3д моделью.



Рисунок 64. Персонаж игрока(слева) и неигровой персонаж(справа) в готовой природной сцене.

ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы были выполнены следующие задачи:

1. Разработана небольшая природная сцена с высокой детализацией и погодой, изменяемой по вводу пользователя;
2. Разработана и оптимизирована природная сцена с ландшафтом большого размера;
3. Разработаны реалистичные шейдеры воды и инструмент для создания водоёмов;
4. Создан анимированный игровой персонаж;
5. Создан искусственный интеллект неигрового персонажа, разработана боевая система, позволяющая выбирать и атаковать неигровых персонажей.

В результате работы были изучены методы создания природных сцен. Изучены средства разработки на языке C++, и технология Blueprints предоставляемые движком Unreal Engine 4. Приобретены знания об организации проектов. Изучены различные методы оптимизации природных сцен.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Unreal Engine 4 Documentation // Unreal Engine Documentation URL: <https://docs.unrealengine.com/>. Дата обращения: 07.04.2024.
2. Geometry instancing // Wikipedia, the free encyclopedia URL: https://en.wikipedia.org/wiki/Geometry_instancing. Дата обращения: 21.04.2024.
3. Modeling – Blender Manual // Blender Manual URL: <https://docs.blender.org/manual/en/latest/modeling/index.html>. Дата обращения: 18.02.2022.
4. Мірмар // Wikipedia, the free encyclopedia URL: <https://en.wikipedia.org/wiki/Мірмар>. Дата обращения 05.03.2024.
5. Level of Detail (computer graphics) // Wikipedia, the free encyclopedia URL: [https://en.wikipedia.org/wiki/Level_of_detail_\(computer_graphics\)](https://en.wikipedia.org/wiki/Level_of_detail_(computer_graphics)). Дата обращения: 05.04.2024.
6. Creating and Using LODs // Unreal Engine Documentation URL: <https://docs.unrealengine.com/4.27/en-US/WorkingWithContent/Types/StaticMeshes/HowTo/LODs/>. Дата обращения: 05.04.2024.
7. Инстансинг [Электронный ресурс] // Habr: интернет-портал. URL: <https://habr.com/ru/post/352962/>. Дата обращения: 03.04.2024.
8. UE4 Optimization: Instancing // YouTube: видео хостинг. URL: <https://www.youtube.com/watch?v=oMIbV2rQO4k>. Дата обращения: 03.04.2024.
9. Programming Quick Start // Unreal Engine Documentation URL: <https://docs.unrealengine.com/5.0/en-US/unreal-engine-cpp-quick-start/>. Дата обращения: 07.04.2024.
10. Unreal Engine | Character Movement Component: In-Depth // YouTube, Delgoodie URL: www.youtube.com/playlist?list=PLXJlkahwiwPmeABEhjwIALvxRSZkzoQpk. Дата обращения: 08.02.2024
11. Rendering Inside // Playdead Mikkel Gjol & Mikkel Svendsen URL: https://loopit.dk/rendering_inside.pdf. Дата обращения: 02.05.2024

12. Creating Natural Landscapes for Games // Fernando Quinn URL: <https://80.lv/articles/creating-natural-landscapes-for-games/>. Дата обращения: 31.11.23
13. Improving Vegetation Through Vertex Normals // Matt Billeci URL: <https://www.artstation.com/artwork/w6nQ96> Дата обращения: 03.12.23
14. Creating a Dreamy Game-Ready Landscape in UE4 & Substance Designer // Ioana Santamarian URL: <https://80.lv/articles/creating-a-dreamy-game-ready-landscape-in-ue4-substance-designer/> Дата обращения: 15.05.24
15. Mixamo Animation Retargeting // UNAMedia URL: <https://www.unamedia.com/ue5-mixamo/docs/tutorial/> Дата обращения: 17.05.24
16. Mastering Unreal Engine 4.X // Muhammad A.Moniem URL: <https://cread.jd.com/read/startRead.action?bookId=30372728&readType=1> Дата обращения: 20.05.24
17. Unreal Engine Learning Portal // Unreal Engine Online Learning URL: <https://www.unrealengine.com/marketplace/en-US/content-cat/assets/onlinelearning> Дата обращения: 03.04.24
18. Unreal Engine 4 Game Development Essentials // Satheesh Pv URL: http://www.acornpub.co.kr/acorn_guest/UnrealEngine4GameDevelopmentEssentialsColorImages.pdf Дата обращения: 17.03.24
19. Game Engine Architecture // Jason Gregory URL: <https://www.gameenginebook.com/> Дата обращения: 13.03.24
20. How To Make Animations For The Unreal Engine 4 Mannequin // Unreal University URL: <https://www.youtube.com/watch?v=tnZv7KQPai0> Дата обращения: 15.04.24

ПРИЛОЖЕНИЕ А. Графическая часть выпускной квалификационной работы

В графическую часть выпускной квалификационной работы входят 8 чертежей.

- Графики зависимости FPS от плотности сетки ландшафта;
- Итоговая лесная сцена при солнечной и дождливой погоде;
- Итоговая сцена с островом;
- Демонстрации методов рельефного текстурирования;
- Водоёмы, внедренные в сцену с островом;
- Демонстрация переходов персонажа между различными состояниями;
- Анимации атак разных направлений;
- Анимации ударов мечом в различных направлениях, полученные с помощью интерполяции между исходными анимациями;