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 _____ **Фёдоров А.В.**
(Группа) (Подпись, дата) (Фамилия И.О.)

Руководитель ВКР _____ **Витюков Ф.А.**
(Подпись, дата) (Фамилия И.О.)

Нормоконтролёр _____ ------
(Подпись, дата) (Фамилия И.О.)

2024 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ

Заведующий кафедрой РК6
(Индекс)
_____ Карпенко А.П.
(Фамилия И.О.)
«15» февраля 2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра

Студент группы РК6-81Б

Фёдоров Артемий Владиславович
(фамилия, имя, отчество)

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

Источник тематики (НИР кафедры, заказ организаций и т.п.): кафедра.

Тема квалификационной работы утверждена распоряжением по факультету РК №___ от «__» _____ 2024 г.

Техническое задание

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.

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.

Оформление выпускной квалификационной работы

Расчетно-пояснительная записка на 76 листах формата А4.

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

Работа содержит 8 графических листов формата А4.

Дата выдачи задания: «10» февраля 2022 г.

В соответствии с учебным планом выпускную квалификационную работу выполнить в полном объеме в срок до «11» июня 2022 г.

Руководитель квалификационной работы _____ **Витюков Ф.А.**
(Подпись, дата) (Фамилия И.О.)

Студент _____ **Фёдоров А.В.**
(Подпись, дата) (Фамилия И.О.)

Примечание: 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.
(Фамилия И.О.)

«15» февраля 2024 г.

КАЛЕНДАРНЫЙ ПЛАН
выполнения выпускной квалификационной работы

Студент группы РК6-81Б

Фёдоров Артемий Владиславович
(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	_____	Руководитель ВКР	Витюков Ф.А.
2.	1 часть. Аналитическая часть	18.02.2024	_____	Руководитель ВКР	Витюков Ф.А.
3.	Утверждение окончательных формулировок решаемой проблемы, цели работы и перечня задач	28.02.2024	_____	Заведующий кафедрой	Карпенко А.П.
4.	2 часть. Практическая часть 1	21.04.2024	_____	Руководитель ВКР	Витюков Ф.А.
5.	3 часть. Практическая часть 2	23.05.2024	_____	Руководитель ВКР	Витюков Ф.А.
6.	1-я редакция работы	28.05.2024	_____	Руководитель ВКР	Витюков Ф.А.
7.	Подготовка доклада и презентации	04.06.2024	_____	Студент	Фёдоров А.В.
8.	Заключение руководителя	10.06.2024	_____	Руководитель ВКР	Витюков Ф.А.
9.	Допуск работы к защите на ГЭК (нормоконтроль)	17.06.2024	_____	Нормоконтролер	Грошев С.В.
10.	Внешняя рецензия	17.06.2024	_____		
11.	Защита работы на ГЭК	21.06.2024	_____		

Студент _____ Фёдоров А.В.
(подпись, дата) (Фамилия И.О.)

Руководитель ВКР _____ Витюков Ф.А.
(подпись, дата) (Фамилия И.О.)

АННОТАЦИЯ

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

В данной работе рассмотрены и проиллюстрированы следующие этапы: прохождения 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

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. Из всех инструментов Unreal Engine выделяет ориентированность на реалистичность получаемого изображения и возможность работы «в реальном времени», то есть не требуются длительные расчёты как при рендере видео 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 три основные части:

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

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. Потенциальный зритель или игрок не должен натыкаться на куски локации, выглядящие пустыми или переполненными, так как это сильно скажется на уровне 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 технологий оптимизации.

Цели работы:

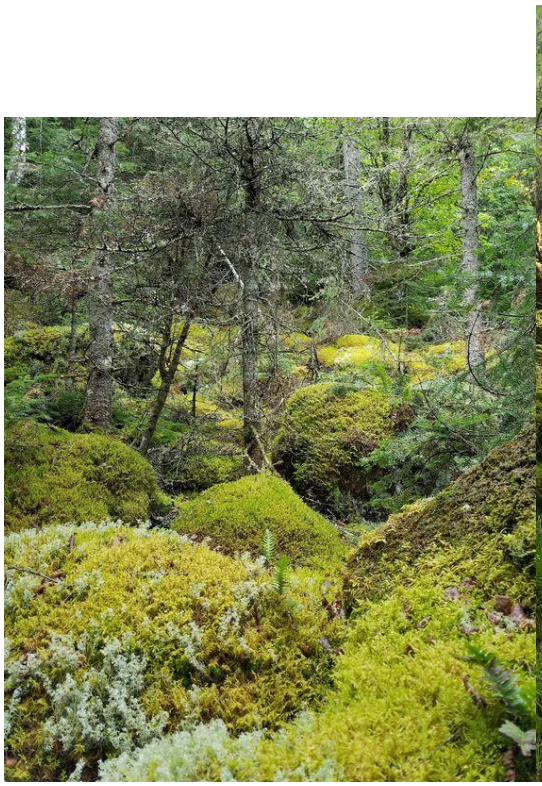
1. создать природную сцену со статичной камерой;
2. разработать систему управлению погодой по нажатию клавиши;
3. создать и оптимизировать природную сцену большого размера;
4. разработать инструменты для создания водоёмов;

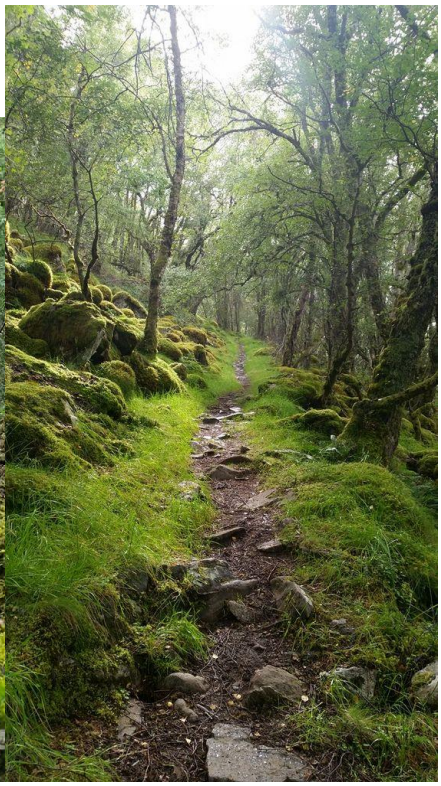
Для выполнения работы были использованы средства следующих программ:

- 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 заключается в изучении референсов, создание самого ландшафта (земли), заполнении его большими, средними и малыми объектами, настройки освещения WIP





В качестве референсов были взяты различные фотографии лесных тропинок и склонов

1.1. Создание ландшафта

Ландшафт сцены был создан с помощью программы blender 3d и импортирован в проект виде Static Mesh.

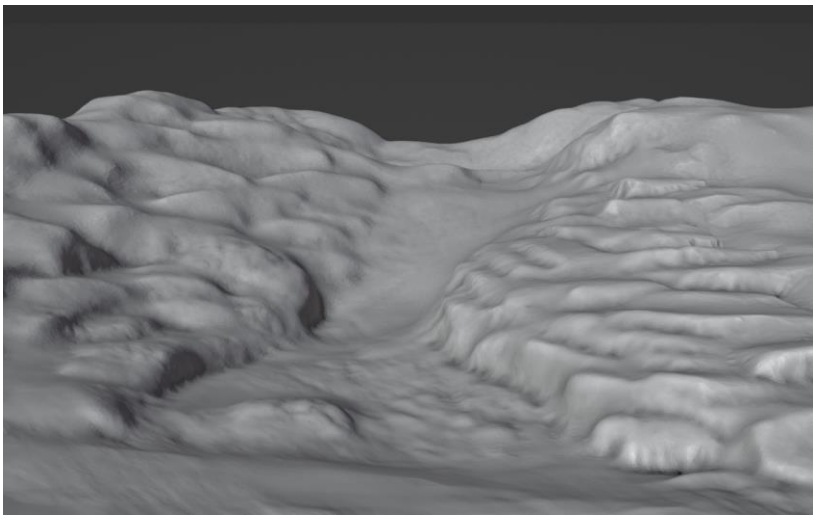


Рисунок . Базовая 3д модель лесной тропы

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 отвечает за материал травы, Высокие значения в зеленом канале отвечает за материал камней, значения, близкие к нулю в зеленом и красном канале отвечают за материал песка. Благодаря этому возможно распределение материалов ландшафта с помощью “раскрашивания” модели внутри редактора Unreal Engine.

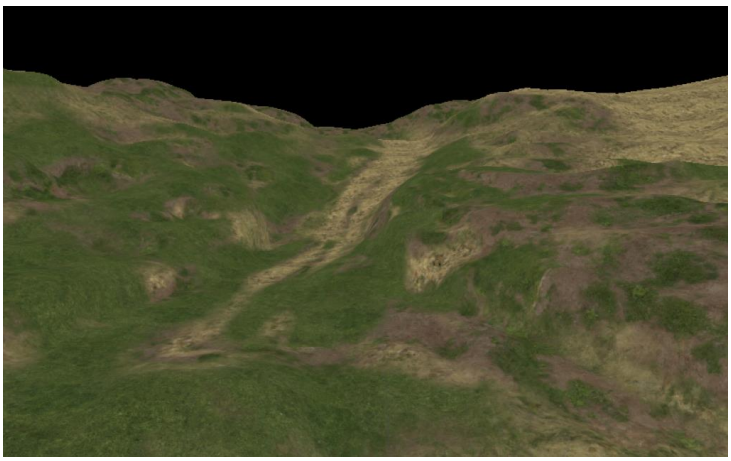


Рисунок . Многослойный материал примененный к модели

1.2. Заполнение сцены объектами

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

Рисунок . 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. С помощью инструмента Foliage Tool были расставлены модели травы, упавших веток, маленьких камней и различных ростков.

Рисунок . 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 5. Сцена при ясной погоде

Эффект дождя был создан при помощи повышения плотности тумана, 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 6. 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 SetClearWeather(), отвечающая за переключение погоды на ясную.

```
	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);
		}

	}
```

1.4. --

2. Разработка внутриигровых систем

ККИ (CCG, Collectible Card Game, коллекционная карточная игра) – разновидность настольных и компьютерных игр. Как правило, ККИ являются стратегическими играми с элементом случайности. Каждая карта представляет собой элемент игры: существо, заклинание, ресурс, и т.д.

Среди компьютерных карточных игр известны следующие: Hearthstone, The Elder Scrolls: Legends, Magic: The Gathering Arena, Gwent: The Witcher Card Game, и др.

Среди настольных: Magic: The Gathering, Yu-Gi-Oh!, Pokémon, и др.

Перед началом поединка игрок собирает из имеющихся карт колоду, соответствующую правилам игры.

Как правило, матчи в подобных играх проходят в режиме «1 на 1». Целью игрока обычно является понижение очков здоровья оппонента до 0. Ходы игроков чередуются. Каждый из них содержит определенное количество фаз, которое задается правилами игры. Например:

1. взятие карты из колоды в «руку»;
2. выкладка карт на поле боя (при условии наличия ресурсов);
3. атака карт противника;
4. завершение хода.

В рамках коллекционных карточных игр существуют такие системы (зоны), как:

- поле боя (Battlefield) – набор активных карт каждого из игроков;
- «рука» (Hand) – набор карт на текущем ходу;
- библиотека (Library) – колода;
- кладбище (Graveyard) – набор использованных карт.

В данной работе рассматривается разработка визуальной части двух систем: поле боя и «рука».

2.1. Разработка системы «поле боя»

Поле боя – одна из основных зон компьютерных ККИ. Представляет собой часть 3d-сцены, на которую происходит выкладка карт игроков. При создании системы поля боя необходимо учесть следующие требования:

1. реализация должна учитывать специфику игры;
2. карты на поле боя должны четко восприниматься игроком;

Специфика игры может подразумевать наличие карт с уникальными свойствами и расположением, наличие дополнительных зон на поле боя, наличие ограничений.

Например, в *Magic: The Gathering Arena* специфика игры требует наличия множества отдельных зон на поле боя для карт различных типов. В *TES: Legends* существует две половины поля, на каждом из которых идет отдельная схватка. С другой стороны, в *Hearthstone* поле боя имеет строгое ограничение по количеству карт и не требует наличия дополнительных подсистем.

В результате анализа возможных подходов к выкладке карт на поле боя было принято решение использовать многорядную систему с группировкой карт одного типа. Она позволяет снять строгие ограничения по количеству выкладываемых карт, за счет чего система становится более гибкой, при этом сохраняя целостность восприятия и экономя место на поле боя. Пример изображен на рисунке 27:

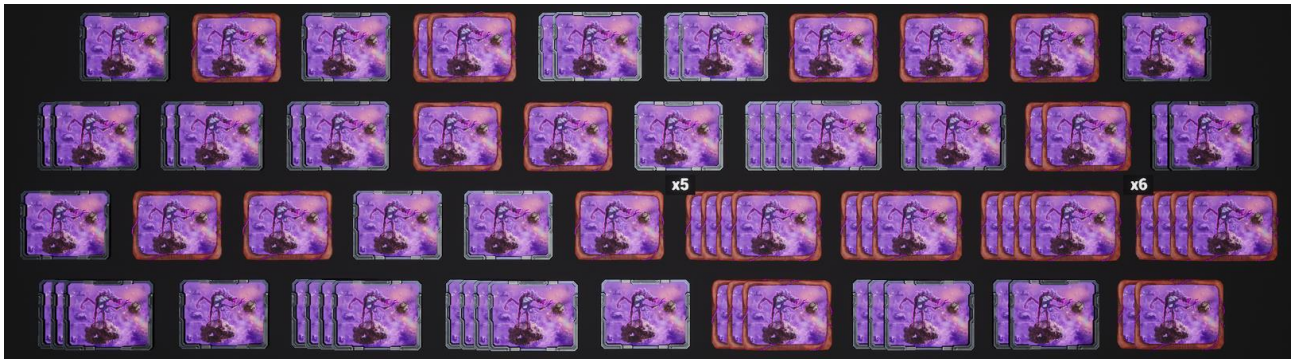


Рисунок 27. Поле боя с четырьмя заполненными рядами

2.1.1. Основные принципы реализации

Каждый игрок имеет собственную область на поле боя, в которую происходит выкладка карт. Выкладка происходит по рядам, поддерживающим группировку карт.

Группировка карт происходит по их типу. Тип, в свою очередь, определяется двумя параметрами: идентификатор колоды, идентификатор карты в колоде. Группы не имеют строгого ограничения по количеству карт в них. Для удобства восприятия групп и работы с ними используются следующие приёмы:

1. изначально задается ограничение по количеству отрисовываемых карт в группе. Если текущее число карт не превышает соответствующее значение, добавление новой карты явно отразится в изменении внешнего вида группы. В противном случае, рядом с ней появится счётчик, показывающий актуальное количество карт (рис. 28);
2. поскольку реальное число карт в группе может отличаться от количества видимых, необходимо добавить возможность выбора из нее элементов. Для этого создан дополненный вариант ряда, имеющий задний фон. В случае раскрытия группы для выбора он появляется над областью с исходной группой и перекрывает ее собой (рис. 29).

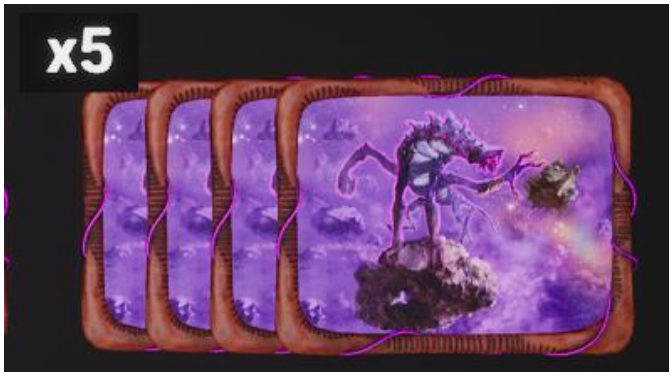


Рисунок 28. Группа карт со счётчиком

Рисунок 29. Выбор карт из группы

Изначально на поле боя создается два ряда. В начале игры выкладывается основная карта – она должна располагаться строго посередине нижнего ряда, который является уникальным. Далее карты выкладываются по принципу, изображенному на рисунке 30 (серым обозначена основная карта). При заполнении всех существующих рядов происходит анимация их масштабирования и перемещения, затем – создание нового ряда и добавление карты в него.

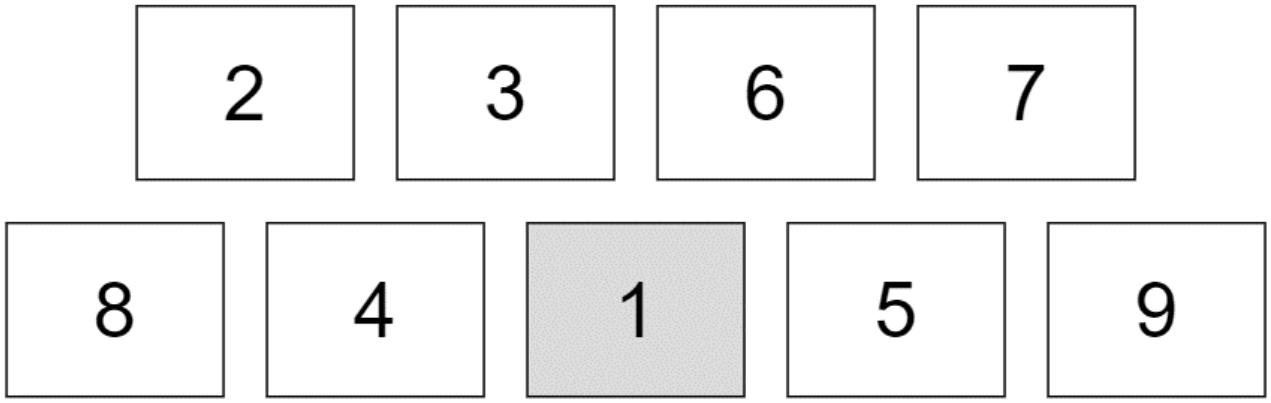


Рисунок 30. Порядок выкладки карт по начальным рядам

Также для работы с большим количеством карт реализована возможность приближения камеры к выбранной половине поля боя.

2.1.2. Программная реализация

Для реализации объекта «поле боя» были разработаны следующие классы:

1. AActorContainer – 3d-контейнер для произвольных actor’ов. Позволяет добавлять и удалять actor’ов, а также перемещать содержимое в течение времени относительно своей системы координат;
2. ACardContainerBase – класс, производный от AActorContainer. Является 3d-контейнером для карт. Позволяет добавлять (создавать), и удалять карты по их данным (FCardData). Также предоставляет возможность перемещения содержимого в течение времени;
3. AInstancedCardContainer – класс, производный от ACardContainerBase. 3d-контейнер для карт, использующий механизм рендер пулов;
4. UActorMovementComponent - класс, позволяющий перемещать произвольные actor’ы по сцене в течение заданного времени;
5. UInstancedCardMovementComponent – класс, производный от UActorMovementComponent. Позволяет перемещать карты и их инстансы по сцене в течение заданного времени. Используется для анимации перемещения карт внутри одного ряда;
6. AGroupingCardRow – класс ряда, поддерживающего группировку карт по типу. Тип карты определяется номером колоды и номером карты в колоде. Отвечает за расчет трансформов групп карт, а также за отображение счетчиков;
7. ACompositeCardRow – класс композитного ряда, состоящего из трех рядов. Один из них находится строго по центру и используется для размещения основной карты, остальные – слева и справа от среднего;
8. AGroupViewCardRow – ряд, позволяющий просматривать содержимое группы. Имеет задний фон, который перекрывает часть поля боя;
9. AGroupingBattlefieldPlayerSection – класс части поля, принадлежащей одному игроку. Представляет собой совокупность рядов, поддерживающих группировку карт. Отвечает за расчет трансформов рядов;

10. UCardRowMovementComponent – класс, производный от UActorMovementComponent. Позволяет перемещать ряды по сцене в течение заданного времени. Используется для анимации перемещения и масштабирования рядов внутри части поля боя;
11. ABattlefield – класс, представляющий само поле боя. Предоставляет контроль над секциями по локальному идентификатору игрока. Позволяет добавлять карты на поле боя, а также удалять их.
12. ABattlefieldOverview – класс, управляющий камерами на поле боя. Содержит по одной камере на область игрока, положение каждой из которых рассчитывается исходя из текущих габаритов области, а также полностью статичную камеру, предоставляющую обзор всего поля боя.

Пример поля боя изображен на рисунке 31:

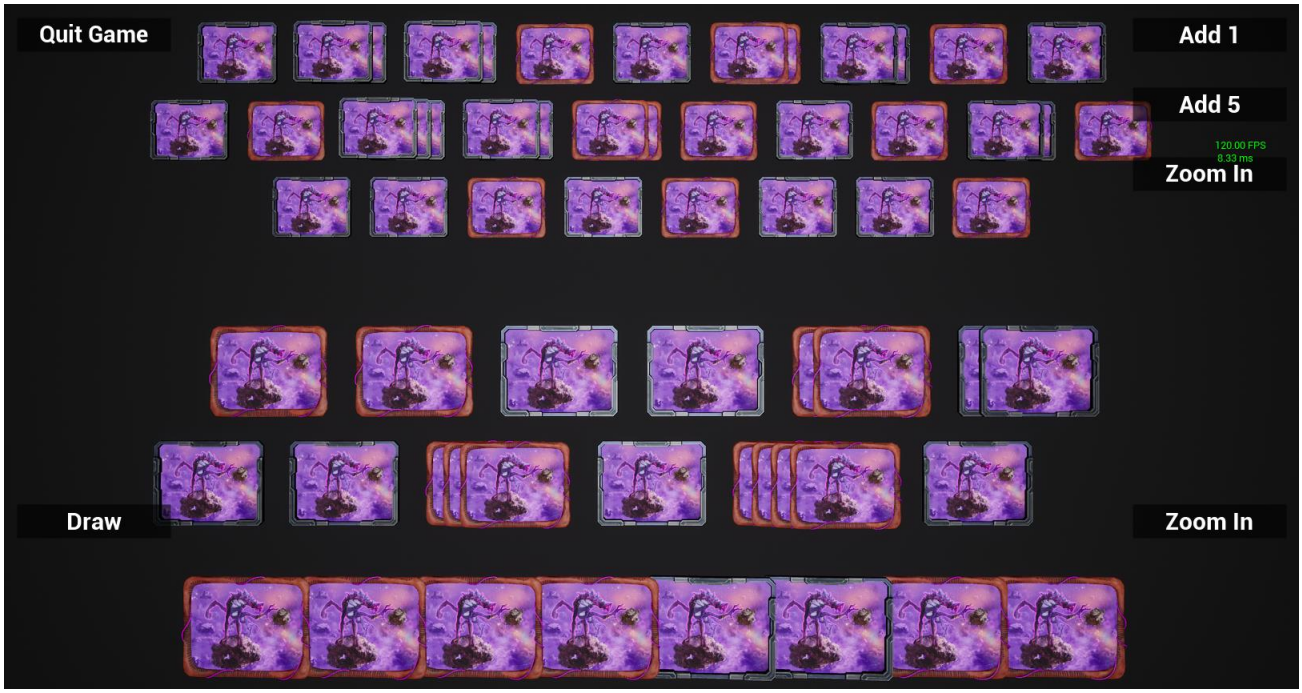


Рисунок 31. Поле боя

Так как количество рядов не имеет строгих ограничений, а при добавлении новых рядов их масштаб будет уменьшаться, необходимо реализовать возможность приближения камеры к областям игроков на поле боя. Для этого реализован класс ABattlefieldOverview, имеющий дополнительные камеры над каждой из областей. Положение камер меняется в зависимости от количества

рядов. Цель – попадание всех карт в область видимости с наибольшим возможным масштабом.

При нажатии на кнопку «Zoom In» происходит приближение камеры к выбранной области игрока. Результат изображен на рисунке 32:

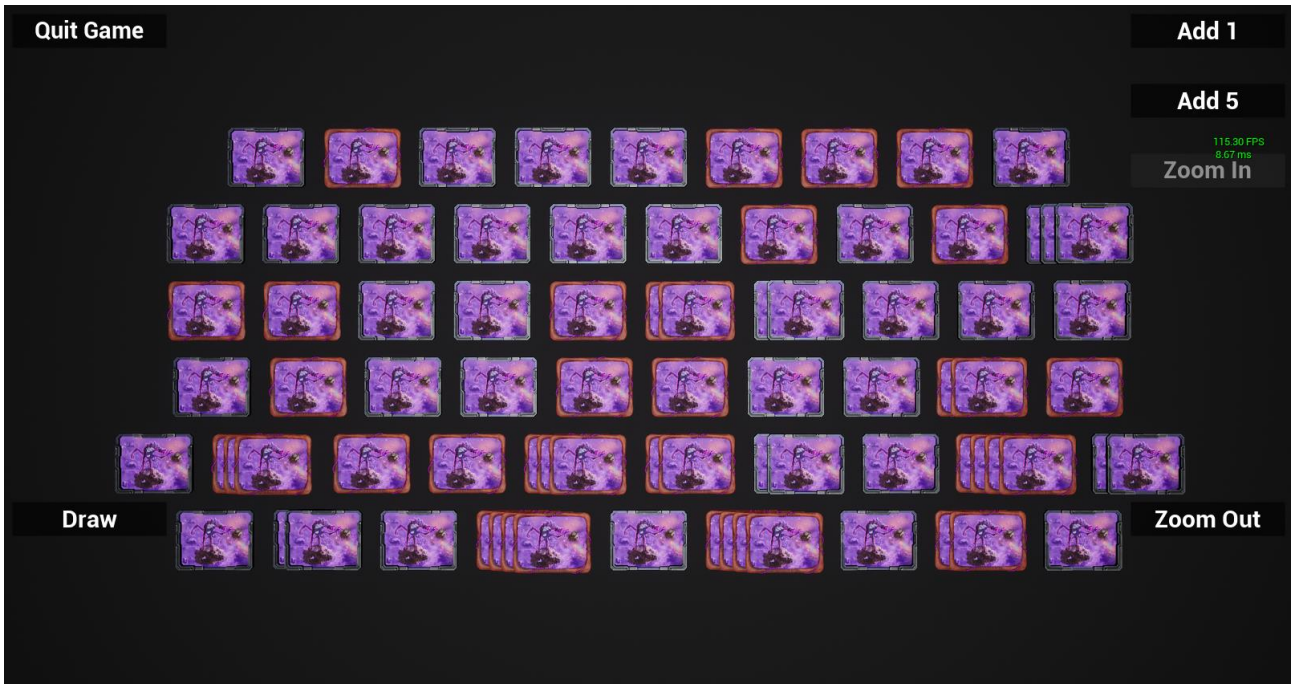


Рисунок 32. Приближение камеры к области игрока

Для того, чтобы выйти из режима приближения, достаточно нажать на кнопку «Zoom Out».

При изменении количества карт в области поля боя при необходимости происходит анимация перемещения карт.

Анимация добавления карты имеет два основных этапа:

1. скейлинг и перемещение рядов, добавление нового ряда при необходимости;
2. перемещение карт в выбранном ряду и добавление карты.

Анимация удаления карты также имеет два основных этапа:

1. удаление карты, а также удаление ряда, если он пуст и если общее количество рядов в области превышает 2;
2. скейлинг и перемещение рядов.

В данном случае анимация перемещения имеет фиксированную длительность, не зависящую от расстояния. Для реализации такого анимированного перехода необходимо иметь:

1. исходный transform;

2. текущий transform (в начальный момент времени равен исходному);

3. целевой transform (рассчитывается заранее).

Далее достаточно провести интерполяцию transform'а от текущего значения до целевого покомпонентно.

Скорость интерполяции местоположения можно задать как:

$$v_{interp} = v_{anim} \cdot dist, \quad (1)$$

где v_{interp} – скорость интерполяции, v_{anim} – скорость анимации (некоторый коэффициент), $dist$ – расстояние между исходным и целевым местоположением.

Скорость интерполяции для масштабирования можно задать как:

$$v_{interp} = v_{anim} \cdot amount, \quad (2)$$

где v_{interp} – скорость интерполяции, v_{anim} – скорость анимации (некоторый коэффициент), $amount$ – «расстояние» между исходным и целевым значением вектора масштаба.

Скорость интерполяции поворота можно задать как v_{anim}, где v_{anim} – скорость анимации (некоторый коэффициент). Для интерполяции поворота необходимо рассчитывать промежуточные значения каждого из компонентов поворота: pitch, yaw, roll.

2.2 Разработка системы «руки» игрока

В контексте компьютерных коллекционных карточных игр объект «рука» представляет собой набор карт, которые игрок имеет на определенном ходу.

При определении наиболее оптимального варианта реализации с точки зрения удобства игрока необходимо оценить:

1. расположение «руки» на поле боя;
2. долю экрана, занимаемую областью «руки»;
3. логику взаимодействия с картами в «руке»: выбор карты, просмотр описания карты, её параметров, способностей и перков;
4. логику выкладывания карты на поле боя.

В рамках данной работы был проведен анализ решений, представленных в следующих играх:

1. Magic: The Gathering Arena (MTG Arena);
2. The Elder Scrolls: Legends (TES Legends);
3. Gwent: The Witcher Card Game;
4. Hearthstone;
5. Shadowverse.

2.2.1. Анализ возможных подходов

Magic: The Gathering Arena

«Рука» представлена объектом снизу экрана по центру, имеющим 2 состояния:

1. компактный режим. Он применён по умолчанию в начале игры. Используется для осмотра поля боя, минимизирует область экрана, используемую «рукой». Активируется при клике на любое свободное место (рис. 33);
2. рабочий режим. В нем карты занимают нижнюю $\frac{1}{4}$ экрана, переход в него производится при клике в области «руки». Используется для управления картами.

Рисунок 33. Просмотр информации о карте в компактном режиме «руки» в MTG Arena

Логика взаимодействия с картами:

1. используется логика удержания нажатия на карте (OnMouseButtonHold). Для того, чтобы открыть карту и начать выкладывать ее на поле боя, требуется удерживать нажатие на ней;

2. примерно через 0.4 секунды после начала удерживания нажатия на карте появляется дополнительная информация о её перках;
3. выкладывание карт производится через drag-and-drop в любую точку экрана;
4. после вытаскивания карты в начале этапа размещения «рука» переходит в компактный режим с пробелом в том месте, где была карта;
5. на время хода противника «рука» переходит в компактный режим. Возвращается в рабочий режим во время хода игрока.

Подробная информация о карте представлена объектом, занимающим $\frac{1}{4}$ экрана: режим ее просмотра открывается при удержании нажатия на карте, появляется 3D-объект, занимающий $\frac{1}{12}$ экрана в воздухе, символизирующий карту. При нажатии на карту на поле боя в течение 0.3 сек. на фоне серого модального окна появляется: слева – изображение карты, справа – список с детальным описанием перков карты.

Итоги:

1. шрифты подобраны неудачно, они тонкие, плохо читаются на мобильных устройствах;
2. информация о карте не должна быть снизу от карты – её нужно помещать в блоки слева и справа (в зависимости от положения карты), как это сделано в других проектах. В текущей реализации это приводит к нерациональному использованию пространства и ухудшает читабельность описаний карт.

The Elder Scrolls: Legends

«Рука» представлена объектом, расположенным в левой нижней части экрана, занимает примерно $\frac{1}{8}$ экрана по высоте, не пересекается с портретом героя. У выкладки карт в «руке» нет изгиба.

Логика взаимодействия с картами:

1. используется логика удержания нажатия на карте (OnMouseButtonHold). Для того, чтоб открыть карту и начать её выкладывать на поле боя, требуется удерживать нажатие на ней;
2. примерно через 0.7 секунды после начала удерживания нажатия на карте появляется дополнительная информация о её перках (рис. 34);
3. выкладывание карт производится через drag-and-drop в необходимую зону поля битвы. Зона, в которую выкладывается карта, подсвечивается;
4. после того, как карта вынута из «руки», до окончательного выкладывания карты остаётся пробел в том месте, где она была.

Рисунок 34. Просмотр информации о карте в «руке» в TES: Legends

Подробная информация о карте представлена объектом, занимающим $\frac{1}{5}$ экрана: режим ее просмотра открывается при удержании нажатия на карте, далее

появляется 3D-объект, занимающий $\frac{1}{30}$ экрана в воздухе, символизирующий карту.

При нажатии на карту на поле боя в течение 0.7 сек на фоне невидимого модального окна появляется:

1. изображение карты – справа или слева – в зависимости от расположения карты на поле боя (рис. 35);
2. список с детальным описанием перков – справа или слева – в зависимости от расположения изображения карты.

Рисунок 35. Просмотр информации о перках карты на поле боя в TES: Legends

Итоги:

1. оформление великолепно, лучшее среди конкурентов;
2. шрифты подобраны удачно, их отлично видно даже на мобильных платформах;
3. иконки и значения атаки и защиты специально увеличены для мобильных платформ;
4. информация о карте снизу минималистична и обычно умещается в максимум 4 строки. Вся остальная информация раскрывается во

всплывающих списках перков и их описаний слева и справа от изображения карты на экране детальной информации;

5. за счёт невидимого модального окна при просмотре детальной информации о карте не происходит переключение внимания с поля боя.

Gwent: The Witcher Card Game

«Рука» представлена объектом снизу экрана по центру, имеющим 2 состояния:

1. компактный режим. Он применён по умолчанию в начале игры. Используется для осмотра поля боя, минимизирует область экрана, используемую «рукой». Активируется при клике на любое свободное место;
2. рабочий режим. В нем карты занимают нижнюю $\frac{1}{4}$ экрана, переход в него производится при клике в области «руки». Используется для управления картами.

Рисунок 36. Просмотр информации о карте в руке в Gwent

Логика взаимодействия с картами:

1. используется логика клика по карте (OnMouseButtonDown) для её выбора и автоматического отображения дополнительной информации (рис. 36, 37);

2. нажатие на карту (OnMouseButtonHold) в течение 1 секунды открывает модальное окно с серым прозрачным фоном с полной информацией о карте (слева направо): изображение, описание карты, описание перков;
3. выкладывание карт производится через drag-and-drop в необходимую зону поля битвы. Зона, в которую выкладывается карта, подсвечивается. «Рука» переводится в компактный режим. Карта, которая вынимается из «Руки», выдвигается в ней вперёд. Появляется стрелка, следующая за курсором мыши и показывающая, куда будет выложена карта. Полупрозрачный «Аватар» карты того же размера, что и другие карты на поле боя, появляется под курсором мыши;
4. при переходе хода к противнику «Рука» переходит в компактный вид, камера центрируется на поле боя.

Рисунок 37. Просмотр информации о карте на поле боя в Gwent

Итоги:

1. логика клика по карте (OnMouseButtonDown) реализована крайне удачно;
2. шрифты подобраны неплохо, их хорошо видно даже на мобильных платформах;
3. значения мощи карты специально увеличены для мобильных платформ;
4. информация о карте снизу не отображается;

5. серое полупрозрачное модальное окно отрывает от поля битвы, нарушает восприятие.

Hearthstone

«Рука» представлена объектом, расположенным в правой нижней части экрана и имеющим 2 состояния:

1. компактный режим. Он применён по умолчанию в начале игры. «Рука» переводится в него при клике на любое свободное место. Можно полноценно работать с «рукой» в компактном режиме: просматривать детальную информацию о картах, выкладывать карты на стол. «Рука» различает обычный клик для перехода из компактного режима в рабочий и клик с удержанием для просмотра информации о карте и выкладывания карты на поле боя;
2. рабочий режим – карты занимают нижнюю $\frac{1}{4}$ экрана, переход в него производится при клике в области «руки». Используется для управления картами. В этом режиме «рука» закрывает портрет героя и его способность (рисунок 38).

Рисунок 38. Рабочий режим объекта «рука» в Hearthstone

Логика взаимодействия с картами:

1. используется логика удержания нажатия на карте (OnMouseButtonHold). Для того, чтобы открыть карту и начать её выкладывать на стол, требуется удерживать нажатие на ней;
2. нажатие на карту (OnMouseButtonHold) в течение 0.5-0.7 секунды открывает дополнительную информацию о перках карты (рис. 39);
3. выкладывание карт производится через drag-and-drop в любую точку экрана;
4. после того, как карта вынута из «руки», «рука» сохраняет режим с пробелом в том месте, где была карта. Исключение – использование карт способностей главного героя: в таком случае «рука» переходит в компактный режим, чтобы перевести фокус на героя;
5. на время хода противника «рука» переходит в компактный режим, не возвращается в рабочий режим во время хода игрока.

Рисунок 39. Просмотр информации о карте в компактном режиме «руки» в Hearthstone

Итоги:

1. прямые шрифты подобраны неплохо, изогнутые читаются хуже;
2. информация о карте представлена лаконично, нет отдельных экранов для её просмотра – это большой плюс, но этот подход портится

принципом удержания нажатия на карту (OnMouseButtonHold), который должен быть заменён на обычный клик;

3. поле боя имеет фиксированный размер, согласно науке о минимальной площади экранов с разным соотношением сторон. В современном мире нужно использовать адаптивные поля боя – поскольку такая техническая возможность есть, это намного интереснее, но в целом удобство от используемого подхода не страдает.

Shadowverse

«Рука» представлена объектом справа, имеющим 2 состояния:

1. компактный режим. Он применён по умолчанию в начале игры. «Рука» переводится в него при клике на любом свободном месте. Непригоден для управления картами, используется для осмотра поля боя.
2. рабочий режим – карты занимают нижнюю $\frac{1}{4}$ экрана, переход в него производится по клику на «руке». Используется для управления картами. В этом режиме «рука» закрывает портрет героя и его способность. Здоровье героя и цифра доп. способности приподнимаются при переводе «руки» в рабочий режим.

Логика взаимодействия с картами:

1. используется логика клика по карте (OnMouseButtonDown) для её выбора и автоматического отображения дополнительной информации в левом верхнем углу, вне зависимости от положения карты. При попытке выложить карту или осуществить атаку виджет в левом верхнем углу скрывается;
2. дополнительные модальных окна не создаются. Вся информация о карте предоставляется по клику (п.1);
3. выкладывание карт производится через drag-and-drop в любую точку экрана. «Рука» при этом не меняет режим отображения, оставаясь в рабочем. На месте вынутой карты остаётся пустота, пока карта не выложена на поле боя;

4. при переходе хода к противнику «рука» не переходит в компактный вид, дополнительные манипуляции не производится.

Подробная информация о карте представлена объектом с минималистичным портретом карты. Данный объект (виджет) появляется всегда в одном и том же месте – в левом верхнем углу, вне зависимости от положения карты в «руке» или на поле боя. При попытке выложить карту на поле боя появляется 3D-объект, занимающий $\frac{1}{24}$ экрана в воздухе, символизирующий карту. Карта в «руке» в этот момент скрывается. Отдельного окна для детального просмотра информации о карте нет – всё отображается в виджете в левом верхнем углу.

Итоги:

1. UI использует очень удачную схему работы с картами через клик по ним, удерживать нажатие (OnMouseButtonHold) не требуется;
2. применена интересная и удачная идея унифицированного отображения информации о карте в виджете в левом верхнем углу;
3. Шрифты подобраны хорошо, отрисовка текста о способностях карты происходит в screen space виджете в левом верхнем углу.

2.2.2. Основные принципы реализации

В результате обзора существующих решений были приняты следующие подходы для реализации взаимодействия игрока с картами в «руке»:

1. выкладка карт из «руки» производится либо с помощью drag-and-drop, либо при нажатии на кнопку «выложить карту»;
2. при наведении (ПК) либо нажатии (на мобильных платформах) на карту должен появляться объект с информацией о карте;
3. при клике (ПК) либо втором нажатии (на мобильных платформах) на карту должен происходить её выбор;
4. при выборе карты должна появляться опция выкладки карты на поле боя.

Для объекта с информацией о карте выделены следующие принципы:

1. в случае карты в «руке» – объект расположен над картой. Для карт, расположенных на правой стороне экрана – объект может смещаться левее от центра карты, чтобы не вылезать за границы экрана. Информация о карте должна быть расположена слева от объекта. Для карт, расположенных на левой стороне экрана – объект может смещаться правее от центра карты, информация о карте должна быть расположена справа от объекта;
2. в случае карты на поле боя – расположение объекта определяется тем, где расположена карта. Для карт, расположенных на правой стороне экрана – объект появляется слева, информация о карте – еще левее. Для карт, расположенных на левой стороне экрана – объект появляется справа, информация о карте – еще правее;
3. информация о карте отображается в блоках, находящихся в контейнере. При необходимости добавляется возможность вертикального скроллинга;

Для контейнера «рука» выделены следующие принципы:

1. контейнер привязан к нижнему краю экрана по центру;
2. заполнение производится с выравниванием по левой стороне;
3. размер рассчитывается исходя из параметров экрана.

2.2.3. Программная реализация

Для реализации объекта «рука» были созданы следующие классы:

1. AHand – класс, производный от AActor [1][11]. Представляет собой саму «руку» игрока. Содержит функциональность добавления и удаления карт, а также их выбора. Выполняет перерасчет своих габаритов при изменении соотношения сторон [2] viewport’a;
2. UHandEventDispatcher – класс, производный от UActorComponent. Реализует функциональность очереди для событий «руки»;
3. UHandControls – виджет, который позволяет брать новые карты в «руку», а также, при условии наличия выбранных карт, позволяет выкладывать их на поле боя. Является производным классом от UUserWidget (UMG-виджет);
4. ACardInfo – класс, производный от AActor. Позволяет отображать информацию о карте и ее способностях;
5. UCardAbilityInfo – виджет, который может хранить в себе информацию о способности карты. Отображает название способности и ее описание;
6. UCardInfo – виджет, который может хранить в себе информацию обо всех способностях карты. Представляет собой контейнер-сетку (UUniformGridPanel) для виджетов отдельных способностей;
7. UCardSelectionComponent – класс, производный от UActorComponent. Хранит данные о выбранных на текущий момент картах.

Основной класс объекта «рука» – AHand. Он предоставляет всю необходимую функциональность для внешнего взаимодействия. Основные задачи: добавление и удаление карт, их выбор, расчет локальных трансформов содержимого, расчет собственных габаритов в зависимости от соотношения сторон viewport’a. Для добавления/удаления карт используется очередь (UHandEventDispatcher). Необходимость ее использования исходит из наличия анимации перемещения карт – каждое событие должно обрабатываться отдельно, по завершению обработки предыдущего.

Как и для карт, расположенных на поле боя, для рендеринга карт в руке используется система рендер пулов (п. «Повышение производительности при рендеринге групп 3d-объектов»).

При изменении размера окна происходит перерасчет габаритов «руки», а также трансформов карт в ней по следующим правилам:

1. по вертикали «рука» должна занимать 20% высоты экрана;
2. по горизонтали «рука» должна занимать 80% экрана (необходимо для дальнейшего помещения элементов пользовательского интерфейса);
3. рука должна быть привязана к нижнему краю экрана по центру.

Для определения местоположения «руки» необходимо получить мировые координаты, соответствующие центру нижнего края экрана, а также направление между точкой, в которой находится камера, и точкой с полученными мировыми координатами. Задав расстояние между камерой и целевой точкой, можно получить результирующее местоположение по формуле:

$$l_h = l_{deproj.} + dist \cdot d_{deproj.}, \quad (3)$$

где:

l_h – положение (location) «руки»;

l_{deproj} – мировые координаты, соответствующие центру нижнего края экрана (результат депроекции);

d_{deproj} – направление между точкой, в которой находится камера, и точкой с полученными мировыми координатами (результат депроекции);

$dist$ – желаемое расстояние между камерой и результирующей точкой.

На рисунках 40 – 42 изображена «рука» при различных соотношениях сторон. Как можно заметить, во всех случаях она остается снизу по центру экрана. Процент экрана, занимаемый контейнером «руки», тоже остаётся константным.

Рисунок 40. «Рука» при соотношении сторон 16:9

Рисунок 41. «Рука» при соотношении сторон 21:9

Рисунок 42. «Рука» при соотношении сторон 4:3

Для обозначения выбранных карт был создан материал M_SelectionBackground. Он позволяет варьировать цвет подсветки выбранных карт с помощью передаваемого на вход индекса. На рисунке 43 изображен его граф в Material Editor:

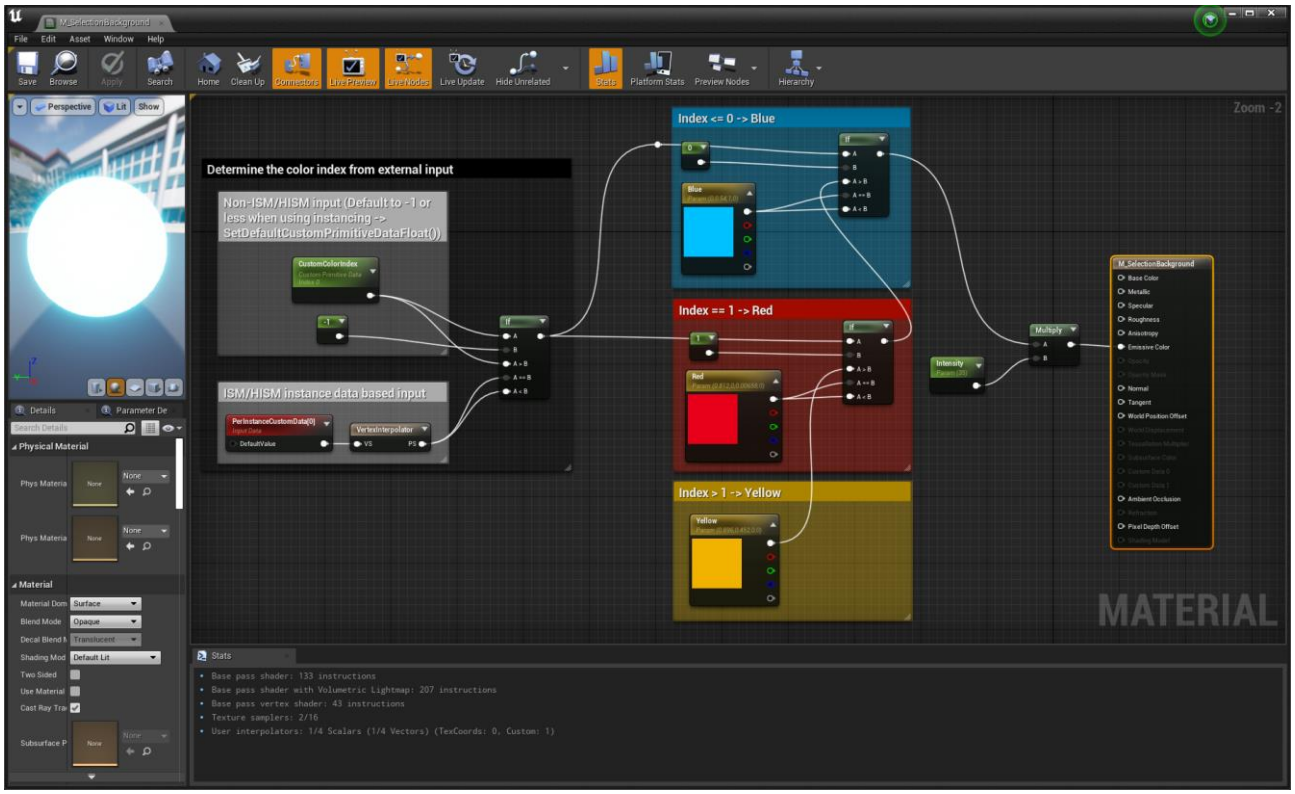


Рисунок 43. Материал подсветки выделенных карт.

В зависимости от переданного индекса изменяется цвет излучения. Цель – индикация выбранных карт в различных сценариях. Например, выбранные противником карты должны подсвечиваться красным цветом, свои карты – голубым. Варианты представлены на рисунке 44.

Рисунок 44. Варианты подсветки выделенных карт.

Задать индекс цвета можно как для инстансов определенного ISM/HISM (с помощью PerInstanceCustomData, так и для actor’ов с обыкновенными UStaticMesh (с помощью CustomPrimitiveData).

При нажатии на карту в руке происходит ее выбор. При повторном нажатии – снятие выбора. При наличии хотя бы одной выбранной карты активируется кнопка выкладки на поле боя «Play». Пример выбранных карт изображен на рисунке 45:

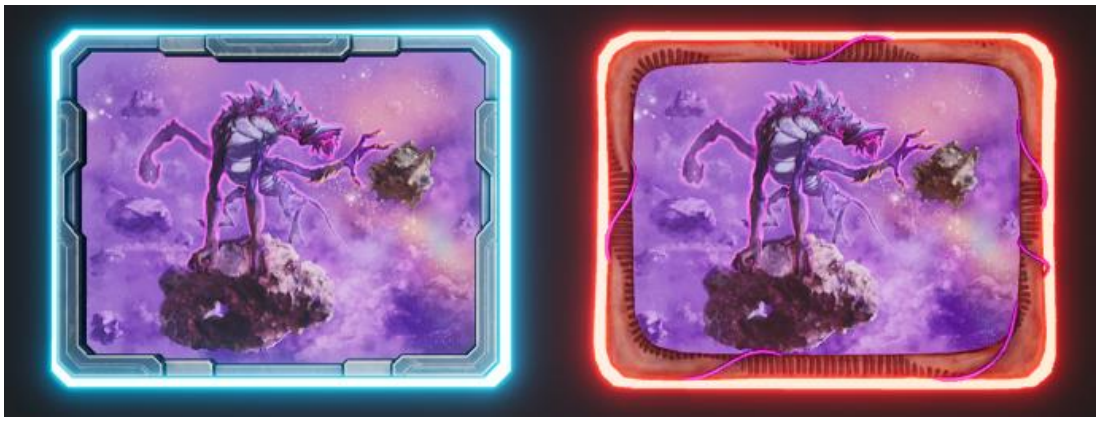


Рисунок 45. Выбранные карты

При наведении курсора мыши на карту в «руке» открывается режим просмотра информации. Он содержит в себе увеличенную модель карты и описание ее способностей (рис. 46). Сторона, с которой должен отображаться виджет с описанием, определяется текущим положением карты. Если она находится в правой половине экрана, информация отображается слева, иначе – справа.

Рисунок 46. Информация о способностях карты

3. Повышение производительности при рендеринге групп 3d-объектов

При создании 3d-приложений с рендерингом в реальном времени используется множество различных способов оптимизации на разных уровнях.

Например, при создании 3d-моделей для дальнейшего использования, оптимизация заключается в понижении количества полигонов при сохранении требуемого визуального уровня детализации. Чем меньше полигонов содержит модель, тем меньше вычислительных ресурсов требуется на ее обработку.

При условии, что модель не статична относительно камеры, дополнительная оптимизация заключается в создании альтернативных, менее детализированных версий модели, и их дальнейшей подмене в зависимости от расстояния. Данная техника – Level of Detail (LOD) – позволяет дополнительно понизить обрабатываемое количество полигонов модели.

При отрисовке сцен используется подход, называемый culling (отбраковка). Он позволяет заранее отбросить те объекты, которые не будут находиться в области видимости камеры. Например, объекты, находящиеся вне зоны обзора (frustum culling), либо объекты, полностью перекрытые чем-либо (depth culling).

Также используется механизм батчинга геометрии. Его суть заключается в объединении запросов на отрисовку объектов с различными моделями и материалами. За счёт этого достигается понижение нагрузки на CPU.

При отрисовке множества однообразных объектов, т.е. объектов, имеющих одну и ту же модель и материалы, используется техника, называемая инстансингом геометрии. Именно о ней и пойдет речь в данной работе.

3.1.1 Механизм инстансинга геометрии

Инстансинг геометрии – один из способов оптимизации рендеринга. Он используется в тех случаях, когда необходимо отрисовывать большое количество однообразных объектов [3][8]. Основная его цель – понижение количества запросов на отрисовку (draw calls).

Формированием запросов на отрисовку занимается центральный процессор (CPU), их исполнением (отрисовкой) – видеокарта (GPU). При большом количестве простых для исполнения запросов возникает проблема: видеокарта слишком быстро справляется с отрисовкой и начинает простаивать в ожидании. Таким образом, производительность ограничивается возможностями процессора.

Инстансинг частично решает данную проблему. Вместо того, чтобы обрабатывать каждую копию объекта по отдельности, он позволяет объединить их обработку в единый запрос на отрисовку. Таким образом, общее количество запросов падает, а их «сложность» растет. За счёт этого происходит перераспределение затрат на обработку между процессором и видеокартой. Понижается зависимость итоговой производительности от возможностей процессора, а также понижается время простоя видеокарты.

В рамках движка Unreal Engine 4 существуют два компонента, реализующие механизм инстансинга [9]:

1. UInstancedStaticMeshComponent (ISMC);

2. UHierarchicalInstancedStaticMeshComponent (HISMC).

HISMC является дополненной версией ISMC. Он имеет поддержку Level of Detail (LOD).

3.2. Разработка системы пулов отрисовки (Render Pools)

В рамках движка Unreal Engine 4 с помощью существующих компонентов инстансинга (ISMC/HISMC) можно добиться значительного прироста производительности. Как правило, однако, они создаются на уровне использующих их actor’ов или иных компонентов. Будем называть их *локальными* компонентами инстансинга. Принцип изображен на рисунке 47. У этого существуют следующие недостатки:

1. при использовании одних и тех же мешей и материалов разными actor’ами не происходит их объединение при обработке;
2. при использовании нескольких мешей и материалов в рамках одного actor’а приходится создавать отдельные ISMC/HISMC для каждой комбинации, что частично противоречит изначальной цели инстансинга – появляются дополнительные запросы на отрисовку.

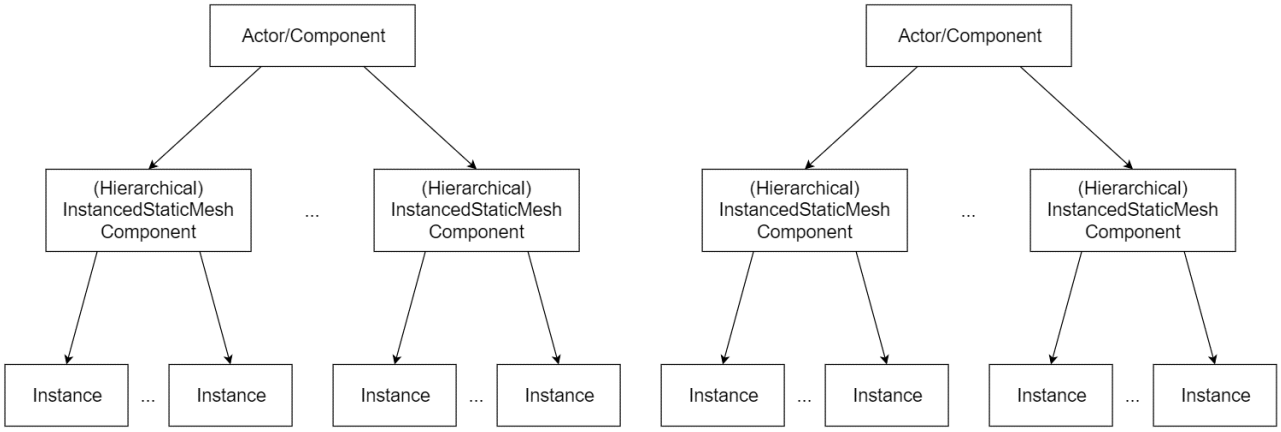


Рисунок 47. Схема использования локальных компонентов инстансинга

От вышеперечисленных недостатков можно избавиться «централизацией» инстансинга – вынесением ISMC/HISMC с уровня actor’а на уровень сцены. Будем называть их *глобальными* компонентами инстансинга:

1. все копии объектов с одинаковыми комбинациями мешей и материалов будут обрабатываться единым глобальным компонентом;
2. при использовании нескольких мешей и материалов в рамках одного actor’а не придется создавать множество локальных компонентов – достаточно будет хранить индексы инстансов и указатели на соответствующие глобальные компоненты.

Система пулов отрисовки (render pools) внедряет вышеописанный подход. Ее задача – максимально уменьшить количество существующих ISMC/HISMC за счёт централизации работы с ними. Принцип изображен на рисунке 48.

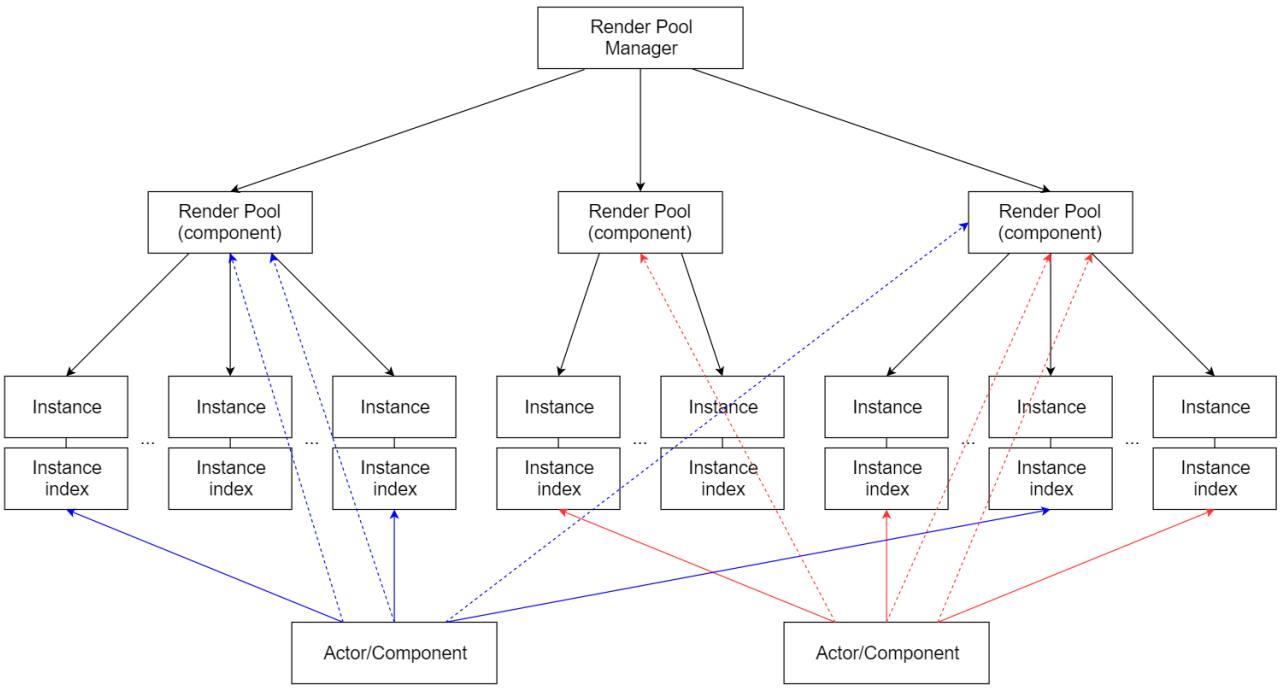


Рисунок 48. Схема использования системы пулов отрисовки

3.2.1. Принципы реализации

Разработка проводилась с помощью средств движка Unreal Engine 4 на языке программирования C++.

В процессе разработки были созданы два класса:

1. URenderPoolComponent;

2. ARenderPoolManager.

Для локального хранения данных об инстансах была создана структура FRenderPoolIndexData, содержащая в себе индекс инстанса и указатель на URenderPoolComponent.

Для хранения данных об ассетах, которые должны использоваться в системе пулов отрисовки, создана структура FActorAssetData. Она хранит в себе данные о меше и материалах, которые должны использоваться в render pool’е.

URenderPoolComponent является надстройкой над HISMC – он хранит указатели на внешние массивы с FRenderPoolIndexData. Цель хранения заключается в их дальнейшем обновлении: при удалении или добавлении инстанса в соответствующий пул отрисовки для соблюдения актуальности и целостности данных необходимо иметь обратную связь с внешними массивами, которые находятся в использующих систему классах. Для настройки обратной связи необходимо сначала привязать внешний массив к желаемому пулу.

ARenderPoolManager контролирует жизненный цикл render pool’ов, а также управляет созданием/удалением инстансов. Получить, создать, удалить необходимый пул можно по FActorAssetData.

Рассмотрим классы и структуры:

1. URenderPoolComponent – класс, производный от HISMC. Позволяет обновлять внешние массивы с индексами инстансов при их добавлении/удалении. Для автоматического обновления необходимо привязать соответствующие массивы. Несмотря на то, что данный класс самодостаточен, для работы с системой пулов отрисовки рекомендуется использовать менеджер;
2. ARenderPoolManager – класс, производный от AActor. Позволяет создавать, удалять пулы, а также определять их по FActorAssetData. Предоставляет высокоуровневый доступ к управляемым пулам;
3. FRenderPoolIndexData – структура, содержащая данные об индексе инстанса и его пуле;
4. FActorAssetData – структура, содержащая данные о меше и материалах.

3.2.2. Анализ производительности

В рамках работы был проведен анализ производительности с инстансингом и без. Для сравнения было измерено:

1. количество кадров в секунду (FPS);
2. количество отрисованных примитивов сцены (треугольников);
3. количество вызовов отрисовки (draw calls).

Измерения проводились поэтапно для 2^n, $n = 3, 4, \ldots$ actor'ов/инстансов. Объекты располагались в виде сетки $2^{0.5n} \times 2^{0.5n}$ для чётных n, $2^{0.5n+1} \times 2^{0.5n}$ для нечётных. Высота камеры изменялась в зависимости от количества actor'ов на сцене так, чтобы все объекты попадали в поле зрения. Принцип изображён на рисунке 49.

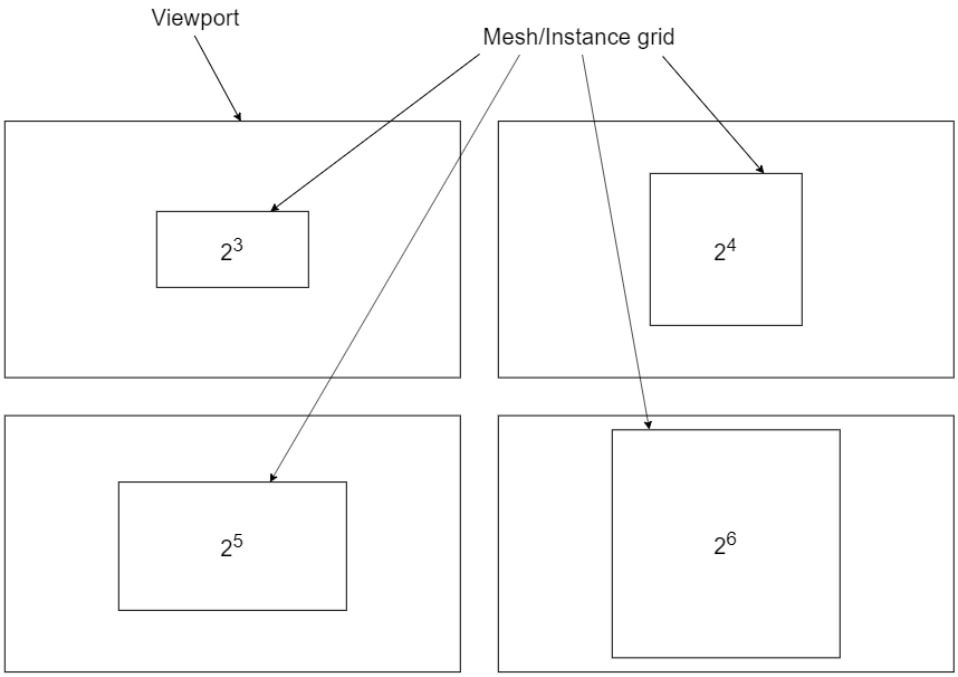


Рисунок 49. Способ расположения actor'ов/инстансов

На рисунке 50 изображен график зависимости FPS от количества actor'ов/инстансов при использовании инстансинга (ISMC) и без (множество actor'ов с SMC – UStaticMeshComponent):

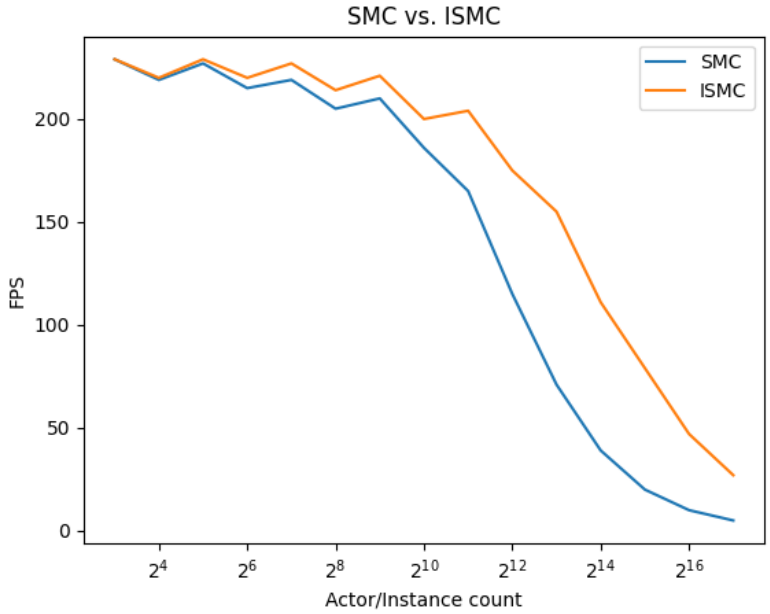


Рисунок 50. График зависимости FPS от количества actor’ов/инстансов

На рисунке 51 изображен график зависимости количества отрисованных треугольников от количества actor’ов/инстансов для аналогичных случаев:

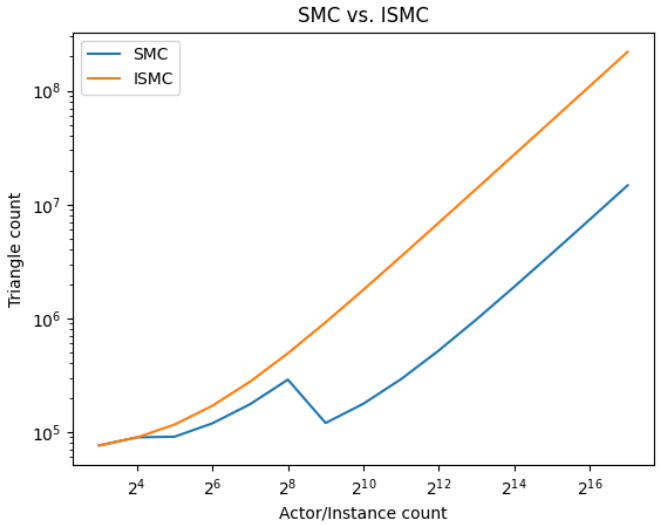


Рисунок 51. График зависимости количества треугольников от количества объектов/инстансов

Можно заметить, что на графике SMC, представляющем множество отдельных actor’ов с UStaticMeshComponent, присутствуют участки, где количество треугольников либо временно перестаёт расти, либо падает. Это обусловлено тем, что, в отличие от UInstancedStaticMeshComponent, UStaticMeshComponent поддерживает использование LOD. При 2^5 actor’ов происходит замена LOD0 на LOD1, а при 2^9 actor’ов – замена LOD1 на LOD2.

На рисунке 52 изображен график зависимости количества вызовов отрисовки от количества actor’ов/инстансов:

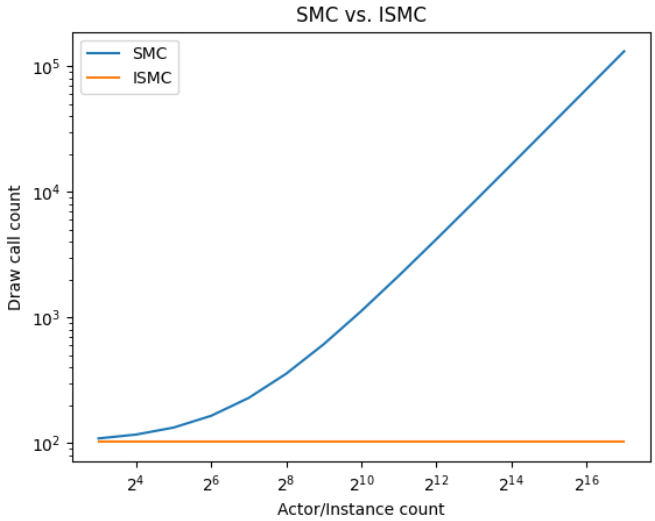


Рисунок 52. График зависимости количества draw calls от количества actor’ов/инстансов

При использовании инстансинга наблюдается постоянное количество вызовов отрисовки, в то время как рендеринг каждого actor’а требует отдельные draw calls, повышая нагрузку на CPU.

Подводя итоги, можно сделать следующие выводы:

- использование instancing’а крайне выгодно в случае необходимости отрисовывать большое количество однотипных объектов;
- при использовании instancing’а наблюдается рост FPS, даже без поддержки LOD;
- при использовании instancing’а количество вызовов отрисовки константно.

Следовательно, использование инстансинга в рассматриваемом сценарии полностью целесообразно. Единственным его значительным недостатком в рамках Unreal Engine 4 являются проблемы с тенями, однако, в данном случае, они не требуются.

Также был проведен анализ производительности при использовании UInstancedStaticMeshComponent и UHierarchicalInstancedStaticMeshComponent. Методика тестирования аналогична описанной ранее.

На рисунке 53 изображен график зависимости FPS от количества инстансов при использовании ISMC и HISMC:

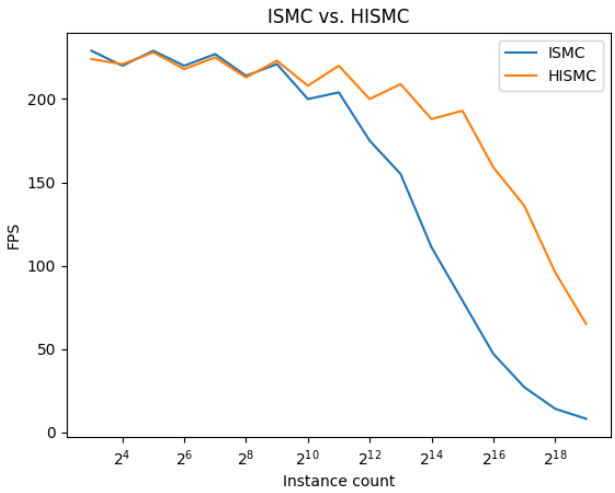


Рисунок 53. График зависимости FPS от количества инстансов

Глядя на график выше, можно заметить, что при большом ($> 2^9$) количестве инстансов количество кадров в секунду при использовании HISMC значительно выше, чем при использовании ISMC. Повышение производительности достигается за счёт того, что HISMC поддерживает LOD. При 2^9 инстансах начинает использоваться LOD2, имеющий всего 50 треугольников (LOD0 имеет 840), следовательно, общий triangle count одного draw call'а значительно понижается, результируя в повышении FPS. LOD1 особых изменений в производительность не вносит, поскольку при том количестве инстансов, при котором он используется ($2^5 - 2^8$), общий triangle count всё ещё достаточно низок, чтобы заметить какие-либо значительные изменения.

На рисунке 54 изображён график зависимости количества отрисовываемых треугольников от количества инстансов:

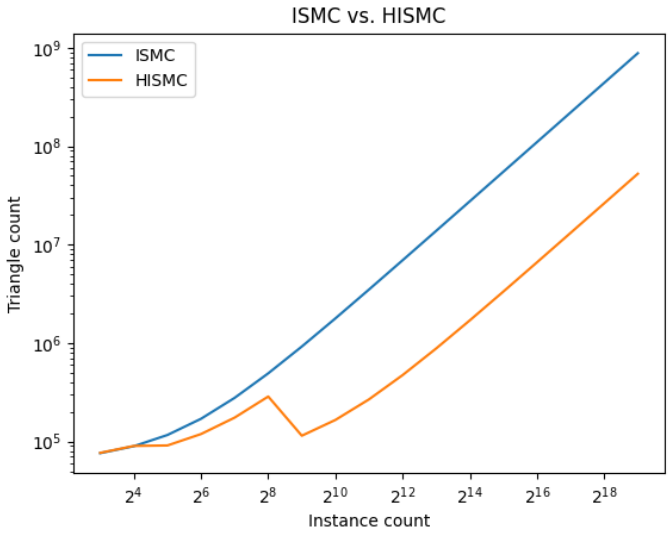


Рисунок 54. График зависимости количества треугольников от количества инстансов

В данном случае наблюдается результат, аналогичный полученному в сравнении производительности с инстансингом и без (рисунок 48). В диапазоне от 2^4 до 2^5 инстансов заметно снижение роста triangle count из-за замены LOD0 на LOD1. В диапазоне от 2^8 до 2^9 инстансов происходит понижение triangle count ввиду замены LOD1 на LOD2.

Из результатов анализа следует, что HISMC выгодно использовать при наличии и применении LOD, значительно понижающих triangle count модели.

В большинстве сценариев, которые могут иметь место в рассматриваемой карточной игре, количество карт не дойдет до 2^9 штук. Следовательно, в стандартных ситуациях, разница между ISMC и HISMC с точки зрения производительности наблюдаться не будет. Однако все же есть несколько причин, по которым стоит отдать предпочтение HISMC:

- использование LOD частично помогает бороться с алиасингом;
- в экстремальных случаях ($> 2^9$ карт) производительность будет гораздо выше и стабильнее.

ЗАКЛЮЧЕНИЕ

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.

Также в рамках данной работы были изучены различные методы оптимизации природных сцен.

Также в рамках данной работы были разработаны элементы компьютерной коллекционной карточной игры. Созданы, оптимизированы и подготовлены для использования в движке 3d-модели карт. Проведен анализ реализаций внутриигровых систем в существующих продуктах. Разработаны следующие внутриигровые системы: система выкладки карт на поле боя, система «руки» игрока.

Проведена оптимизация отрисовки большого количества однообразных 3d-объектов. Для достижения этой цели разработана система пулов отрисовки (Render Pool System), которая обобщает и упрощает работу с инстансингом геометрии.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Unreal Engine 4 Documentation // Unreal Engine Documentation URL: https://docs.unrealengine.com/. Дата обращения: 07.04.2022;
2. Display aspect ratio // Wikipedia, the free encyclopedia URL: https://en.wikipedia.org/wiki/Display_aspect_ratio/. Дата обращения: 13.04.2022;
3. Geometry instancing // Wikipedia, the free encyclopedia URL: https://en.wikipedia.org/wiki/Geometry_instancing. Дата обращения: 21.04.2022;
4. Modeling – Blender Manual // Blender Manual URL: https://docs.blender.org/manual/en/latest/modeling/index.html. Дата обращения: 18.02.2022;
5. Mipmap // Wikipedia, the free encyclopedia URL: https://en.wikipedia.org/wiki/Mipmap. Дата обращения 05.03.2022;
6. Level of Detail (computer graphics) // Wikipedia, the free encyclopedia URL: https://en.wikipedia.org/wiki/Level_of_detail_(computer_graphics). Дата обращения: 05.04.2022;
7. Creating and Using LODs // Unreal Engine Documentation URL: https://docs.unrealengine.com/4.27/en-US/WorkingWithContent/Types/StaticMeshes/HowTo/LODs/. Дата обращения: 05.04.2022;
8. Инстансинг [Электронный ресурс] // Habr: интернет-портал. URL: https://habr.com/ru/post/352962/. Дата обращения: 03.04.2022;
9. UE4 Optimization: Instancing // YouTube: видео хостинг. URL: https://www.youtube.com/watch?v=oMIbV2rQO4k. Дата обращения: 03.04.2022;
10. Божко А.Н., Жук Д.М., Маничев В.Б. Компьютерная графика. [Электронный ресурс] // Учебное пособие для вузов. − М.: Изд-во МГТУ им. Н. Э. Баумана, 2007. - 389 с., - ISBN 978-5-7038-3015-4, Режим доступа: http://ebooks.bmstu.ru/catalog/55/book1141.html. Дата обращения: 10.02.2022;
11. Programming Quick Start // Unreal Engine Documentation URL: https://docs.unrealengine.com/5.0/en-US/unreal-engine-cpp-quick-start/. Дата обращения: 29.12.2021.