НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
Кафедра автоматизованих систем обробки інформації і управління

УДК: 004

«До захисту допущено»
В.о. завідувача кафедри
_____ *О.А.Павлов*
(підпис) (ініціали, прізвище)
“___” _____2019 р.

Дипломний проект
на здобуття ступеня бакалавра

з напряму підготовки *6.050101 «Комп’ютерні науки»*

на тему: « *Трасування променів в реальному часі для комп'ютерних ігор* »

Виконав: студент *4* курсу, групи *ІС-з51*
Степовий Тарас Олександрович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник *старший викладач, к.е.н. Родіонов П.Ю.* _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з графічної документації *доц., к.т.н., доц. Тєлишева Т.О.* _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент Степовий Т.О. _____
(підпис)

Київ – 2019 р.

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломного проекту складається з шести розділів, містить 23 рисунки, 12 таблиць, 1 додаток, 22 джерел.

Дипломний проект присвячений візуалізації та демонстрації роботи алгоритму трасування світлових променів в сучасних комп'ютерних іграх.

У розділі інформаційного забезпечення наведені цілі та актуальність роботи – реалізм сучасних комп'ютерних ігор та потенційний вплив алгоритму трасування променів на подальший їх розвиток.

Розділ математичного забезпечення присвячений безпосередній алгоритмічній складності алгоритму трасування променів та його математичній реалізації.

У технологічному розділі описані використаний інструментарій, засоби та технології необхідні для реалізації програмного продукту

ТРАСУВАННЯ ПРОМЕНІВ, КОМП'ЮТЕРНІ ІГРИ

ABSTRACT

Structure and scope of work. Explanatory note of the diploma project consists of six sections, contains 23 figures, 12 tables, 1 applications, 22 sources.

The diploma project is devoted to visualization and demonstration of the light tracing algorithm and its application to modern computer games.

The section of the information support provides the goals and relevance of the work - the realism of modern computer games and the potential impact of the ray tracing algorithm on their further development.

The section of mathematical support is devoted to the direct algorithmic complexity of the ray tracing algorithm and its mathematical implementation.

The technological section describes the used tools, instruments and technologies needed to implement the software product.

RAY TRACING, COMPUTER GAMES

ЗМІСТ

ВСТУП

Головною ціллю комп'ютерної графіки завжди було створення найбільш реалістичних та наближених до правди світів, проте це завжди було надзвичайно важко та майже неможливо, адже дуже важко просимулювати кожен візуальний аспект світу та передати стовідсоткове його відображення на екрани моніторів. Насправді ж така ціль просто неможлива адже кожен комп'ютер та обчислювальна техніка має чіткі межі своєї обчислювальної потужності – що і підводить до цілі комп'ютерної графіки в наш час: передати найбільш реалістичну картинку користувачу використавши найменшу кількість ресурсів.

Цей процес загалом зветься растерізацією – розподілення всіх об'єктів в світі на елементи зрозумілі комп'ютеру та його процесору, а саме використання простих елементів під назвою полігони, що представляють собою трикутники з яких складаються всі елементи стимульованого 3D світу[1], що потім перетворюються на зображення на екрані монітору.

Растеризація визначає, яку частину дисплею охоплює кожен полігон. Якщо дивитись близько однин трикутник може охоплювати весь екран, а якщо він знаходиться далі й розглядається під кутом, то він може охоплювати лише кілька пікселів. Після визначення показуваних пікселів переходять до таких речей, як текстури та освітлення.

Виконання цього для кожного полігону[2] для кожного кадру видається досить затратним процесом, оскільки багато полігонів виявляються прихованими (тобто за іншими полігонами) й їх відображення є явна трата ресурсів комп'ютеру. Протягом багатьох років підходи до растеризації та обладнання поліпшилися, щоб зробити її набагато швидше, а сучасні ігри можуть прийняти мільйони потенційно видимих полігонів і обробити їх за долі секунди.

Проте трасування променів – це дещо інший підхід. Головна його ціль є симуляція світлових променів та їх трасування до поки вони не перетнуться з камерою через яку на віртуальний світ дивиться користувач. Проте мінус такого підходу є його надзвичайна ресурсо-затратність.

Проектування одного єдиного променю включає в себе підрахунок полігону в який промінь влучив та його колір після того як його освітили променем. Тільки беручи до увагу даний факт можна уявити наскільки вимогливий є алгоритм. Насправді ж для отримання справедливого зображення, що схоже на реальність потрібно набагато більше променів за один на піксель, оскільки якщо промінь перетинається з об’єктом треба також брати до уваги інші джерела світла та їх промені. Також не слід забувати й матеріал з якого зроблена поверхня – чи вона є дзеркальною, матовою, тощо.

Щоб визначити кількість світла, що падає на один піксель з одного джерела світла, формула трасування променів повинна знати, наскільки далеко знаходиться джерело світла, наскільки воно яскраве, і кут відбивної поверхні відносно кута джерела світла. Потім процес повторюється для будь-якого іншого джерела світла, включаючи непряме освітлення від світла, відбитого від інших об'єктів сцени. Розрахунки застосовуються до матеріалів, що визначаються рівнем їх розсіяності чи дзеркальної відбивної здатності або обох. Прозорі або напівпрозорі поверхні, такі як скло або вода, заломлюють промені, а не відбивають, додаючи складності розрахункам. Тож всім променям задають деякий параметр відбивання, що обмежує кількість раз, що промінь може відбитися від поверхні.

Проте всі трасувальні розрахунки стали бути можливими на звичайних комп'ютерах завдяки новій лінійці графічних процесорів компанії Nvidia та нових драйверів, що покращують використання ресурсів останнього покоління старих процесорів. Саме тому як ціль даної роботи було поставлено розробити ігровий рівень для візуалізації та використання найновіших трендів в сфері гейм-деву – трасування променів в реальному часі.

Ключовим терміном є реальний час адже саме це є найновішим науковим досягненням в сфері комп'ютерних ігор – трасувати промені на кожному зміненому кадрі гри, під час ігрової сесії в реальному часі. Маючи на увазі поставлену мету, в роботі було реалізовано та оптимізовано роботу алгоритму трасування, створений ігровий майданчик для найкращого відображення використаних технологій, створено систему освітлення та відбиття променів від дзеркальних поверхонь.

Також однією з головних цілей вважається підтримка стабільної частоти кадрів впродовж всієї демонстрації рівня – саме це на даний момент вважається однією з головних проблем здійснення відбиття променів в реальному часі.

Дипломний проект присвячений розробці та оптимізації стабільного трасувальника світлових променів засобами ігрового движка Unreal Engine 4.

Практичне значення одержаних результатів. Розроблений ігровий демонстраційний рівень для одного гравця та оптимізовано алгоритм трасування променів.

Публікації. Результати роботи не були опубліковані.

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Опис предметного середовища

Головним предметом дослідженням даної роботи є знаходження оптимального алгоритму трасування променів та його практичне застосування на ігровій інтерактивній сцені в ігровому движці.

Трасування променів - це інший підхід, до комп'ютерної графіки і загалом до відображення 3D об'єктів, що в основному працює завдяки рекурсивним розрахункам трасування променів, щоб отримати вражаючі зображення, що включають тіні, відображення та багато іншого. Проблема полягає в тому, що для цього потрібно набагато складніші розрахунки аніж класична для поточних комп'ютерних ігор растеризація.

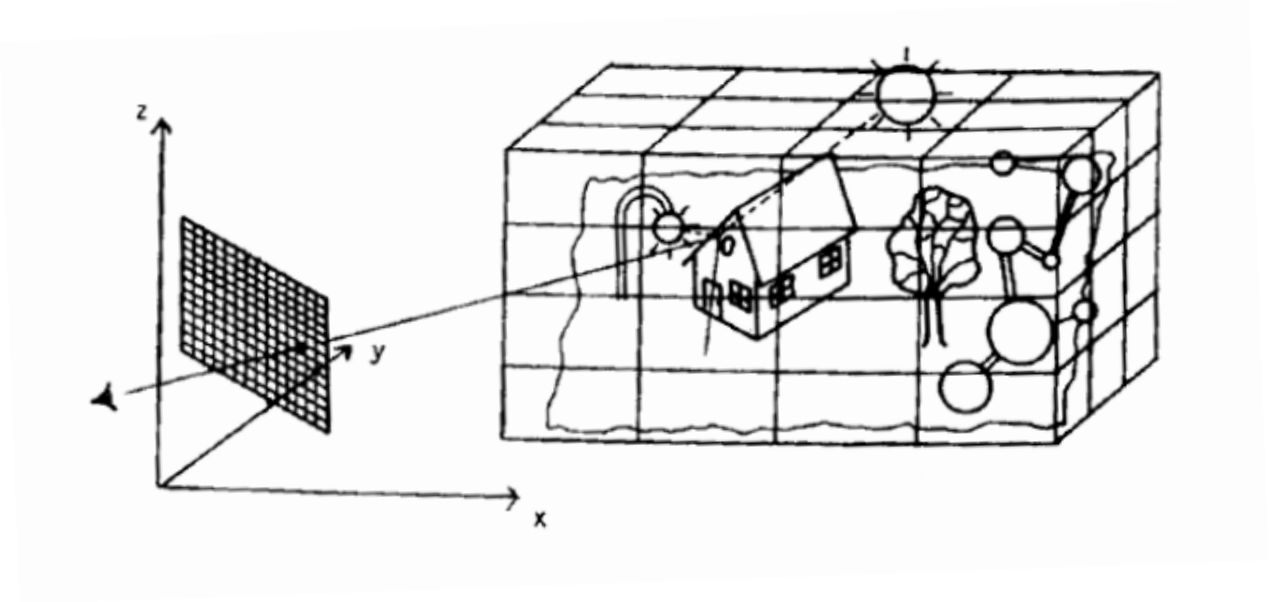


Рисунок 1.1 – Наглядна демонстрація трасування одного променю

Для наглядної демонстрації загальну роботу алгоритму продемонстровано на рисунку 1.1. Рисунок представляє собою 3D сцену заключену в сітку клітин, джерело світла та камеру сцену, що відображено сіткою пікселів екрану на який необхідно спроектувати всю сцену. Ціль алгоритму полягає в проектуванні всіх можливих променів з джерела світла на об’єкти в сцені та потім відбити всі промені від поверхонь в сцені в камеру, що знаходиться під певним кутом до сцени. На рисунку видно симуляцію одного єдиного променю, що прямує від джерела до даху та відбивається в камеру, зафарбовуючи певний піксель екрану в колір, який алгоритм визначив базуючись на даних отриманих від матеріалу відбитої поверхні.

Для того, щоб оптимізувати роботу, ми простежуємо тільки ті промені, які гарантовано потрапляють у камеру користувача і досягають очей. Спочатку здається, що неможливо заздалегідь знати, які промені досягають камери. Врешті-решт, промінь може багато разів відбиватись по кімнаті, перш ніж досягти очей. Однак, якщо ми розглянемо цю проблему з другого боку, то бачимо, що вона має дуже просте рішення. Замість того, щоб простежити промені, що починаються від джерела світла, ми простежуємо їх назад, починаючи з очей.

Розглянемо будь-який піксель екрану, колір якого ми намагаємося визначити. Його колір задається кольором світлового променю, який проходить через цю точку на екрані і досягає очей. Ми можемо так само слідувати за променями назад, починаючи від очей і проходячи через точку на його виході на сцену. Два промені будуть однаковими, за винятком їх напрямку: якщо первинний промінь вийшов безпосередньо з джерела світла, то зворотний промінь буде йти безпосередньо від очей до джерела світла; якщо оригінал спершу відскочив від столу, зворотний промінь також відскакує від столу. Таким чином, зворотний метод робить те ж саме, що і оригінальний метод, за винятком того, що він не витрачає жодних зусиль на промені, які ніколи не доходять до ока.

Так само, як і в методі який рухає світловий промінь від джерела світла до ока, метод може використати дуже велике число відскоків, перш ніж промінь коли-небудь потрапить в око, так і в зворотному методі перш ніж кожен промінь попаде назад на джерело алгоритму може знадобитися дуже велика кількість відскоків, що не є добрим показником ресурсо-затратності алгоритму. Оскільки нам необхідно встановити певну межу на кількість відскоків, щоб слідувати променям, ми робимо наступне наближення: кожен раз, коли промінь потрапляє на об'єкт, ми слідуємо за одним новим промені від точки перетину прямо до джерела світла.

Також, що перед тим як описувати дію алгоритму та його специфікацію вважаю за необхідне ввести базову термінологію. При створенні будь-якої комп'ютерної графіки необхідно мати список об'єктів, які програма відобразить. Ці об'єкти є частиною сцени або світу, як показано на рисунку 1.2, тому, коли використовується термін "дивлячись на світ", ми посилаємося на трасувальник променів та те як він відображує об’єкти з певної позиції перегляду. У графіці ця позиція перегляду називається оком або камерою. Слідуючи аналогії з камерою, так само, як камера потребує фільму, на який проектується і записується сцена, у графіці ми маємо вікно перегляду, на якому ми малюємо сцену. Відмінність полягає в тому, що в камерах плівка розміщена за діафрагмою або фокусною точкою, у графіці вікно перегляду знаходиться перед фокусною точкою. Отже, колір кожної точки на реальній плівці обумовлений світловим промінням (насправді, групою променів), який проходить через діафрагму і потрапляє на плівку, тоді як у комп'ютерній графіці кожен піксель кінцевого зображення задається світловим промінням що потрапляє у вікно перегляду на його шляху до очей. Результати, однак, однакові.

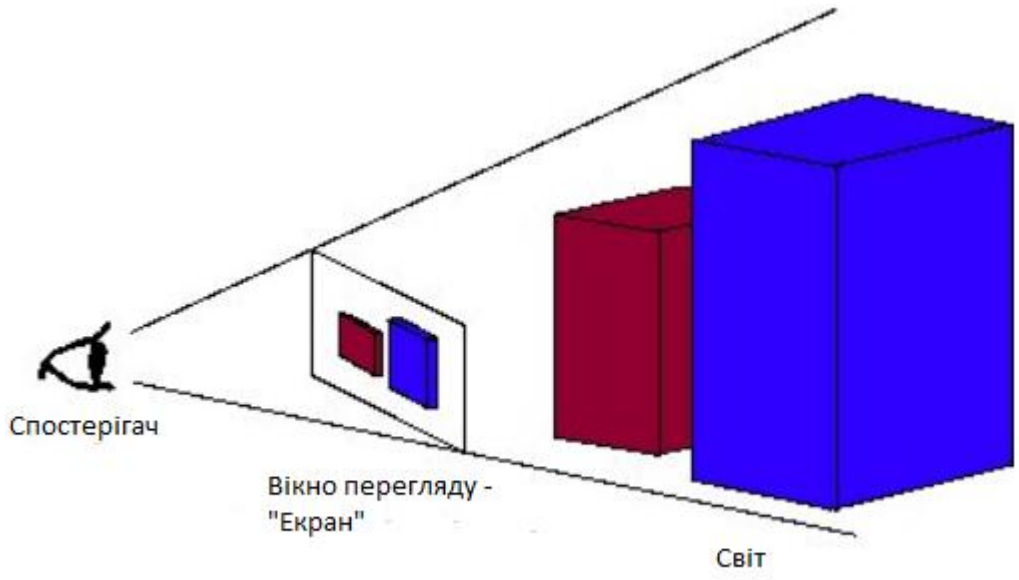


Рисунок 1.2 – Загальне поняття світу в комп’ютерній графіці

1.1.1 Опис процесу діяльності

Основними процесами діяльності є:

а) Рендерінг об’єктів що входять в кут огляду гравця – обробка всієї ігрової сцени та всіх фізичних об’єктів, що її наповнюють бажано 20 разів за секунду чи більше для плавності передачі зображення.

б) Зчитування вхідних даних гравця – переміщення камери, що підконтрольна гравцю згідно з його вводом.

в) Тіньова карта або мапування тіней – створення тіней для об’єктів шляхом перевірки видимості пікселя з джерела світла, за допомогою порівняння пікселя з z-буфером або глибинним зображенням куту обзору джерела світла, що зберігається у вигляді текстури.

г) Відбиття світлових променів від поверхностей – трасування або ж кастування променю з камери(куту огляду гравця) на всі можливі поверхності, що бачить користувач. Здійснюється за допомогою рекурсивного отримання кольору пікселя на який вказує змодельований промінь.

1.1.2 Опис функціональної моделі

Специфікацію функціональної поведінки системи представимо у вигляді діаграми варіантів використання. На рисунку 1.1 наведено всі функції системи та описано акторів, які будуть їх використовувати.

Рисунок 1.3 – USE-case діаграма

Безпосередньо із системою будуть взаємодіяти два один актор:

- гравець – це особа, яка може працювати із основними функціями системи та впливати на стан гри;

Відповідно до визначених варіантів використання виявлено функціональні вимоги та встановлено їх пріоритетність. Результат наведено в таблиці 1.1.

Таблиця 1.1 – Функціональні вимоги програми

Функціональні вимоги

Варіант використання	Функціональна вимога	Пріоритет
Керування камерою	1. Система надає можливість гравцю керувати камерою, її положенням та кутом обзору.	Високий
Переміщення персонажу по рівню	1. Система надає можливість гравцю рухати персонажа разом з камерою	Високий
Кадрова частота	2. Система впродовж роботи підтримує стабільну кадрову частоту, вище за 30	Високий

1.2 Огляд наявних аналогів

Перед початком роботи над дипломним проектом було здійснено пошук застосунків зі схожою функціональністю.

Оскільки семантика створеної програми дуже специфічна був проведений можливий аналіз вибору інших ігрових движків[3] а не самої програми. У таблиці 1.2 наведено назви наявних аналогів та режими доступів до них.

Таблиця 1.2 – Наявні аналоги

Назва	Режим доступу
Unity	https://unity.com
Unreal Engine 4	https://www.unrealengine.com
CryEngine 5	https://www.cryengine.com/
Godot Engine	https://godotengine.org/

Проаналізуємо функції, які виконують зазначені вище програмні продукти. Дані по функціям наведено в таблиці 1.3.

Таблиця 1.3 – Огляд функцій наявних аналогів

Аналоги / Функції	Unity	Godot	CryEngine 5
М’які тіні	-	-	-
Трасування відображень	+	-	+
Повна інтеграція трасування променів	+	-	-
Повний реліз алгоритму трасування	-	-	-

Із таблиці 1.3 видно, що ті аналоги, які представлені нині на ринку, не охоплюють увесь спектр функцій, необхідних для трасування променів, чим й обумовлений вибір Unreal Engine 4 в якому наявні всі зазначені – критичні для алгоритму функції.

1.3 Постановка задачі

1.3.1 Призначення розробки

Запропонований програмний продукт призначений для максимально чіткої візуалізації роботи алгоритму трасування променів в реальному часі. Під трасуванням променів в реальному часі мається на увазі постійний, рекурсивний підрахунок всіх променів світла(трасування) та їх проходження та відбиття від всіх можливих поверхонь під час того як гравець має можливість впливати на рівень, переміщати персонажа, тощо.

1.3.2 Цілі та задачі розробки

Мета роботи – створення ігрового рівня, системи освітлення та відбиття відображень в дзеркальних поверхнях в реальному часі. Як другорядну мету ставимо перед собою ціль оптимізувати роботу алгоритму трасування променів в реальному часі, адже деяка алгоритму уже запропонована та частково імпелентована в ігровий движок Unreal Engine 4, за допомогою якого і ведеться значна частина роботи. Також слід зазначити що за мету роботи ставимо поліпшення і, можливо, в залежності від досягнутих результатів, реальна імплементація алгоритму трасування в спосіб обробки освітлення в ігоровому движці для досягнення реалістичного відображення створеного рівня на сучасних комп'ютерах, що дозволяють трасувати промені на своїх графічних процессорах.

Для реалізації поставленої мети необхідно розв'язати наступні задачі.

1) Створення реалістичної системи освітлення для застосування техніки м'яких тіней та відображення тіней в дзеркалах;

2) Створення реалістичної системи освітлення для трасування усіх вихідних променів із кожного джерела світла;

3) Створення реалістичної системи відбиття зображень від поверхонь;

4) Оптимізувати вбудований алгоритм трасування променів та його роботу та продуктивність.

Висновок до розділу

У даному розділі було розглянуто предметне середовище. Дипломний проект присвячений розробці ігрового рівня для демонстрації роботи системи трасування променів.

При описі предметного середовища було визначено та описано процеси діяльності, що представлені у вигляді двох незалежних процесів, для кожного з яких наведено структурні схеми. Також було створено функціональну модель системи, а саме: побудовано USE-case діаграму, описано акторів системи та визначено функціональні вимоги з пріоритетами виконання відповідно до варіантів використання.

Наступним етапом роботи над дипломним проектом став пошук аналогів запропонованого движку з подібним функціоналом. Було порівняно функції, які виконують знайдені програмні продукти. На даний момент було виявлено декілька систем зі схожим функціоналом, але вони не охоплюють увесь спектр функцій, які необхідні для реалізації даного дипломного проекту. Жоден з аналогів продуктів не підтримує використання системи м’яких тіней та жоден з наявних на ринку аналогів не має стабільного релізу системи трасування.

Також у одному з підрозділів було сформовано постановку задачі, визначено призначення, мету та задачі розробки.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Вхідні дані

Беручи до уваги специфіку розробленого програмного продукту вхідні мають дещо специфічний характер. Оскільки основне призначення алгоритму є трасування й відбиття променів від різних поверхонь єдиний масив вхідних даних та основна вибірка даних з якою здійснюються всі підрахунки це поверхні усіх об’єктів сцени: саме від об’єктів трасуються усі промені в сцені та від них залежить параметри відбитих променів. Оскільки, як зазначено в розділі 1, всі об’єкти симульованого світу складаються з найменших можливих частинок, які підраховує комп’ютер, **полігонів**, то саме них й беремо за вхідні дані. Параметри полігонів, їх розміщення, властивості, тощо в першу чергу впливають на алгоритм трасування та є найважливішими даними з якими працює алгоритм. Як наглядний приклад трасування променю та його відбиття – зображена на рисунку 2.1 модель зайця, що складається з полігонів та промодельований промінь на один з полігонів з яких складається модель зайця.

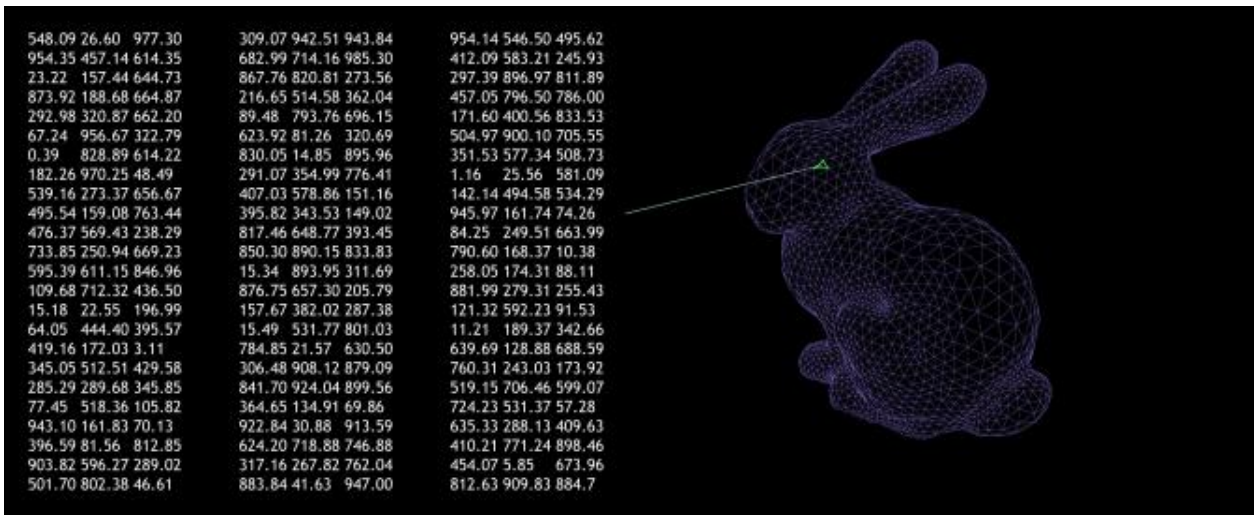


Рисунок 2.1 – Модель об’єкту з полігонів

На рисунку 2.2 представлено порівняння вигляду вихідного об’єкту та полігонів з яких складається об’єкт. Проте необхідно звернути увагу, що алгоритм трасування променів працює не тільки з полігонами, але й з **матеріалами**.

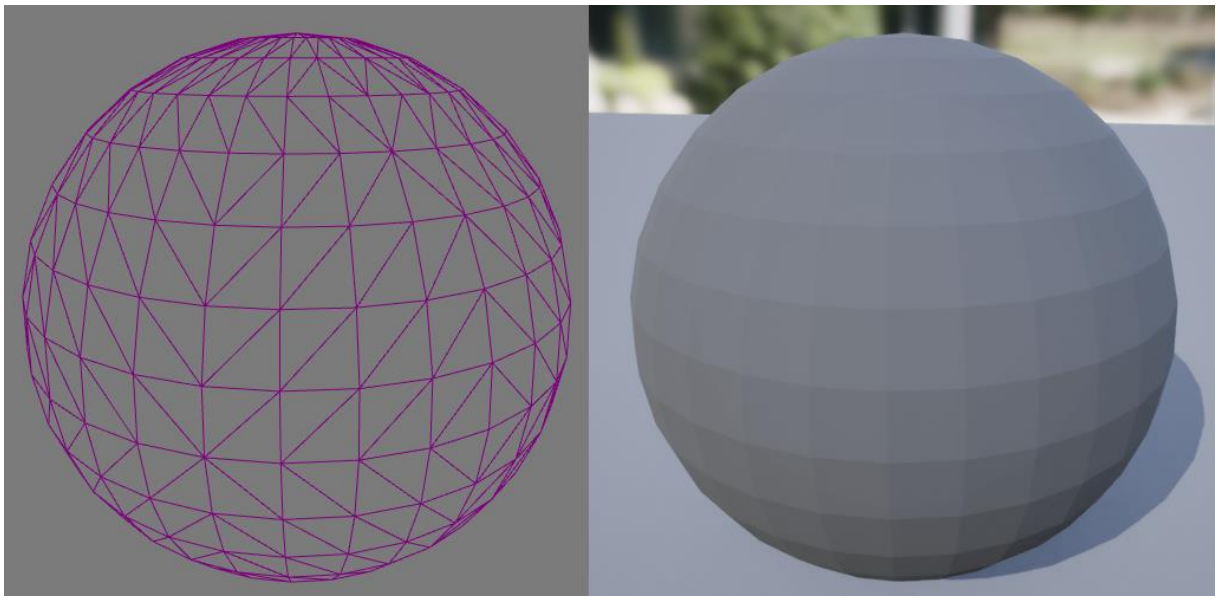


Рисунок 2.2 – Модель об’єкту з полігонів

Матеріали – параметри поверхонь які використовує трасувальник в той момент коли підраховує наскільки яскравий повинен вийти відбитий промінь. Якщо полігони з яких складаються об’єкти впливають лише на кут відбитого променю, то матеріал поверхні впливає на силу та колір променю. На рисунку 2.2 зображений об’єкт без матеріалу поверх нього – тобто без ніяких параметрів поверхні, які міг би використати алгоритм під час підрахунку. Причина чому м’яч, що зображений на рисунку справа відрізняється від лівого полягає в движку, який автоматично зафарбовує пусті елементи полігонів непрозорим, не відбиваючим світло кольором. Нижче на рисунку 2.3 зображений м’яч с параметрами дзеркальної поверхні яка відбиває всі надходженні промені.

Рисунок 2.3 – Фінальний вигляд моделі

Слід зазначити, що для придачі полігонам такого вигляду движок представляє певні параметри поверхні, які підконтрольні розробнику. Змінюючи параметри, та доповнюючи їх розробники ігор та представники індустрії кінематографу можуть добитися майже реального відображення об’єктів всередині сцени. Далі на рисунку 2.4 зображений можливий список параметрів матеріалу.

Рисунок 2.4 – Параметри матеріалу

Не менш важливим параметром і сетом даних є масив променів в кінцевій сцені – неможливо прорахувати шлях променів без самих променів. В самому ігровому движці всі джерела світла представлені не об’єктами сцени, що не видимі камері. Джерела світла випускають промені в залежності від заданих розробником параметрів, дивлячись на рисунок 2.5 можна побачити деякі параметри та представлення джерела як в движці так і в кінцевій сцені.

Рисунок 2.5 – Представлення джерел світла

2.2 Вихідні дані

Вихідними даними є відпрацьована сцена з об’єктами в їх кінцевому вигляді. За вихідні дані також можна вважати опрацьоване освітлення та тіні в сцені, проте слід мати на увазі, що оскільки розрахунки ведуться в реальному часі, результати роботи алгоритму не статичні і розраховуються постійно, тому важко навести конкретний результат. Загалом можна вважати за вихідні дані параметри освітлення, відбиття та тіней адже саме ці параметри є роботою алгоритму трасування променів.

Як вибірка кінцевих результатів будуть подані найбільш яскраві приклади роботи алгоритму та їх порівняння з класичними підходами програмування ігрових сцен.

Відбиття променів від поверхонь:

Рисунок 2.6 – Порівняння якості відбиття між трасувальним алгоритмом та класичним підходом до дзеркал

Рисунок 2.7 – Порівняння якості відбиття між трасувальним алгоритмом з параметр відбиття 1 та 3

На представлених рисунках відображені результати роботи алгоритму у сенсі відображення зображень у дзеркальних поверхнях. На рисунку 2.6 зображено порівняння однакових дзеркальних поверхонь, проте з різним підходом до обробки відображень. В верхній частині рисунку показано роботу класичних відображень, де кожний кадр робиться два підрахунки: перший створює зображення всього перед дзеркалом з позиції за дзеркалом, а другий використовує згенеровану карту відображень в першому кроці, для створення ілюзії відображення для гравця і положення його камери. В нижній же частині рисунку продемонстрована робота алгоритму трасування променів, що відбиває всі світлові промені від поверхні в камеру, створюючи найближче до реальності відображення.

На рисунку 2.7 представлені також два зображення для порівняння, проте в обох випадках відображення опрацьовує алгоритм трасування променів. В верхньому рисунку алгоритм працює з параметром відбиття 1, що значить, що алгоритм протрасує промені від джерела світла, до відображення лише один раз та всі відбиття від поверхонь проводяться лише один раз. На нижньому зображені алгоритм відбиває промені максимум 3 рази, що дуже сильно впливає на якість та реальність зображення в представленому випадку – коли дві дзеркальні поверхні знаходяться навпроти.

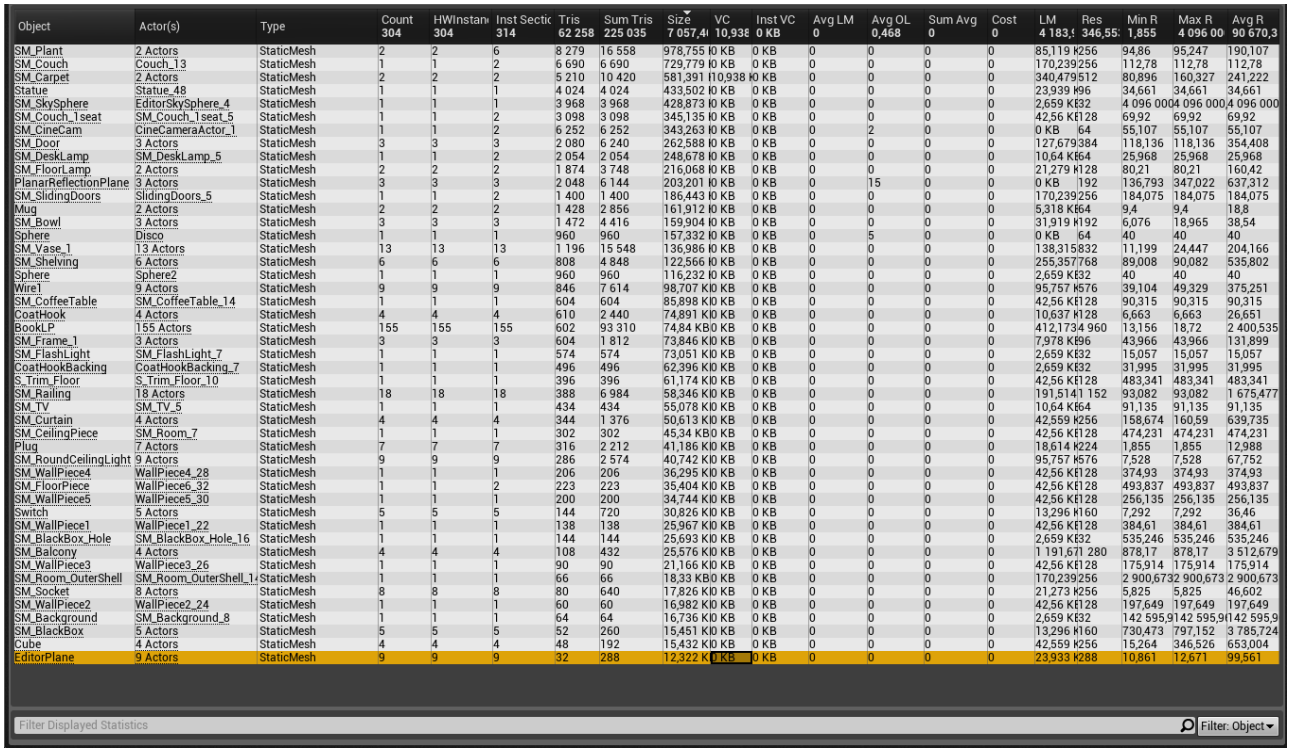
Але за якість трасувальник бере велику ціну в сенсі продуктивності роботи комп'ютеру на якому здійснюється підрахунок – оскільки алгоритм робить в 3 рази більше підрахунків обладнання на якому відображена сцена в реальному часі зачасту буде показувати занижену частоту кадрів. Спостерігачу ж, що присутній в ігровій сцені та рухає камеру буде набагато виразніша недостача кадрів аніж якість відображення в дзеркалі. Тому наразі, якщо користуватися трасувальником променів буде бажано уникати ситуацій дизайну ігрових та кінематографічних рівней де два дзеркала будуть розміщені навпроти і необхідно використовувати великий параметр відбиття світлових променів. Наразі з технологіями обробки відображень програмісти намагаються повністю уникнути відображень, або ж відвести увагу гравця від них для досягнення реалістичного дизайну.

Рисунок 2.8 – Порівняння якості та реалістичності тіні між трасувальним алгоритмом для тіней та звичайним мапуванням

Як і зображено на рисунку 2.8 трасування променів також дуже сильно впливає на якість відображення тіней в скомпонованій сцені, адже саме трасування світлових променів дозволяє точно дізнатись які промені потраплять на поверхність об’єкту, а які продовжать свій шлях, щоб потім створити тінь цього ж об’єкту. Знову, підхід повністю відрізняється від класичного мапування тіней й відповідно дозволяє досягати набагато кращої якості фінального зображення. Хоча як і в випадку з трасуванням відображень даний метод потребує набагато більших витрат.

2.3 Структура масивів інформації

Дані, що використовує алгоритм можна відобразити в виді масиву інформації обробки полігонів, і зробити це за допомогою движка, як показано на рисунку 2.9 – також на рисунку присутня інша важлива інформація для розробника, такі дані як використана пам’ять, кількість використаних полігонів, частота повторювання об’єкту в сцені, тощо.

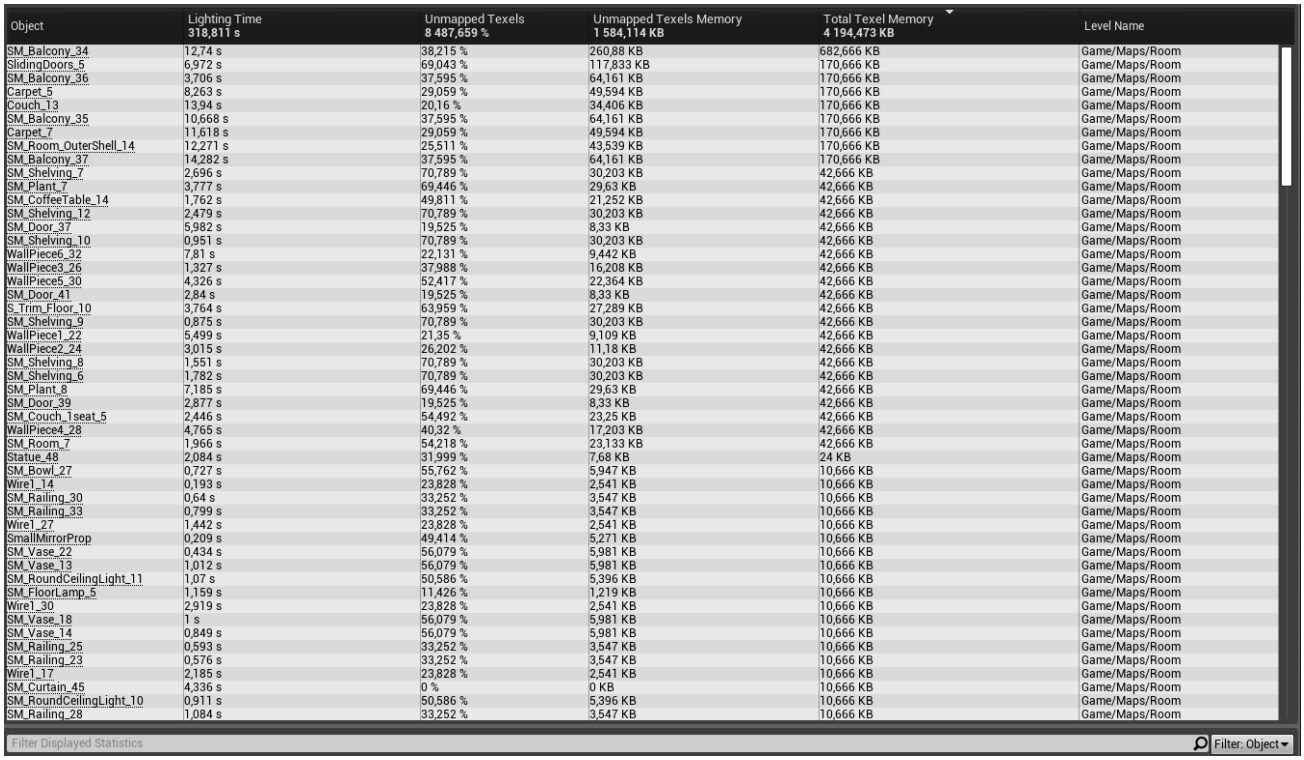


Object	Actor(s)	Type	Count 304	HWInstan 304	Inst Sectic 314	Tris 62 258	Sum Tris 225 035	Size 7 057,4	VC 10,938	Inst VC 0 KB	Avg LM 0	Avg OL 0,468	Sum Avg 0	Cost 0	LM 4 183,5	Res 346,55	Min R 1,855	Max R 4 096 00	Avg R 90 670,3
SM_Plant	2 Actors	StaticMesh	2	2	6	8 279	16 558	978,755 K	0 KB	0 KB	0	0	0	0	85,119 K	256	94,86	95,247	190,107
SM_Couch	Couch_13	StaticMesh	1	1	2	6 690	6 690	729,779 K	0 KB	0 KB	0	0	0	0	170,239	256	112,78	112,78	112,78
SM_Carpet	2 Actors	StaticMesh	2	2	2	5 210	10 420	581,391 K	10,938 K	0 KB	0	0	0	0	340,479	512	80,896	160,327	241,222
Statue	Statue_48	StaticMesh	1	1	1	4 024	4 024	433,502 K	0 KB	0 KB	0	0	0	0	23,939 K	96	34,661	34,661	34,661
SM_SkySphere	EditorSkySphere_4	StaticMesh	1	1	1	3 968	3 968	428,873 K	0 KB	0 KB	0	0	0	0	2,659 K	32	4 096 000	4 096 000	4 096 000
SM_Couch_1seat	SM_Couch_1seat_5	StaticMesh	1	1	2	3 098	3 098	345,135 K	0 KB	0 KB	0	0	0	0	42,56 K	128	69,92	69,92	69,92
SM_CineCam	CineCameraActor_1	StaticMesh	1	1	2	6 252	6 252	343,263 K	0 KB	0 KB	0	2	0	0	0 KB	64	55,107	55,107	55,107
SM_Door	3 Actors	StaticMesh	3	3	3	2 080	6 240	262,588 K	0 KB	0 KB	0	0	0	0	127,679	384	118,136	118,136	354,408
SM_DeskLamp	SM_DeskLamp_5	StaticMesh	1	1	2	2 054	2 054	248,678 K	0 KB	0 KB	0	0	0	0	10,64 K	64	25,968	25,968	25,968
SM_FloorLamp	2 Actors	StaticMesh	2	2	2	1 874	3 748	216,068 K	0 KB	0 KB	0	0	0	0	21,279 K	128	80,21	80,21	160,42
PlanarReflectionPlane	3 Actors	StaticMesh	3	3	3	2 048	6 144	203,201 K	0 KB	0 KB	0	15	0	0	0 KB	192	136,793	347,022	637,312
SM_SlidingDoors	SlidingDoors_5	StaticMesh	1	1	2	1 400	1 400	186,443 K	0 KB	0 KB	0	0	0	0	170,239	256	184,075	184,075	184,075
Mug	2 Actors	StaticMesh	2	2	2	1 428	2 856	161,912 K	0 KB	0 KB	0	0	0	0	5,318 K	64	9,4	9,4	18,8
SM_Bowl	3 Actors	StaticMesh	3	3	3	1 472	4 416	159,904 K	0 KB	0 KB	0	0	0	0	31,919 K	192	6,076	18,965	38,54
Sphere	Disco	StaticMesh	1	1	1	960	960	157,332 K	0 KB	0 KB	0	5	0	0	0 KB	64	40	40	40
SM_Vase_1	13 Actors	StaticMesh	13	13	13	1 196	15 548	136,986 K	0 KB	0 KB	0	0	0	0	138,315	832	11,199	24,447	204,166
SM_Shelving	6 Actors	StaticMesh	6	6	6	808	4 848	122,566 K	0 KB	0 KB	0	0	0	0	255,357	768	89,008	90,082	535,802
Sphere	Sphere2	StaticMesh	1	1	1	960	960	116,232 K	0 KB	0 KB	0	0	0	0	2,659 K	32	40	40	40
Wire1	9 Actors	StaticMesh	9	9	9	846	7 614	98,707 KB	0 KB	0 KB	0	0	0	0	95,757 K	576	39,104	49,329	375,251
SM_CoffeeTable	SM_CoffeeTable_14	StaticMesh	1	1	1	604	604	85,898 KB	0 KB	0 KB	0	0	0	0	42,56 K	128	90,315	90,315	90,315
CoatHook	4 Actors	StaticMesh	4	4	4	610	2 440	74,891 KB	0 KB	0 KB	0	0	0	0	10,637 K	128	6,663	6,663	26,651
BookLP	155 Actors	StaticMesh	155	155	155	602	93 310	74,84 KB	0 KB	0 KB	0	0	0	0	412,173	4 960	13,156	18,72	2 400,535
SM_Frame_1	3 Actors	StaticMesh	3	3	3	604	1 812	73,846 KB	0 KB	0 KB	0	0	0	0	7,978 K	96	43,966	43,966	131,899
SM_FlashLight	SM_FlashLight_7	StaticMesh	1	1	1	574	574	73,051 KB	0 KB	0 KB	0	0	0	0	2,659 K	32	15,057	15,057	15,057
CoatHookBacking	CoatHookBacking_7	StaticMesh	1	1	1	496	496	62,396 KB	0 KB	0 KB	0	0	0	0	2,659 K	32	31,995	31,995	31,995
S_Trim_Floor	S_Trim_Floor_10	StaticMesh	1	1	1	396	396	61,174 KB	0 KB	0 KB	0	0	0	0	42,56 K	128	483,341	483,341	483,341
SM_Railing	18 Actors	StaticMesh	18	18	18	388	6 984	58,346 KB	0 KB	0 KB	0	0	0	0	191,514	1 152	93,082	93,082	1 675,477
SM_TV	SM_TV_5	StaticMesh	1	1	1	434	434	55,078 KB	0 KB	0 KB	0	0	0	0	10,64 K	64	91,135	91,135	91,135
SM_Curtain	4 Actors	StaticMesh	4	4	4	344	1 376	50,613 KB	0 KB	0 KB	0	0	0	0	42,559 K	256	158,674	160,59	639,735
SM_CeilingPiece	SM_Room_7	StaticMesh	1	1	1	302	302	45,34 KB	0 KB	0 KB	0	0	0	0	42,56 K	128	474,231	474,231	474,231
Plug	7 Actors	StaticMesh	7	7	7	316	2 212	41,186 KB	0 KB	0 KB	0	0	0	0	18,614 K	224	1,855	1,855	12,988
SM_RoundCeilingLight	9 Actors	StaticMesh	9	9	9	286	2 574	40,742 KB	0 KB	0 KB	0	0	0	0	95,757 K	576	7,528	7,528	67,752
SM_WallPiece4	WallPiece4_28	StaticMesh	1	1	1	206	206	36,295 KB	0 KB	0 KB	0	0	0	0	42,56 K	128	374,93	374,93	374,93
SM_FloorPiece	WallPiece6_32	StaticMesh	1	1	2	223	223	35,404 KB	0 KB	0 KB	0	0	0	0	42,56 K	128	493,837	493,837	493,837
SM_WallPiece5	WallPiece5_30	StaticMesh	1	1	1	200	200	34,744 KB	0 KB	0 KB	0	0	0	0	42,56 K	128	256,135	256,135	256,135
Switch	5 Actors	StaticMesh	5	5	5	144	720	30,826 KB	0 KB	0 KB	0	0	0	0	13,296 K	160	7,292	7,292	36,46
SM_WallPiece1	WallPiece1_22	StaticMesh	1	1	1	138	138	25,967 KB	0 KB	0 KB	0	0	0	0	42,56 K	128	384,61	384,61	384,61
SM_BlackBox_Hole	SM_BlackBox_Hole_16	StaticMesh	1	1	1	144	144	25,693 KB	0 KB	0 KB	0	0	0	0	2,659 K	32	535,246	535,246	535,246
SM_Balcony	4 Actors	StaticMesh	4	4	4	108	432	25,576 KB	0 KB	0 KB	0	0	0	0	1 191,67	1 280	878,17	878,17	3 512,679
SM_WallPiece3	WallPiece3_26	StaticMesh	1	1	1	90	90	21,166 KB	0 KB	0 KB	0	0	0	0	42,56 K	128	175,914	175,914	175,914
SM_Room_OuterShell	SM_Room_OuterShell_14	StaticMesh	1	1	1	66	66	18,33 KB	0 KB	0 KB	0	0	0	0	170,239	256	2 900,673	2 900,673	2 900,673
SM_Socket	8 Actors	StaticMesh	8	8	8	80	640	17,826 KB	0 KB	0 KB	0	0	0	0	21,273 K	256	5,825	5,825	46,602
SM_WallPiece2	WallPiece2_24	StaticMesh	1	1	1	60	60	16,982 KB	0 KB	0 KB	0	0	0	0	42,56 K	128	197,649	197,649	197,649
SM_Background	SM_Background_8	StaticMesh	1	1	1	64	64	16,736 KB	0 KB	0 KB	0	0	0	0	2,659 K	32	142 595,9	142 595,9	142 595,9
SM_BlackBox	5 Actors	StaticMesh	5	5	5	52	260	15,451 KB	0 KB	0 KB	0	0	0	0	13,296 K	160	730,473	797,152	3 785,724
Cube	4 Actors	StaticMesh	4	4	4	48	192	15,432 KB	0 KB	0 KB	0	0	0	0	42,559 K	256	15,264	346,526	653,004
EditorPlane	9 Actors	StaticMesh	9	9	9	32	288	12,322 K	1 KB	0 KB	0	0	0	0	23,933 K	288	10,861	12,671	99,561

Filter Displayed Statistics — Filter: Object

Рисунку 2.9 – Дані про об’єкти сцени

Згідно з рисунком 2.10 також наявна вся інформація про освітлення всіх об’єктів сцени: затрачений час на підрахунок, використана пам’ять графічного процесору, відсоток освітленості об’єктів та інше.



Object	Lighting Time 318,811 s	Unmapped Texels 8 487,659 %	Unmapped Texels Memory 1 584,114 KB	Total Texel Memory 4 194,473 KB	Level Name
SM_Balcony_34	12,74 s	38,215 %	260,88 KB	682,666 KB	Game/Maps/Room
SlidingDoors_5	6,972 s	69,043 %	117,833 KB	170,666 KB	Game/Maps/Room
SM_Balcony_36	3,706 s	37,595 %	64,161 KB	170,666 KB	Game/Maps/Room
Carpet_5	8,263 s	29,059 %	49,594 KB	170,666 KB	Game/Maps/Room
Couch_13	13,94 s	20,16 %	34,406 KB	170,666 KB	Game/Maps/Room
SM_Balcony_35	10,668 s	37,595 %	64,161 KB	170,666 KB	Game/Maps/Room
Carpet_7	11,618 s	29,059 %	49,594 KB	170,666 KB	Game/Maps/Room
SM_Room_OuterShell_14	12,271 s	25,511 %	43,539 KB	170,666 KB	Game/Maps/Room
SM_Balcony_37	14,282 s	37,595 %	64,161 KB	170,666 KB	Game/Maps/Room
SM_Shelving_7	2,696 s	70,789 %	30,203 KB	42,666 KB	Game/Maps/Room
SM_Plant_7	3,777 s	69,446 %	29,63 KB	42,666 KB	Game/Maps/Room
SM_CoffeeTable_14	1,762 s	49,811 %	21,252 KB	42,666 KB	Game/Maps/Room
SM_Shelving_12	2,479 s	70,789 %	30,203 KB	42,666 KB	Game/Maps/Room
SM_Door_37	5,982 s	19,525 %	8,33 KB	42,666 KB	Game/Maps/Room
SM_Shelving_10	0,951 s	70,789 %	30,203 KB	42,666 KB	Game/Maps/Room
WallPiece6_32	7,81 s	22,131 %	9,442 KB	42,666 KB	Game/Maps/Room
WallPiece3_26	1,327 s	37,988 %	16,208 KB	42,666 KB	Game/Maps/Room
WallPiece5_30	4,326 s	52,417 %	22,364 KB	42,666 KB	Game/Maps/Room
SM_Door_41	2,84 s	19,525 %	8,33 KB	42,666 KB	Game/Maps/Room
S_Trim_Floor_10	3,764 s	63,959 %	27,289 KB	42,666 KB	Game/Maps/Room
SM_Shelving_9	0,875 s	70,789 %	30,203 KB	42,666 KB	Game/Maps/Room
WallPiece1_22	5,499 s	21,35 %	9,109 KB	42,666 KB	Game/Maps/Room
WallPiece2_24	3,015 s	26,202 %	11,18 KB	42,666 KB	Game/Maps/Room
SM_Shelving_8	1,551 s	70,789 %	30,203 KB	42,666 KB	Game/Maps/Room
SM_Shelving_6	1,782 s	70,789 %	30,203 KB	42,666 KB	Game/Maps/Room
SM_Plant_8	7,185 s	69,446 %	29,63 KB	42,666 KB	Game/Maps/Room
SM_Door_39	2,877 s	19,525 %	8,33 KB	42,666 KB	Game/Maps/Room
SM_Couch_1seat_5	2,446 s	54,492 %	23,25 KB	42,666 KB	Game/Maps/Room
WallPiece4_28	4,765 s	40,32 %	17,203 KB	42,666 KB	Game/Maps/Room
SM_Room_7	1,966 s	54,218 %	23,133 KB	42,666 KB	Game/Maps/Room
Statue_48	2,084 s	31,999 %	7,68 KB	24 KB	Game/Maps/Room
SM_Bowl_27	0,727 s	55,762 %	5,947 KB	10,666 KB	Game/Maps/Room
Wire1_14	0,193 s	23,828 %	2,541 KB	10,666 KB	Game/Maps/Room
SM_Railing_30	0,64 s	33,252 %	3,547 KB	10,666 KB	Game/Maps/Room
SM_Railing_33	0,799 s	33,252 %	3,547 KB	10,666 KB	Game/Maps/Room
Wire1_27	1,442 s	23,828 %	2,541 KB	10,666 KB	Game/Maps/Room
SmallMirrorProp	0,209 s	49,414 %	5,271 KB	10,666 KB	Game/Maps/Room
SM_Vase_22	0,434 s	56,079 %	5,981 KB	10,666 KB	Game/Maps/Room
SM_Vase_13	1,012 s	56,079 %	5,981 KB	10,666 KB	Game/Maps/Room
SM_RoundCeilingLight_11	1,07 s	50,586 %	5,396 KB	10,666 KB	Game/Maps/Room
SM_FloorLamp_5	1,159 s	11,426 %	1,219 KB	10,666 KB	Game/Maps/Room
Wire1_30	2,919 s	23,828 %	2,541 KB	10,666 KB	Game/Maps/Room
SM_Vase_18	1 s	56,079 %	5,981 KB	10,666 KB	Game/Maps/Room
SM_Vase_14	0,849 s	56,079 %	5,981 KB	10,666 KB	Game/Maps/Room
SM_Railing_25	0,593 s	33,252 %	3,547 KB	10,666 KB	Game/Maps/Room
SM_Railing_23	0,576 s	33,252 %	3,547 KB	10,666 KB	Game/Maps/Room
Wire1_17	2,185 s	23,828 %	2,541 KB	10,666 KB	Game/Maps/Room
SM_Curtain_45	4,336 s	0 %	0 KB	10,666 KB	Game/Maps/Room
SM_RoundCeilingLight_10	0,911 s	50,586 %	5,396 KB	10,666 KB	Game/Maps/Room
SM_Railing_28	1,084 s	33,252 %	3,547 KB	10,666 KB	Game/Maps/Room

Filter Displayed Statistics — Filter: Object

Рисунку 2.10 – Дані про освітлення сцени

Висновок до розділу

У розділі з описом інформаційного забезпечення було описано вхідні та вихідні дані комплексу задач – їх параметри та можливе застосування масивів даних алгоритмом. У підрозділі вхідних даних було також дане пояснення та розглянуто тип даних, що подаються на вхід та методи їх обробки алгоритмом. У підрозділі вихідних даних були надані приклади очікуваних даних та було виконано порівняння досягнутих зображень за допомогою алгоритму трасування, та класичного підходу.

3 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Змістовна постановка задачі

Однією головних з цілей сучасної роботи в комп'ютерній графіці є створення найбільш реалістичних і якомога детальніших зображень. Як деякі з найбільш важливих факторів - правильне моделювання світла та реалістична текстура поверхні додають до картинки багато інформації та придають фінальному зображення реалістичності, хоча й зображення є лише проекцією. Сьогодні більшість застосувань комп'ютерної графіки сильно залежать від максимально досяжного ступеня реалізму. Сфери застосування комп’ютерної графіки включають САПР(Система автоматизованого проектування і розрахунку), анімацію і візуалізацію, робототехніку, архітектуру, рекламу, реконструкцію для медичних установ та багато інших цілей. Що таке реалістичне зображення? Правильне моделювання різних ефектів освітлення є найважливішою частиною справжнього фотографічного реалізму. Для того, щоб пояснити це детально, представлено серію фотографій, рисунки 3.1 та 3.2. Досконалі реалістичні зображення можуть бути створені лише на потужних комп'ютерах за допомогою складного програмного забезпечення, та є результатом складних розрахунків.

Рисунок 3.1 – Зображення сцени без використання ефектів освітлення

Рисунок 3.2 – Зображення сцени з використанням ефектів освітлення

Це особливо стосується трасування променів. Як тривіальне пояснення трасування променів - для кожного пікселя зображення трасується промінь з точки зору(камери) на 3d-сцену для обчислення його першого зіткнення з об'єктом, див. рисунок 3.2. Якщо об'єкт відображає або заломлює промені, відповідний відбитий промінь розраховується згідно з законом відбиття або заломлення.

Ці нові промені розглядаються аналогічно. Для того, щоб розрахувати тіні, ми повинні знайти, чи перетинається промінь між об’єктом і джерелом світла якимось другорядним об’єктом. Якщо існує об'єкт, що перетинає цей промінь, то точка перетину лежить в тіні цього джерела світла, а його інтенсивність не враховується при розрахунках інтенсивності. Розрахунок інтенсивності здійснюється за формулою освітлення, що включає параметри матеріалу, прикріпленого до геометричних об'єктів сцени.

3.2 Математична постановка задачі

Трасування променів називається так тому, що алгоритм намагається імітувати шлях, який проходять світлові промені, коли вони відскакують від різних поверхонь стимульованого світу – сцени. Мета полягає в тому, щоб визначити колір кожного світлового променю, який потрапляє в кут огляду - камеру. Світловий в даному випадку можна розглядати як єдиний фотон.

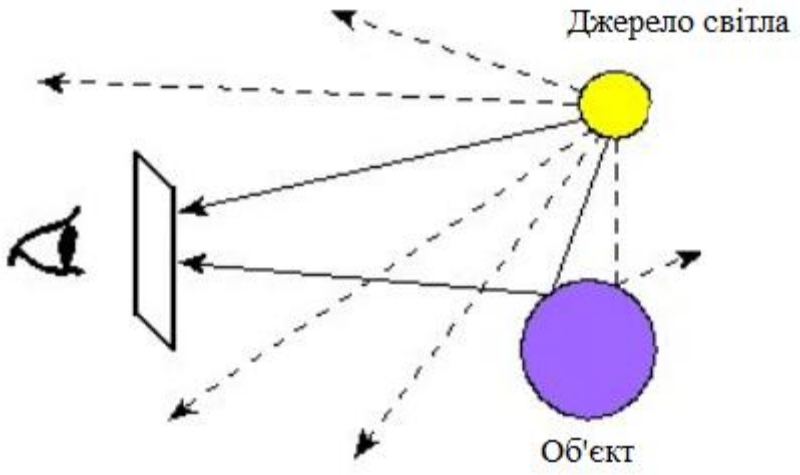


Рисунок 3.3 – Загальний концепт трасування променів

Проте постає проблема в продуктивності та оптимізації алгоритму. Оскільки алгоритм трасує усі промені доволі багато ресурсів апаратного забезпечення витрачаються в нікуди, адже далеко не всі промені потрапляють в камеру – отже ресурси затрачені на підрахунок шляху та відбиття всіх цих променів були витрачені даремно, для наглядної демонстрації променів, що не потрапляють в камеру див. рисунок 3.3.

Для того, щоб зберегти апаратні ресурси, ми простежуємо тільки ті промені, які гарантовано потрапляють у вікно перегляду і досягають очей. Спочатку здається, що неможливо заздалегідь знати, які промені досягають очей. Врешті-решт, будь-який промінь може багато разів відбиватись по кімнаті, перш ніж досягти очей. Однак, проблема має просте рішення якщо роздивитися її від протилежного. Замість того, щоб простежити промені, що починаються від джерела світла, алгоритм простежує їх з іншого боку, починаючи з очей. Два промені будуть однаковими, за винятком їхнього напрямку: якщо первинний промінь вийшов безпосередньо з джерела світла, то зворотний промінь буде йти виключно до джерела світла; якщо оригінал спершу відскочив від об’єкту, зворотний промінь також відскакує об’єкту.. Таким чином, зворотний метод робить те ж саме, що і оригінальний метод, за винятком того, що він не витрачає жодних зусиль на промені, які ніколи не доходять до ока.

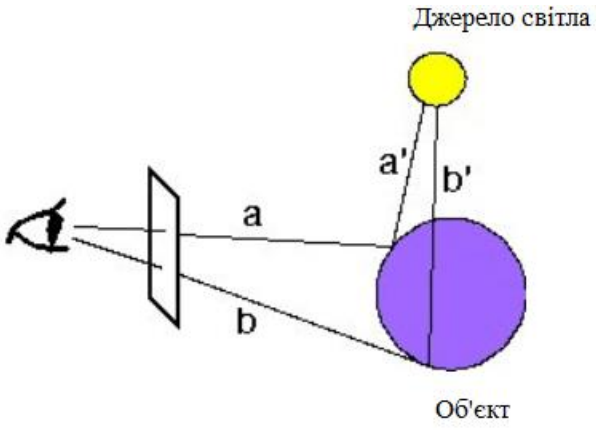


Рисунок 3.4 – Трасування променю від камери до джерела

Це, є одна з технік оптимізацій, яку було використано в даній роботі, для зменшення використання апаратних ресурсів. На малюнку ми бачимо два промені, **a** та **b**, які перетинаються з фіолетовою сферою. Щоб визначити колір **a**, ми слідуємо за новим променем **a'** безпосередньо до джерела світла. Як ви можете бачити, **b** буде затінено, оскільки промінь **b'** у напрямку до джерела світла блокується самою сферою. Промінь **a** також був би затінений, якщо інший об'єкт заблокував промінь **a'**. Так само, як і в методі світло-джерело-око, підрахунок відскоків промені може зайняти дуже багато часу, перш ніж промінь коли-небудь потрапить в око, у зворотному ж методі багато часу можуть зайняти відскоки до джерела, перш ніж промінь кожен потрапить на світло. Оскільки нам потрібно встановити певну межу на кількість відскоків, щоб слідувати променям, ми робимо наступне наближення: кожен раз, коли промінь потрапляє на об'єкт, ми слідуємо за одним новим промені від точки перетину безпосередньо до джерела світла, див. рисунок 3.4.

3.3 Обґрунтування методу розв’язання

Нижче буде розглянуто версію алгоритму трасування світлових променів, а саме підрахунок відбитого променю від заданої площини.

Розглянемо промінь як:

$$R_{\text{початковий}} = R_0 = [\, X_0 Y_0 Z_0] \quad (1.1)$$

$$R_{\text{напрям}} = R_d = [\, X_d Y_d Z_d], \quad (1.2)$$

$$\text{де } X_d^2 + Y_d^2 + Z_d^2 = 1,$$

Що представляє промінь у вигляді:

Набір точок на прямій $R(t) = R_0 + R_d * t$, де t >0. (1.3)

Напрямок променю не потрібно нормувати для цих розрахунків. Однак така нормалізація рекомендується, інакше t буде представляти відстань – довжину вектора напряму.

Розглянемо площину відбиття в виді [$A\ B\ C\ D$], та позначимо її P що предсатвляє площину в вигляді:

$$P = A * x + B * y + C * z + D = 0 \quad (1.4)$$

$$\text{Де } A^2 + B^2 + C^2 = 1.$$

Одиничний вектор нормалі позначимо як:

$$P_{\text{нормалі}} = P_n = [\, A\ B\ C\ ]$$

а відстань від початку координатної системи до площини P вийде просто D. Знак D означає на якій стороні координатної системи знаходиться початкова точка системи. Це неявне формулювання площини.

Дистанцію від початку променю до точки перетину з площиною P можна підрахувати замінивши частини рівняння (1.4) на формулу прямої (1.3), і отримати рівняння:

$$A * (X_0 + X_d * t) + B * (Y_0 + Y_d * t) + C * (Z_0 + Z_d * t) + D = 0 \quad (1.5)$$

Отже отримали рівняння для t:

$$t = \frac{-(A * X_0 + B * Y_0 + C * Z_0 + D)}{A * X_d + B * Y_d + C * Z_d} \quad (1.6)$$

В векторній формі рівняння буде виглядати так:

$$t = \frac{-(P_n * R_0 + D)}{P_n * R_0} \quad (1.7)$$

Щоб використати рівняння (1.6) максимально ефективно підраховуємо скалярний добуток:

$$v_d = P_n * R_d = A * X_d + B * Y_d + C * Z_d \quad (1.9)$$

Якщо ж $v_d = 0$, то промінь паралельний площині і перетину не відбувається. Звичайно, промінь може бути в одній площині, але цей випадок не має значення в практиці; Якщо промінь потрапляє на край полігону, на візуалізацію це ніяк не впливає. Також, якщо $v_d > 0$, нормаль площини направлена від променю. Якщо система моделювання використовує односторонні планарні об'єкти, тестування може закінчитися тут, так як площина не відбиває промені. Якщо промінь проходить ці випробування, обчислюємо другий добуток:

$$v_0 = -(P_n * R_0 + D) = -(A * X_0 + B * Y_0 + C * Z_0 + D). \quad (1.10)$$

Тепер підраховуємо відношення двох добутків:

$$t = v_0 / v_d \quad (1.11)$$

Якщо $t < 0$, то пряма, що задана променем перетинає площину за точкою початку променю – отже перетину не відбувається зовсім. Якщо ж перетин відбувається, підраховуємо точку перетину:

$$r_i = [x_i\ y_i\ z_i] = [X_0 + X_d * t \quad Y_0 + Y_d * t \quad Z_0 + Z_d * t] \tag{1.12}$$

Знак нормального вектора P_n може регулюватися залежно від напряму вектору R_d. Знак нормалі повинен бути зворотнім, щоб його направлення було протилежне джерелу проміню.

$$\text{Якщо } P_n * R_d < 0 \tag{1.13}$$

$$\text{То } r_n = P_n;$$

$$\text{Інакше } r_n = -P_n.$$

3.4 Опис методів розв'язання

Описаний вище алгоритм покриває лише відбиття від площини, що задана сценою і є найпростішим можливим варіантом відбиття променів, проте в даному під-розділі будуть дані швидкі описання інших поверхонь для яких необхідний свій алгоритм відбиття, хоча й не повністю.

Ось неповний список можливих алгоритмів для підрахунку точок перетину світлових променів з поверхнями:

- Перетин з сферою
- Зворотне трасування сфери
- Перетин з полігоном
- Опукле чотиристоронне зворотне трасування
- Трикутне зворотне трасування

Висновок до розділу

В даному розділі дипломного проекту було детально розглянуто підходи та рішення до поставлених задач. Також було порівняно ефективність деяких з алгоритмів трасування світлових променів, продемонстровано алгебраїчну імплементацію алгоритму, та базові підрахунки, що робить апаратне забезпечення під час стимулювання ігрової сцени. Було наглядно показано різницю в скомпонованих зображеннях при використанні алгоритму трасування та при відсутності візуальних ефектів. В одному з підрозділів було описано техніки оптимізації, що використовуються в сучасній комп'ютерній графіці й в даному дипломному проекті, та вплив описаних технік на ресурси апаратного забезпечення.

4 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Засоби розробки

При розробці програмного продукту були використані такі засоби та інструментарії розробки:

- платформа: **OS Windows 10 RS5 збірка 1809 – 64 біт**[4] – сімейство пропрієтарних операційних систем корпорації Microsoft, орієнтованих на застосування графічного інтерфейсу при управлінні; Вказана специфічна версія операційної системи, адже тільки вона наразі підтримує останні графічні драйвери компанії NVIDIA, які дозволяють використовувати вбудовані алгоритми трасування променів в ігровому движці.
- Ігровий движок: **Unreal Engine 4** [5] – движок, що розробляється і підтримується компанією Epic Games. Вибір зроблений в користь саме цього движка, адже він розповсюджується по моделі open-source та написаний на мові програмування C++, що дозволяє напряму впливати на пам’ять апаратного забезпечення та оптимізувати застосування;
- середовище розробки: **Visual Studio 2015**[6] –продукт фірми Майкрософт, який включають інтегроване середовище розробки програмного забезпечення та ряд інших інструментальних засобів. Використовувалось середовище виключно для редагування коду, адже воно інтегроване до ігрового движка**;**
- мови написання коду програми: **C++**;
- **C++**[7] – мова програмування високого рівня з підтримкою кількох парадигм програмування;

4.2 Вимоги до технічного забезпечення

4.2.1 Загальні вимоги

Так як розроблений програмний продукт використовує останні та найбільш новітні технології рендерінгу апаратні та програмні вимоги доволі високі. Апаратне забезпечення:

1) Процесор: Intel Core i7-4770K чи вище;
2) Оперативна пам’ять: 8 ГБ чи вище;
3) Відеокарта: GeForce GTX 1070 / GeForce RTX 2060 / Radeon RX Vega 56 чи вище.

Трасування променів відбувається насамперед на графічному процесорі, отже головною і найбільш вимогливою частиною апаратного забезпечення є саме він. Варто зазначити, що Nvidia почала підтримувати технологію трасування на графічних картах серії 10 тільки з квітня 2019 року, тому, наявність останніх драйверів будуть критичними для коректної роботи алгоритму.

Програмне забезпечення:

1) 64-розрядна Windows 10 RS5 з обновленням 1809;
2) DirectX – версія 12;
3) Nvidia graphics driver – версія 430.86 чи вище.

4.3 Архітектура програмного забезпечення

4.3.1 Діаграма класів

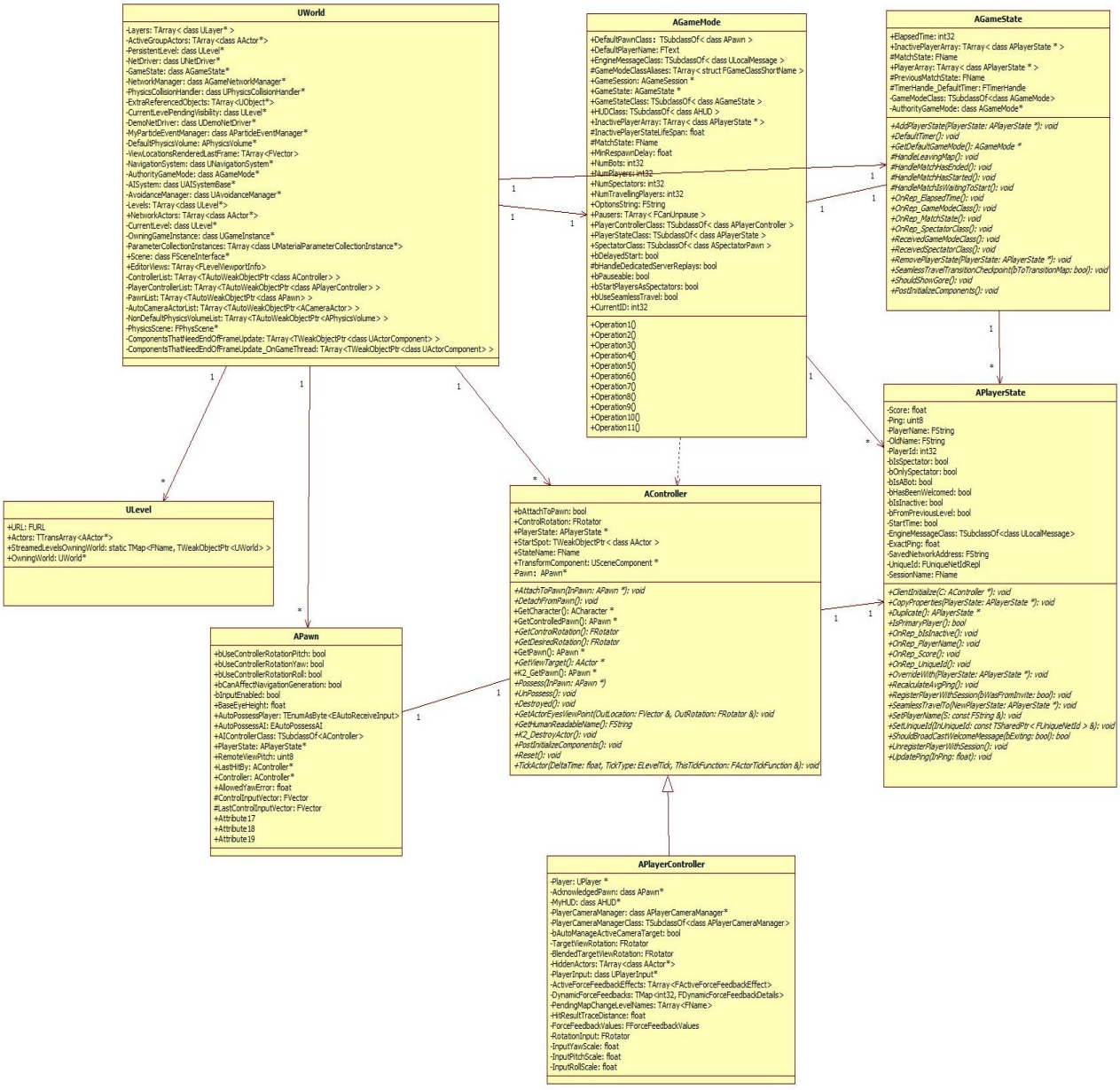


Рисунок 4.1 – Діаграма класів

Діаграма класів в даному випадку представляє собою згенеровану движком ієрархію класів, хоча й далеко не всіх. На рисунку 4.1 відображено основні класи движка й їх основні методи.

Насамперед створюючи діаграму класів було взято лише основні класи, що напряму впливають на стан ігрової сцени – класи **UWorld, ULevel**;
Контролери движка – класи **APawn, AController, APlayerController**;
Та стан гравця та підконтрольної йому камери – класи **AGameMode, AGameState, APlayerState**.

4.3.2 Діаграма послідовності

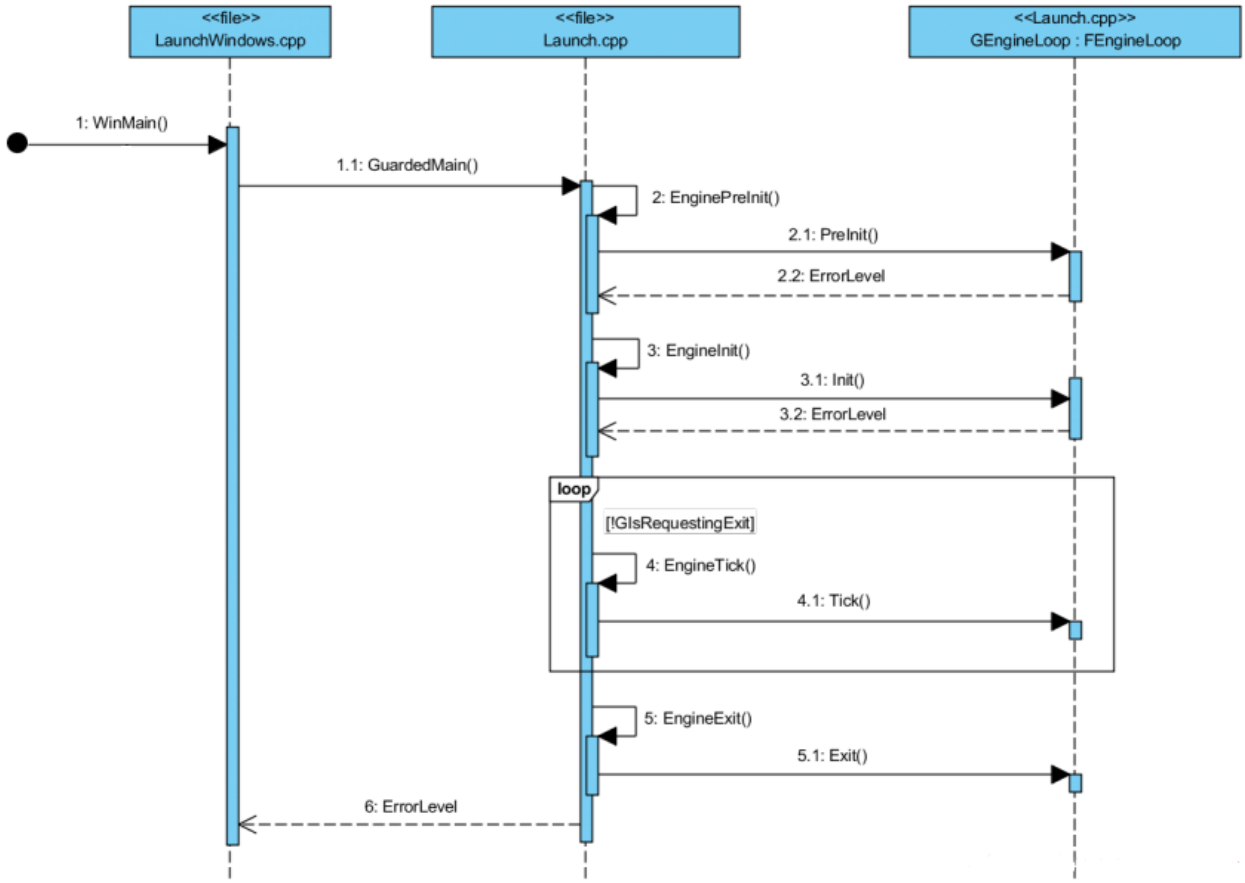


Рисунок 4.2 – Діаграма послідовності

На рисунку 4.2 представлена загальна діаграма послідовності роботи використаного в даному проекті движка Unreal Engine 4. При запуску движка відбувається ініціалізація функції WinMain()(проте движок крос-платформний тож для не Windows платформ точкою входу є функція main() описана в відповідному файлі такі як LaunchLinux.cpp, LaunchMac.cpp, тощо).

Після запуску движка відбувається його ініціалізація – функції GuardedMain() та EnginePreInit(). Основна робота відбувається в функції EngineTick(), що відбувається до поки робота движка не завершиться функцією Exit().

4.3.3 Діаграма компонентів

Рисунок 4.3 – Діаграма компонентів

Беручи увагу обсяжність алгоритму діаграму компонентів було зроблено та представлено у виді компонентів всього ігрового движка, а всі частини алгоритму на представленій на рисунку 4.3 діаграми були інкапсульовані в компоненті “Graphics” – адже алгоритм є саме невід’ємною частиною графічної складової стимульованого світу.

Сама ж діаграма представляє собою сукупність усіх ключових компонентів ігрового движка та засобів необхідних для відображення та стимулювання 3D сцен максимально наближених до реальності. Сам движок був розбитий на 3 головних під-компонента:

- Game Tools (засоби гри, необхідні для розроблення рівней та сцен);
- Game Content (наповнення гри та ігрового рівня такі як графічний інтерфейс, що надається гравцю, аудіо наповнення, тощо);
- Game Utility (можливий додатковий інструментарій, що може знадобитися движку для роботи, це можуть бути математичні бібліотеки, засоби збереження даних, тощо)

4.3.4 Специфікація функцій

Оскільки в роботі і в фінальному продукті кількість класів доволі велика, в даному розділі будуть наведені основні класи та їх функції, повний список буде наведений в додатку А.

Таблиця 4.1 – Класи програми

Назва класу	**Відповідає за**
AmbientSound	Постійний звук на другому плані
CameraActor	Клас камери та все, що він інкапсулює
GeometryCacheActor	Виводить геометричні об’єкти
GeometryCollectionActor	Зберігає інформацію про геометричні об’єкти сцени
SkyLight	Освітлення від неба, промені сонця
LevelBounds	Межі ігрового рівня/сцени
LevelSequenceActor	Клас з методами для запису відео та кіно-зйомки
DirectionalLight	Вид джерела світла – направлене світло
PointLight	Вид джерела світла – точкове джерело
SpotLight	Вид джерела світла - прожектор
Pawn	Клас, що інкапсулює в собі всі методи пов’язані з даними, що вводить користувач
Character	Клас ігрового персонажу
SplineMeshActor	Клас, що інкапсулює в собі всі трьохвимірні об’єкти незвичайної форми

Висновок до розділу

В даному розділі було розглянуто програми, та мови програмування, за допомогою яких було реалізовано програмний продукт. Також були переглянуті умови та вимоги для коректної роботи продукту – пояснено чому висуваються саме такі вимоги та наведені пояснення щодо обраних технологій.

В під-розділі архітектура програмного забезпечення були наведені ключові класи ігрового движка, прояснено, чому використовувались деякі класи. Також були розглянені та побудовані діаграми послідовностей та компонентів. В обох зазначених діаграмах використовувались послідовності та компоненти самого ігрового движка, а не специфічно алгоритму трасування світлових променів, адже він був вбудований безпосередньо в самий ігровий движок.

5 ТЕХНОЛОГІЧНИЙ РОЗДІЛ

5.1 Керівництво користувача

Беручи до увагу специфіку проекту та відсутності екранних форм в даному розділі будуть представлені знімки екрану розробленого ігрового рівня. Керівництво користувача як таке буде доволі простим, адже єдиний спосіб користувача впливати на розроблений продукт – це переміщенням ігрової камери. Для запуску проекту користувачу необхідно його запустити за допомогою сконфігурованого файлу запуску RealisticRendering.exe.

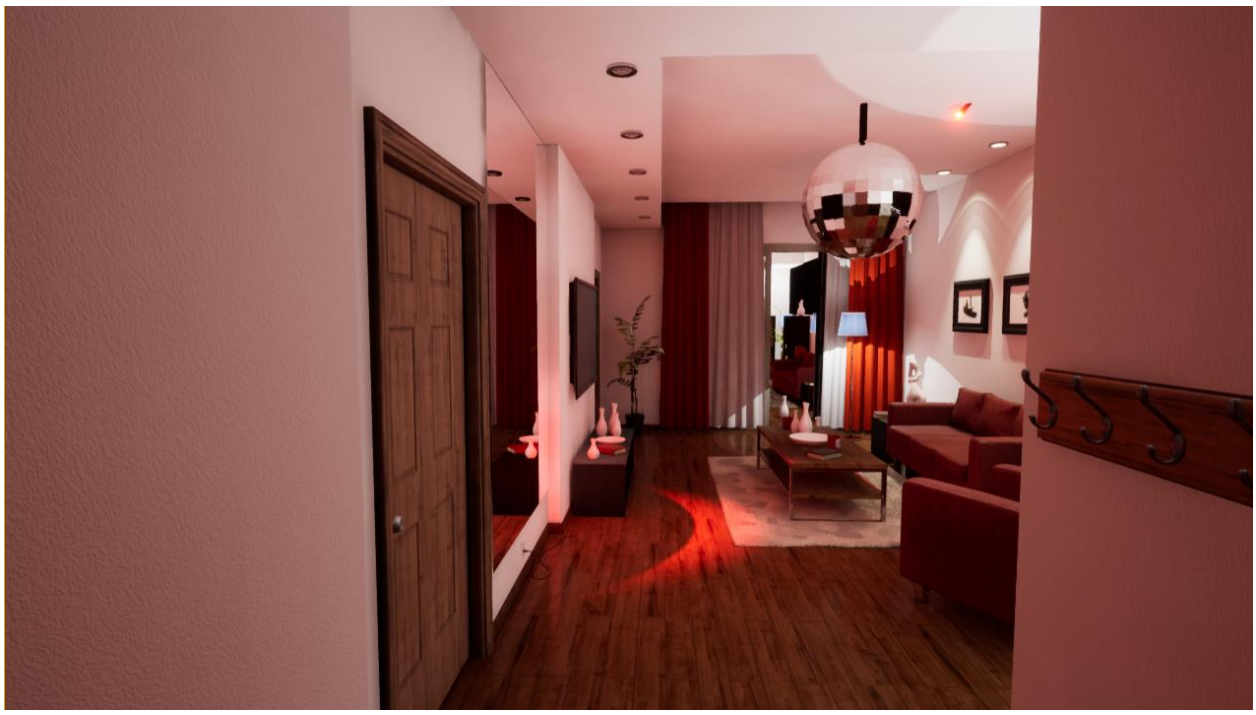


Рисунок 5.1 – Стартова позиція гравця після запуску проекту

Надалі після завантаження рівня користувач може переміщати ігрову камеру за допомогою клавіш:

- “w” для руху вперед;
- “s” для руху назад;
- “a” для руху вліво;
- “d” для руху вправо.

Зміна кута камери здійснюється за допомогою миші.

Далі будуть представлені рисунки – знімки рівня під різними ракурсами.

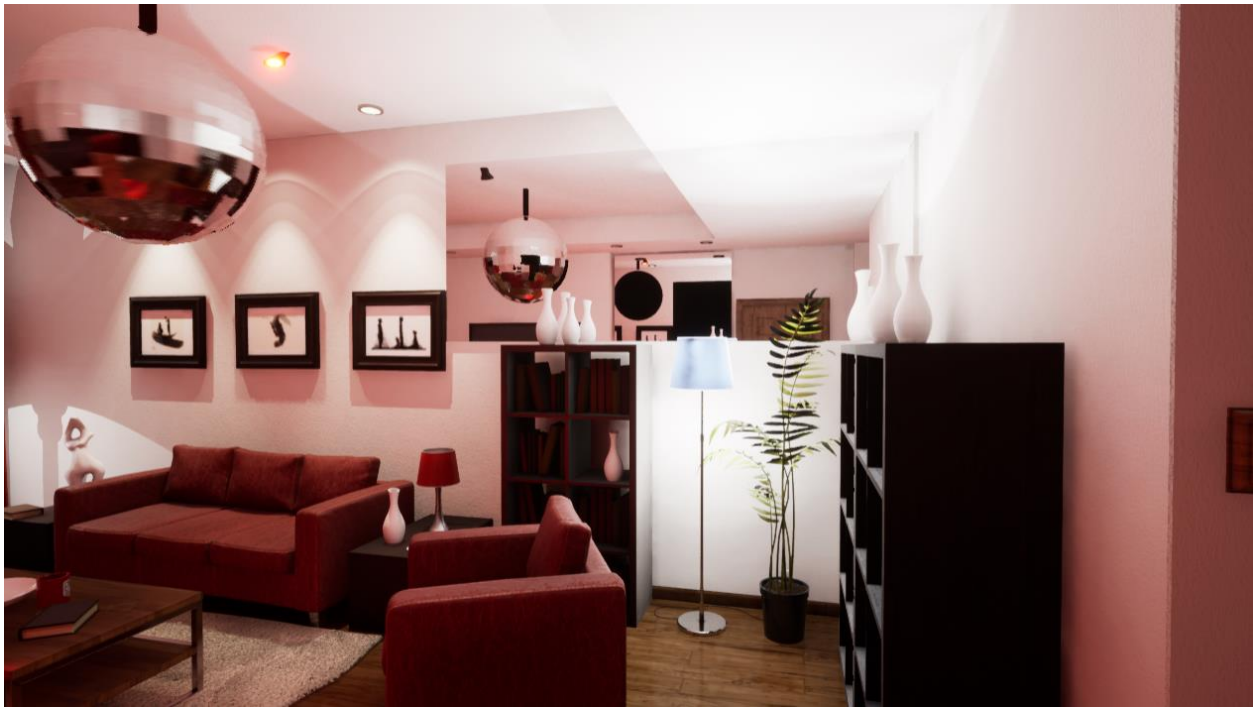


Рисунок 5.2 – Знімок з середини сцени

Рисунок 5.3 – Знімок з середини сцени

Рисунок 5.4 – Знімок з середини сцени

Рисунок 5.5 – Знімок з середини сцени

5.2 Випробування програмного продукту

Головною метою проекту вважалося досягнення максимально чіткої та реалістичного зображення, застосовуючи при цьому мінімальну кількість ресурсів. Саме в ресурсо-затратності і буде полягати тестування розробленого продукту. Проте оскільки кінцевому користувачу не важливий факт затрати ресурсів, а тільки якість та плавність отриманого зображення – будемо враховувати і цей факт. Як міра плавності зображення буде використовуватися величина “Кадри за секунду” – максимальна кількість можливих оброблень усіх пікселей екрану за секунду. У кінематографі за стандарт вважається 24 кадри за секунду, у розробці ж ігор завжди намагаються досягнути результату 30 кадрів та вище – саму таку величину ми й візьмемо за рекомендовану й провіримо, чи може алгоритм досягнути бажаного результату.

5.2.1 Мета випробувань

Метою випробувань являється перевірка відповідності функцій комплексу задач трасування світлових променів вимогам технічного завдання.

5.2.2 Загальні положення

Випробування проводяться на основі наступних документів:

– ГОСТ 34.603–92. Інформаційна технологія. Види випробувань автоматизованих систем;
– ГОСТ РД 50-34.698-90. Автоматизовані системи вимог до змісту документів.

5.2.3 Результати випробувань

Таблиця 5.1 – Запуск системи

Мета тесту	Перевірка функції «Запуск системи»
Початковий стан моделі	Проект закритий
Вхідні дані:	Наявний проект, файл запуску
Схема проведення тесту:	Запустити файл конфігурації RealisticRendering.exe.
Очікуваний результат:	Проект запускається. Помилки та системні повідомлення відсутні.
Стан моделі після проведення випробувань:	Повідомлення про старт завантаження ігрового рівня.

Таблиця 5.2 – Час запуску проекту

Мета тесту	Перевірка часу запуску проекту
Початковий стан моделі	Проект відразу після його запуску
Вхідні дані:	Наявний проект, файл запуску
Схема проведення тесту:	Запустити файл конфігурації RealisticRendering.exe. Зачекати до моменту відображення сцени.
Очікуваний результат:	Коректне відображення ігрової сцени. Час завантаження не перевищує 5 хвилин при використанні рекомендованого апаратного забезпечення.
Стан моделі після проведення випробувань:	Завантажена ігрова сцена.

Таблиця 5.3 – Перевірка на наявність графічних артефактів[8](помилок)

Мета тесту	**Перевірка функції «Вибору лікаря »**
Початковий стан моделі	Завантажена ігрова сцена
Вхідні дані:	-
Схема проведення тесту:	Перевірка вручну, чи наявні в симуляції сцени помилок графічного процесору – чи наявні в сцені так звані графічні артефакти
Очікуваний результат:	Відсутність графічних артефактів
Стан моделі після проведення випробувань:	Коректно відображена ігрова сцена

Таблиця 5.4 – Перевірка контролера движка та камери

Мета тесту	**Перевірка функції «Контроль камери»**
Початковий стан моделі	Коректно відображена ігрова сцена
Вхідні дані:	
Схема проведення тесту:	По черзі натиснути кнопки “w”, “a”, “s”, “d”
Очікуваний результат:	Після натиску на відповідну кнопку камера зміщується відповідно: – При натиску “w” вперед; – При натиску “s” назад; – При натиску “a” вліво; – При натиску “d” вправо.
Стан моделі після проведення випробувань:	Коректно відображена ігрова сцена

Таблиця 5.5 – Перевірка зміни кута камери

Мета тесту	**Перевірка функції «Оцінити лікаря»**
Початковий стан моделі	Коректно відображена ігрова сцена
Вхідні дані:	
Схема проведення тесту:	Рух мишою в різних напрямах
Очікуваний результат:	Камера в сцені змінює кут огляду відповідно до руху миші
Стан моделі після проведення випробувань:	Коректно відображена ігрова сцена

Таблиця 5.6 – Перевірка кадрової частоти при відключеному алгоритмі трасування променів

Мета тесту	**Перевірка кадрової частоти**
Початковий стан моделі	Коректно відображена ігрова сцена
Вхідні дані:	
Схема проведення тесту:	Ввести команду r.GPUStatsEnabled, рухатись по рівню, заміряючи кадрову частоту
Очікуваний результат:	Частота кадрів перебуває на рівні 90 або вище впродовж всієї сесії
Стан моделі після проведення випробувань:	Коректно відображена ігрова сцена

Таблиця 5.7 – Перевірка кадрової частоти при увімкненому алгоритмі трасування променів, з параметром відбиття 1

Мета тесту	**Перевірка кадрової частоти**
Початковий стан моделі	Коректно відображена ігрова сцена
Вхідні дані:	
Схема проведення тесту:	Ввести команду r.GPUStatsEnabled, рухатись по рівню, заміряючи кадрову частоту
Очікуваний результат:	Частота кадрів перебуває на рівні 30 або вище впродовж всієї сесії
Стан моделі після проведення випробувань:	Коректно відображена ігрова сцена

Таблиця 5.8 – Перевірка кадрової частоти при увімкненому алгоритмі трасування променів, з параметром відбиття 2

Мета тесту	**Перевірка кадрової частоти**
Початковий стан моделі	Коректно відображена ігрова сцена
Вхідні дані:	
Схема проведення тесту:	Ввести команду r.GPUStatsEnabled, рухатись по рівню, заміряючи кадрову частоту
Очікуваний результат:	Частота кадрів перебуває на рівні 30 або вище впродовж всієї сесії
Стан моделі після проведення випробувань:	Коректно відображена ігрова сцена

Висновок до розділу

В даному розділі було розглянуто детальну користувацьку інструкцію з використання розробленого ігрового рівня, можливі способи використання сцени та метрики, що були використані для заміру відповідності проекту представленим вимогам.

Були написані варіанти тестування системи та знайдені можливі вразливі точки системи – для таких точок були написані варіанти тестування та список можливих результатів. Також було проведене тестування найбільш важливих функціональних та технічних характеристик системи та проведений аналіз ресурсо-затраності системи.

ЗАГАЛЬНІ ВИСНОВКИ

В даному дипломному проекті було розглянуто роботу алгоритму трасування світлових променів в реальному часі – в основному для сфери комп'ютерних ігор, проте в ході роботи були приведені і деякі приклади роботи кінематографу з вищезгаданим алгоритмом. Загалом, були наведені приклади роботи обробників трьох-вимірних сцен в сучасності та порівняно їх роботу з розробленим алгоритмом. Також були приведені аргументи та було наглядно продемонстровані переваги трасування променів в реальному часі для обробки зображень та ефектів поверхонь.

В пояснювальній записці було також обґрунтовано актуальність методу в сучасній сфері та можливі точки застосування, де використання алгоритму вплине на індустрію найбільше.

Можливо саме трасування променів повністю змінить світ комп'ютерної графіки, тому я вибрав саме таку тему дипломного проекту – для можливого покращення та кращого розуміння останніх технологій для обробки трьох-вимірних сцен. Для розробки був обраний движок Unreal Engine 4, адже саме в ньому з'явилась підтримка трасувальників вперше і з тих пір движок є піонером в імплементації описаних технологій. Не останньою причиною є й те, що код движку знаходиться в вільному доступі, тож я вважаю саме завдяки ньому ігрові розробники будуть рухати та розвивати комп'ютерну графіку в виді повної симуляції, тобто трасування променів, вперед.

ПЕРЕЛІК ПОСИЛАНЬ

1. Концепція трьох-вимірного світу [Електронний ресурс]: http://computer-graphics.narod.ru/3d.html
2. Полігони, складові кожної трьох-вимірної симуляції [Електронний ресурс]: https://www.javatpoint.com/computer-graphics-polygon
3. Визначення – ігровий движок [Електронний ресурс]: https://unity3d.com/what-is-a-game-engine
4. Windows OS [Електронний ресурс]: https://www.microsoft.com/uk-ua/windows
5. Unreal [Електронний ресурс]: https://www.unrealengine.com
6. Visual Studio [Електронний ресурс]: https://visualstudio.microsoft.com
7. c++ [Електронний ресурс]: http://www.cplusplus.com
8. Графічні артефакти - визначення [Електронний ресурс]: https://en.wikipedia.org/wiki/Digital_artifact
9. Ray tracing as a method to produce realistic images [Електронний ресурс]: https://www.cs.utah.edu/~shirley/books/fcg2/rt.pdf
10. Ray Tracing Algorithms – Theory and Practice [Електронний ресурс]: https://www.researchgate.net/publication/230583216_Ray_Tracing_Algorithms_-_Theory_and_Practice
11. Ray Tracing: Graphics for the Masses [Електронний ресурс]: https://www.cs.unc.edu/~rademach/xroads-RT/RTarticle.html#color
12. An Improved Illumination Model for Shaded Display [Електронний ресурс]: http://artis.imag.fr/Members/David.Roger/whitted.pdf
13. An Introduction to Ray Tracing [Електронний ресурс]: http://www.realtimerendering.com/raytracing/An-Introduction-to-Ray-Tracing-The-Morgan-Kaufmann-Series-in-Computer-Graphics-.pdf
14. What is ray tracing, and how does it differ from game to game? [Електронний ресурс]: https://www.pcgamer.com/what-is-ray-tracing/

15. UML diagrams in Unreal Engine [Електронний ресурс]: https://wiki.unrealengine.com/index.php?title=UML
16. Class Hierarchy in Unreal Engine [Електронний ресурс]: https://api.unrealengine.com/INT/API/ClassHierarchy/index.html
17. Real-Time Ray Tracing - An overview of Ray Tracing in Unreal Engine 4 [Електронний ресурс]: https://docs.unrealengine.com/en-US/Engine/Rendering/RayTracing/index.html
18. Path Tracer - An overview of the Path Tracer in Unreal Engine 4. [Електронний ресурс]: https://docs.unrealengine.com/en-US/Engine/Rendering/RayTracing/PathTracer/index.html
19. Трассировка лучей на GPU в Unity [Електронний ресурс]: https://habr.com/ru/post/355018/
20. Why do so many video games have an aversion to using working mirrors in their environments? [Електронний ресурс]: https://www.quora.com/Why-do-so-many-video-games-have-an-aversion-to-using-working-mirrors-in-their-environments
21. The NVIDIA Turing GPU Architecture Deep Dive: Prelude to GeForce RTX [Електронний ресурс]: https://www.anandtech.com/show/13282/nvidia-turing-architecture-deep-dive/3
22. The NVIDIA GeForce RTX 2080 Ti & RTX 2080 Founders Edition Review: Foundations For A Ray Traced Future [Електронний ресурс]: https://www.anandtech.com/show/13346/the-nvidia-geforce-rtx-2080-ti-and-2080-founders-edition-review/3

Додаток А

Тексти програмного коду

Трасування променів в реальному часі для комп'ютерних ігор

(Найменування програми (документа))

DVD-R

(Вид носія даних)

39 арк, 271 Кб

(Обсяг програми (документа) , арк.,) Кб)

Київ – 2019 року

Лістинг програми буде приведено в .cpp файлах на мові C++. В додатку будуть приведені виключно файли з безпосередньою імплементацією алгоритму а не загальні файли, щоб нести тільки релевантну інформацію в цьому додатку.

1. RayTracingDebug.cpp

```
// Copyright 1998-2019 Epic Games, Inc. All Rights Reserved.

#include "RHI.h"

#if RHI_RAYTRACING

#include "DeferredShadingRenderer.h"
#include "GlobalShader.h"
#include "SceneRenderTargets.h"
#include "RenderGraphBuilder.h"
#include "SceneUtils.h"
#include "RayTracingDebugDefinitions.h"

#define LOCTEXT_NAMESPACE "RayTracingDebugVisualizationMenuCommands"

DECLARE_GPU_STAT(RayTracingDebug);

class FRayTracingDebugRGS : public FGlobalShader
{
	DECLARE_GLOBAL_SHADER(FRayTracingDebugRGS)
	SHADER_USE_ROOT_PARAMETER_STRUCT(FRayTracingDebugRGS, FGlobalShader)

	BEGIN_SHADER_PARAMETER_STRUCT(FParameters, )
		SHADER_PARAMETER(uint32, VisualizationMode)
		SHADER_PARAMETER_SRV(RaytracingAccelerationStructure, TLAS)
		SHADER_PARAMETER_RDG_TEXTURE_UAV(RWTexture2D<float4>, Output)
		SHADER_PARAMETER_STRUCT_REF(FViewUniformShaderParameters,
ViewUniformBuffer)
	END_SHADER_PARAMETER_STRUCT()

	static bool ShouldCompilePermutation(const FGlobalShaderPermutationParameters&
Parameters)
	{
		return ShouldCompileRayTracingShadersForProject(Parameters.Platform);
	}
};
IMPLEMENT_GLOBAL_SHADER(FRayTracingDebugRGS,
"/Engine/Private/RayTracing/RayTracingDebug.usf", "RayTracingDebugMainRGS", SF_RayGen);

class FRayTracingDebugMS : public FGlobalShader
{
	DECLARE_SHADER_TYPE(FRayTracingDebugMS, Global);

public:

	static bool ShouldCompilePermutation(const FGlobalShaderPermutationParameters&
Parameters)
	{
		return ShouldCompileRayTracingShadersForProject(Parameters.Platform);
	}

	FRayTracingDebugMS() = default;
```

```
	FRayTracingDebugMS(const ShaderMetaType::CompiledShaderInitializerType&
Initializer)
		: FGlobalShader(Initializer)
	{}
};
IMPLEMENT_SHADER_TYPE(, FRayTracingDebugMS,
TEXT("/Engine/Private/RayTracing/RayTracingDebug.usf"), TEXT("RayTracingDebugMainMS"),
SF_RayMiss);


class FRayTracingDebugCHS : public FGlobalShader
{
	DECLARE_SHADER_TYPE(FRayTracingDebugCHS, Global);

public:

	static bool ShouldCompilePermutation(const FGlobalShaderPermutationParameters&
Parameters)
	{
		return ShouldCompileRayTracingShadersForProject(Parameters.Platform);
	}

	FRayTracingDebugCHS() = default;
	FRayTracingDebugCHS(const ShaderMetaType::CompiledShaderInitializerType&
Initializer)
		: FGlobalShader(Initializer)
	{}
};

// Dummy shader permutations to test hit group API
IMPLEMENT_SHADER_TYPE(, FRayTracingDebugCHS,
TEXT("/Engine/Private/RayTracing/RayTracingDebug.usf"), TEXT("RayTracingDebugMainCHS"),
SF_RayHitGroup);

void FDeferredShadingSceneRenderer::PrepareRayTracingDebug(const FViewInfo& View,
TArray<FRayTracingShaderRHIParamRef>& OutRayGenShaders)
{
	// Declare all RayGen shaders that require material closest hit shaders to be
bound

	auto RayGenShader = View.ShaderMap->GetShader<FRayTracingDebugRGS>();
	OutRayGenShaders.Add(RayGenShader->GetRayTracingShader());
}

void FDeferredShadingSceneRenderer::RenderRayTracingDebug(FRHICommandListImmediate&
RHICmdList, const FViewInfo& View)
{
	static TMap<FName, uint32> RayTracingDebugVisualizationModes;
	if (RayTracingDebugVisualizationModes.Num() == 0)
	{
		RayTracingDebugVisualizationModes.Emplace(FName(*LOCTEXT("Radiance",
"Radiance").ToString()),
	RAY_TRACING_DEBUG_VIZ_RADIANCE);
		RayTracingDebugVisualizationModes.Emplace(FName(*LOCTEXT("World Normal",
"World Normal").ToString()),
	RAY_TRACING_DEBUG_VIZ_WORLD_NORMAL);
		RayTracingDebugVisualizationModes.Emplace(FName(*LOCTEXT("BaseColor",
"BaseColor").ToString()),
	RAY_TRACING_DEBUG_VIZ_BASE_COLOR);
		RayTracingDebugVisualizationModes.Emplace(FName(*LOCTEXT("DiffuseColor",
"DiffuseColor").ToString()),
	RAY_TRACING_DEBUG_VIZ_DIFFUSE_COLOR);
```

```
		RayTracingDebugVisualizationModes.Emplace(FName(*LOCTEXT("SpecularColor",
"SpecularColor").ToString()),
	RAY_TRACING_DEBUG_VIZ_SPECULAR_COLOR);
		RayTracingDebugVisualizationModes.Emplace(FName(*LOCTEXT("Opacity",
"Opacity").ToString()),
		RAY_TRACING_DEBUG_VIZ_OPACITY);
		RayTracingDebugVisualizationModes.Emplace(FName(*LOCTEXT("Metallic",
"Metallic").ToString()),
	RAY_TRACING_DEBUG_VIZ_METALLIC);
		RayTracingDebugVisualizationModes.Emplace(FName(*LOCTEXT("Specular",
"Specular").ToString()),
	RAY_TRACING_DEBUG_VIZ_SPECULAR);
		RayTracingDebugVisualizationModes.Emplace(FName(*LOCTEXT("Roughness",
"Roughness").ToString()),
	RAY_TRACING_DEBUG_VIZ_ROUGHNESS);
		RayTracingDebugVisualizationModes.Emplace(FName(*LOCTEXT("Ior",
"Ior").ToString()),
			RAY_TRACING_DEBUG_VIZ_IOR);
		RayTracingDebugVisualizationModes.Emplace(FName(*LOCTEXT("ShadingModelID",
"ShadingModelID").ToString()),
	RAY_TRACING_DEBUG_VIZ_SHADING_MODEL);
		RayTracingDebugVisualizationModes.Emplace(FName(*LOCTEXT("BlendingMode",
"BlendingMode").ToString()),
	RAY_TRACING_DEBUG_VIZ_BLENDING_MODE);

	RayTracingDebugVisualizationModes.Emplace(FName(*LOCTEXT("PrimitiveLightingChannel
Mask", "PrimitiveLightingChannelMask").ToString()),
	RAY_TRACING_DEBUG_VIZ_LIGHTING_CHANNEL_MASK);
		RayTracingDebugVisualizationModes.Emplace(FName(*LOCTEXT("CustomData",
"CustomData").ToString()),
	RAY_TRACING_DEBUG_VIZ_CUSTOM_DATA);
		RayTracingDebugVisualizationModes.Emplace(FName(*LOCTEXT("GBufferAO",
"GBufferAO").ToString()),
	RAY_TRACING_DEBUG_VIZ_GBUFFER_AO);

	RayTracingDebugVisualizationModes.Emplace(FName(*LOCTEXT("IndirectIrradiance",
"IndirectIrradiance").ToString()),
	RAY_TRACING_DEBUG_VIZ_INDIRECT_IRRADIANCE);
		RayTracingDebugVisualizationModes.Emplace(FName(*LOCTEXT("World Position",
"World Position").ToString()),
	RAY_TRACING_DEBUG_VIZ_WORLD_POSITION);
		RayTracingDebugVisualizationModes.Emplace(FName(*LOCTEXT("HitKind",
"HitKind").ToString()),
		RAY_TRACING_DEBUG_VIZ_HITKIND);
		RayTracingDebugVisualizationModes.Emplace(FName(*LOCTEXT("Barycentrics",
"Barycentrics").ToString()),
	RAY_TRACING_DEBUG_VIZ_BARYCENTRICS);
	}

	uint32 DebugVisualizationMode =
RayTracingDebugVisualizationModes.FindRef(View.CurrentRayTracingDebugVisualizationMode);

	if (DebugVisualizationMode == RAY_TRACING_DEBUG_VIZ_BARYCENTRICS)
	{
		return RenderRayTracingBarycentrics(RHICmdList, View);
	}

	FSceneRenderTargets& SceneContext = FSceneRenderTargets::Get(RHICmdList);

	FRDGBuilder GraphBuilder(RHICmdList);

	TShaderMap<FGlobalShaderType>* ShaderMap = GetGlobalShaderMap(FeatureLevel);
```

```
	auto RayGenShader = ShaderMap->GetShader<FRayTracingDebugRGS>();
	auto ClosestHitShader = ShaderMap->GetShader<FRayTracingDebugCHS>();
	auto MissShader = ShaderMap->GetShader<FRayTracingDebugMS>();

	FRHIRayTracingPipelineState* Pipeline = View.RayTracingMaterialPipeline;

	FRayTracingSceneRHIParamRef RayTracingSceneRHI =
View.RayTracingScene.RayTracingSceneRHI;

	FRayTracingDebugRGS::FParameters* RayGenParameters =
GraphBuilder.AllocParameters<FRayTracingDebugRGS::FParameters>();

	RayGenParameters->VisualizationMode = DebugVisualizationMode;
	RayGenParameters->TLAS = RayTracingSceneRHI->GetShaderResourceView();
	RayGenParameters->ViewUniformBuffer = View.ViewUniformBuffer;
	RayGenParameters->Output =
GraphBuilder.CreateUAV(GraphBuilder.RegisterExternalTexture(SceneContext.GetSceneColor())
);

	FIntRect ViewRect = View.ViewRect;

	GraphBuilder.AddPass(
		RDG_EVENT_NAME("RayTracingDebug"),
		RayGenParameters,
		ERenderGraphPassFlags::Compute,
		[this, RayGenParameters, RayGenShader, &SceneContext, RayTracingSceneRHI,
Pipeline, ViewRect](FRHICommandList& RHICmdList)
	{
		SCOPED_GPU_STAT(RHICmdList, RayTracingDebug);

		FRayTracingShaderBindingsWriter GlobalResources;
		SetShaderParameters(GlobalResources, RayGenShader, *RayGenParameters);

		RHICmdList.RayTraceDispatch(Pipeline, RayGenShader->GetRayTracingShader(),
RayTracingSceneRHI, GlobalResources, ViewRect.Size().X, ViewRect.Size().Y);
	});

	GraphBuilder.Execute();
}

#undef LOCTEXT_NAMESPACE
#endif
```

2. RayTracingDeferredMaterials.cpp

```
// Copyright 1998-2019 Epic Games, Inc. All
Rights Reserved.

#include "RayTracingDeferredMaterials.h"
#include "RHIDefinitions.h"
#include "RenderCore.h"
#include "GlobalShader.h"
#include "ShaderParameterStruct.h"
#include "RenderGraphUtils.h"
#include "DeferredShadingRenderer.h"
#include "PipelineStateCache.h"

#if RHI_RAYTRACING

class FRayTracingDeferredMaterialCHS : public FGlobalShader
{
	DECLARE_GLOBAL_SHADER(FRayTracingDeferredMaterialCHS)
	SHADER_USE_ROOT_PARAMETER_STRUCT(FRayTracingDeferredMaterialCHS, FGlobalShader)

		static bool ShouldCompilePermutation(const
```

```
FGlobalShaderPermutationParameters& Parameters)
	{
		return ShouldCompileRayTracingShadersForProject(Parameters.Platform);
	}

	using FParameters = FEmptyShaderParameters;
};

class FRayTracingDeferredMaterialMS : public FGlobalShader
{
	DECLARE_GLOBAL_SHADER(FRayTracingDeferredMaterialMS)
	SHADER_USE_ROOT_PARAMETER_STRUCT(FRayTracingDeferredMaterialMS, FGlobalShader)

		static bool ShouldCompilePermutation(const
FGlobalShaderPermutationParameters& Parameters)
	{
		return ShouldCompileRayTracingShadersForProject(Parameters.Platform);
	}

	using FParameters = FEmptyShaderParameters;
};

IMPLEMENT_GLOBAL_SHADER(FRayTracingDeferredMaterialCHS,
"/Engine/Private/RayTracing/RayTracingDeferredMaterials.usf", "DeferredMaterialCHS",
SF_RayHitGroup);
IMPLEMENT_GLOBAL_SHADER(FRayTracingDeferredMaterialMS,
"/Engine/Private/RayTracing/RayTracingDeferredMaterials.usf", "DeferredMaterialMS",
SF_RayMiss);

FRHIRayTracingPipelineState*
FDeferredShadingSceneRenderer::BindRayTracingDeferredMaterialGatherPipeline(FRHICommandLi
st& RHICmdList, const FViewInfo& View, FRayTracingShaderRHIParamRef RayGenShader)
{
	SCOPE_CYCLE_COUNTER(STAT_BindRayTracingPipeline);

	FRayTracingPipelineStateInitializer Initializer;

	FRayTracingShaderRHIParamRef RayGenShaderTable[] = { RayGenShader };
	Initializer.SetRayGenShaderTable(RayGenShaderTable);

	auto MissShader = View.ShaderMap->GetShader<FRayTracingDeferredMaterialMS>();
	FRayTracingShaderRHIParamRef MissShaderTable[] = { MissShader-
>GetRayTracingShader() };
	Initializer.SetMissShaderTable(MissShaderTable);

	Initializer.MaxPayloadSizeInBytes = 12; // sizeof FDeferredMaterialPayload

	// Get the ray tracing materials
	auto ClosestHitShader = View.ShaderMap-
>GetShader<FRayTracingDeferredMaterialCHS>();
	FRayTracingShaderRHIParamRef HitShaderTable[] = { ClosestHitShader-
>GetRayTracingShader() };
	Initializer.SetHitGroupTable(HitShaderTable);

	FRHIRayTracingPipelineState* PipelineState =
PipelineStateCache::GetAndOrCreateRayTracingPipelineState(Initializer);

	const FViewInfo& ReferenceView = Views[0];

	for (const FVisibleRayTracingMeshCommand VisibleMeshCommand :
ReferenceView.VisibleRayTracingMeshCommands)
	{
		const FRayTracingMeshCommand& MeshCommand =
```

```
 *VisibleMeshCommand.RayTracingMeshCommand;

		const uint32 HitGroupIndex = 0; // Force the default CHS to be used on all
geometry

		const uint32 ShaderSlot = 0; // Multiple shader slots can be used for
different ray types. Slot 0 is the primary material slot.
		const uint32 MaterialIndexInUserData = MeshCommand.MaterialShaderIndex;
		RHICmdList.SetRayTracingHitGroup(
			View.RayTracingScene.RayTracingSceneRHI,
			VisibleMeshCommand.InstanceIndex,
			MeshCommand.GeometrySegmentIndex,
			ShaderSlot,
			PipelineState,
			HitGroupIndex,
			0,
			nullptr,
			MaterialIndexInUserData);
	}

	return PipelineState;
}

class FMaterialSortCS : public FGlobalShader
{
	DECLARE_GLOBAL_SHADER(FMaterialSortCS);
	SHADER_USE_PARAMETER_STRUCT(FMaterialSortCS, FGlobalShader);

	class FSortSize : SHADER_PERMUTATION_INT("DIM_SORT_SIZE", 5);

	using FPermutationDomain = TShaderPermutationDomain<FSortSize>;

	BEGIN_SHADER_PARAMETER_STRUCT(FParameters, )
		SHADER_PARAMETER(int, NumTotalEntries)
		SHADER_PARAMETER_RDG_BUFFER_UAV(StructuredBuffer<FDeferredMaterialPayload>,
MaterialBuffer)
	END_SHADER_PARAMETER_STRUCT()

	static bool ShouldCompilePermutation(const FGlobalShaderPermutationParameters&
Parameters)
	{
		return Parameters.Platform == SP_PCD3D_SM5;
	}
};


IMPLEMENT_GLOBAL_SHADER(FMaterialSortCS, "/Engine/Private/RayTracing/MaterialSort.usf",
"MaterialSortLocal", SF_Compute);

void SortDeferredMaterials(
	FRDGBuilder& GraphBuilder,
	const FViewInfo& View,
	uint32 SortSize,
	uint32 NumElements,
	FRDGBufferRef MaterialBuffer)
{
	if (SortSize == 0)
	{
		return;
	}
	SortSize = FMath::Min(SortSize, 5u);

	// Setup shader and parameters
```

```
		FMaterialSortCS::FParameters* PassParameters =
GraphBuilder.AllocParameters<FMaterialSortCS::FParameters>();
		PassParameters->NumTotalEntries = NumElements;
		PassParameters->MaterialBuffer = GraphBuilder.CreateUAV(MaterialBuffer);

		FMaterialSortCS::FPermutationDomain PermutationVector;
		PermutationVector.Set<FMaterialSortCS::FSortSize>(SortSize - 1);

		// Sort size represents an index into pow2 sizes, not an actual size, so convert
to the actual number of elements being sorted
		const uint32 ElementBlockSize = 256 * (1 << (SortSize - 1));
		const uint32 DispatchWidth = FMath::DivideAndRoundUp(NumElements,
ElementBlockSize);

		TShaderMapRef<FMaterialSortCS> SortShader(View.ShaderMap, PermutationVector);
		FComputeShaderUtils::AddPass(
			GraphBuilder,
			RDG_EVENT_NAME("MaterialSort SortSize=%d NumElements=%d", ElementBlockSize,
NumElements),
			*SortShader,
			PassParameters,
			FIntVector(DispatchWidth, 1, 1));
}

#else // RHI_RAYTRACING

void SortDeferredMaterials(
		FRDGBuilder& GraphBuilder,
		const FViewInfo& View,
		uint32 SortSize,
		uint32 NumElements,
		FRDGBufferRef MaterialBuffer)
{
		checkNoEntry();
}

#endif // RHI_RAYTRACING
```

3. RayTracingDynamicGeometry.cpp

```
// Copyright 1998-2019 Epic Games, Inc. All Rights Reserved.

#include "MeshMaterialShader.h"
#include "ScenePrivate.h"
#include "MeshPassProcessor.inl"
#include "RayTracingDynamicGeometryCollection.h"

#if RHI_RAYTRACING

static bool IsSupportedDynamicVertexFactoryType(const FVertexFactoryType*
VertexFactoryType)
{
		return VertexFactoryType ==
FindVertexFactoryType(FName(TEXT("FNiagaraSpriteVertexFactory"), FNAME_Find));
}

class FRayTracingDynamicGeometryConverterCS : public FMeshMaterialShader
{
		DECLARE_SHADER_TYPE(FRayTracingDynamicGeometryConverterCS, MeshMaterial);
public:
		FRayTracingDynamicGeometryConverterCS(const
FMeshMaterialShaderType::CompiledShaderInitializerType& Initializer)
			: FMeshMaterialShader(Initializer)
		{
```

```
			PassUniformBuffer.Bind(Initializer.ParameterMap,
FSceneTexturesUniformParameters::StaticStructMetadata.GetShaderVariableName());

			RWVertexPositions.Bind(Initializer.ParameterMap, TEXT("VertexPositions"));
			NumMaxVertices.Bind(Initializer.ParameterMap, TEXT("NumMaxVertices"));
			NumCPUVertices.Bind(Initializer.ParameterMap, TEXT("NumCPUVertices"));
		}

		FRayTracingDynamicGeometryConverterCS() = default;

		static bool ShouldCompilePermutation(EShaderPlatform Platform, const FMaterial*
Material, const FVertexFactoryType* VertexFactoryType)
		{
			// #dxr_todo: this should also check if ray tracing is enabled for the
target platform & project
			return IsSupportedDynamicVertexFactoryType(VertexFactoryType) &&
ShouldCompileRayTracingShadersForProject(Platform);
		}

		virtual bool Serialize(FArchive& Ar) override
		{
			bool bShaderHasOutdatedParameters = FMeshMaterialShader::Serialize(Ar);
			Ar << RWVertexPositions;
			Ar << NumMaxVertices;
			Ar << NumCPUVertices;
			return bShaderHasOutdatedParameters;
		}

		void GetShaderBindings(
			const FScene* Scene,
			ERHIFeatureLevel::Type FeatureLevel,
			const FPrimitiveSceneProxy* PrimitiveSceneProxy,
			const FMaterialRenderProxy& MaterialRenderProxy,
			const FMaterial& Material,
			const FMeshPassProcessorRenderState& DrawRenderState,
			const FMeshMaterialShaderElementData& ShaderElementData,
			FMeshDrawSingleShaderBindings& ShaderBindings) const
		{
			FMeshMaterialShader::GetShaderBindings(Scene, FeatureLevel,
PrimitiveSceneProxy, MaterialRenderProxy, Material, DrawRenderState, ShaderElementData,
ShaderBindings);
		}

		void GetElementShaderBindings(
			const FScene* Scene,
			const FSceneView* ViewIfDynamicMeshCommand,
			const FVertexFactory* VertexFactory,
			bool bShaderRequiresPositionOnlyStream,
			ERHIFeatureLevel::Type FeatureLevel,
			const FPrimitiveSceneProxy* PrimitiveSceneProxy,
			const FMeshBatch& MeshBatch,
			const FMeshBatchElement& BatchElement,
			const FMeshMaterialShaderElementData& ShaderElementData,
			FMeshDrawSingleShaderBindings& ShaderBindings,
			FVertexInputStreamArray& VertexStreams) const
		{
			FMeshMaterialShader::GetElementShaderBindings(Scene,
ViewIfDynamicMeshCommand, VertexFactory, bShaderRequiresPositionOnlyStream, FeatureLevel,
PrimitiveSceneProxy, MeshBatch, BatchElement, ShaderElementData, ShaderBindings,
VertexStreams);
		}

		FRWShaderParameter RWVertexPositions;
```

```
	FShaderParameter NumMaxVertices;
	FShaderParameter NumCPUVertices;
};

IMPLEMENT_MATERIAL_SHADER_TYPE(, FRayTracingDynamicGeometryConverterCS,
TEXT("/Engine/Private/RayTracing/RayTracingDynamicMesh.usf"),
TEXT("RayTracingDynamicGeometryConverterCS"), SF_Compute);

FRayTracingDynamicGeometryCollection::FRayTracingDynamicGeometryCollection()
{
	DispatchCommands = MakeUnique<TArray<FMeshComputeDispatchCommand>>();
}

void FRayTracingDynamicGeometryCollection::AddDynamicMeshBatchForGeometryUpdate(
	const FScene* Scene,
	const FSceneView* View,
	const FPrimitiveSceneProxy* PrimitiveSceneProxy,
	const FMeshBatch& MeshBatch,
	FRayTracingGeometry& Geometry,
	uint32 NumMaxVertices,
	FRWBuffer& Buffer)
{
	const FMaterialRenderProxy* FallbackMaterialRenderProxyPtr = nullptr;
	const FMaterial& Material = MeshBatch.MaterialRenderProxy-
>GetMaterialWithFallback(Scene->GetFeatureLevel(), FallbackMaterialRenderProxyPtr);

	const FMaterialRenderProxy& MaterialRenderProxy = FallbackMaterialRenderProxyPtr ?
*FallbackMaterialRenderProxyPtr : *MeshBatch.MaterialRenderProxy;

	TMeshProcessorShaders<
		FMeshMaterialShader,
		FMeshMaterialShader,
		FMeshMaterialShader,
		FMeshMaterialShader,
		FMeshMaterialShader,
		FMeshMaterialShader,
		FRayTracingDynamicGeometryConverterCS> Shaders;

	FMeshComputeDispatchCommand DispatchCmd;

	FRayTracingDynamicGeometryConverterCS* Shader =
Material.GetShader<FRayTracingDynamicGeometryConverterCS>(MeshBatch.VertexFactory-
>GetType());
	DispatchCmd.MaterialShader = Shader;
	FMeshDrawShaderBindings& ShaderBindings = DispatchCmd.ShaderBindings;

	Shaders.ComputeShader = Shader;
	ShaderBindings.Initialize(Shaders.GetUntypedShaders());

	FMeshMaterialShaderElementData ShaderElementData;
	ShaderElementData.InitializeMeshMaterialData(View, PrimitiveSceneProxy, MeshBatch,
-1, false);

	FMeshDrawSingleShaderBindings SingleShaderBindings =
ShaderBindings.GetSingleShaderBindings(SF_Compute);
	FMeshPassProcessorRenderState DrawRenderState(Scene-
>UniformBuffers.ViewUniformBuffer, Scene->UniformBuffers.OpaqueBasePassUniformBuffer);
	Shader->GetShaderBindings(Scene, Scene->GetFeatureLevel(), PrimitiveSceneProxy,
MaterialRenderProxy, Material, DrawRenderState, ShaderElementData, SingleShaderBindings);

	FVertexInputStreamArray DummyArray;
	Shader->GetElementShaderBindings(Scene, View, MeshBatch.VertexFactory, false,
Scene->GetFeatureLevel(), PrimitiveSceneProxy, MeshBatch, MeshBatch.Elements[0],
```

```
 ShaderElementData, SingleShaderBindings, DummyArray);

	DispatchCmd.TargetBuffer = &Buffer;
	DispatchCmd.TargetGeometry = &Geometry;
	DispatchCmd.NumMaxVertices = NumMaxVertices;
	DispatchCmd.NumCPUVertices = MeshBatch.Elements[0].NumPrimitives * 2 *
MeshBatch.Elements[0].NumInstances;

	uint32 DesiredVertexBufferSize = FMath::DivideAndRoundUp((uint32)(NumMaxVertices *
sizeof(FVector)), 4096u) * 4096u;
	if (Buffer.NumBytes != DesiredVertexBufferSize)
	{
		int32 OriginalSize = Buffer.NumBytes;
		Buffer.Initialize(4, DesiredVertexBufferSize / 4, PF_R32_FLOAT,
BUF_UnorderedAccess | BUF_ShaderResource, TEXT("RayTracingDynamicVertexBuffer"));
	}

	check(DispatchCmd.TargetBuffer->NumBytes >= NumMaxVertices * sizeof(FVector));

#if MESH_DRAW_COMMAND_DEBUG_DATA
	FMeshProcessorShaders ShadersForDebug = Shaders.GetUntypedShaders();
	ShaderBindings.Finalize(&ShadersForDebug);
#endif

	DispatchCommands->Add(DispatchCmd);

	check(Geometry.IsInitialized());
	Geometry.Initializer.PositionVertexBuffer = Buffer.Buffer;
	Geometry.Initializer.TotalPrimitiveCount = NumMaxVertices / 3;
	Geometry.RayTracingGeometryRHI =
RHICreateRayTracingGeometry(Geometry.Initializer);
}

void FRayTracingDynamicGeometryCollection::DispatchUpdates(FRHICommandListImmediate&
RHICmdList)
{
	if (DispatchCommands->Num() > 0)
	{
		SCOPED_DRAW_EVENT(RHICmdList, RayTracingDynamicGeometryUpdate);

		for (auto& Cmd : *DispatchCommands)
		{
			FRayTracingDynamicGeometryConverterCS* Shader = Cmd.MaterialShader;

			RHICmdList.SetComputeShader(Shader->GetComputeShader());

			Cmd.ShaderBindings.SetOnCommandListForCompute(RHICmdList, Shader-
>GetComputeShader());
			Shader->RWVertexPositions.SetBuffer(RHICmdList, Shader-
>GetComputeShader(), *Cmd.TargetBuffer);
			SetShaderValue(RHICmdList, Shader->GetComputeShader(), Shader-
>NumMaxVertices, Cmd.NumMaxVertices);
			SetShaderValue(RHICmdList, Shader->GetComputeShader(), Shader-
>NumCPUVertices, Cmd.NumCPUVertices);


	RHICmdList.DispatchComputeShader(FMath::DivideAndRoundUp<uint32>(Cmd.NumMaxVertice
s, 256), 1, 1);
			Shader->RWVertexPositions.UnsetUAV(RHICmdList, Shader-
>GetComputeShader());
		}

		TArray<FAccelerationStructureUpdateParams> BuildParams;
```

```
		for (auto& Cmd : *DispatchCommands)
		{
			BuildParams.Add(FAccelerationStructureUpdateParams {
Cmd.TargetGeometry->RayTracingGeometryRHI, Cmd.TargetBuffer->Buffer });
		}

		RHICmdList.BuildAccelerationStructures(BuildParams);

		Clear();
	}
}

void FRayTracingDynamicGeometryCollection::Clear()
{
	DispatchCommands->Empty();
}

#endif // RHI_RAYTRACING
```

4. RayTracingLighting.cpp

```
// Copyright 1998-2019 Epic Games, Inc. All Rights Reserved.

#include "RayTracingLighting.h"
#include "RHI/Public/RHIDefinitions.h"
#include "RendererPrivate.h"

#if RHI_RAYTRACING

IMPLEMENT_GLOBAL_SHADER_PARAMETER_STRUCT(FRaytracingLightDataPacked,
"RaytracingLightsDataPacked");

void SetupRaytracingLightDataPacked(
	const TSparseArray<FLightSceneInfoCompact>& Lights,
	const FViewInfo& View,
	FRaytracingLightDataPacked* LightData)
{
	TMap<UTextureLightProfile*, int32> IESLightProfilesMap;
	TMap<FTextureRHIParamRef, uint32> RectTextureMap;

	LightData->Count = 0;
	LightData->LTCMatTexture = GSystemTextures.LTCMat-
>GetRenderTargetItem().ShaderResourceTexture;
	LightData->LTCMatSampler = TStaticSamplerState<SF_Bilinear, AM_Clamp, AM_Clamp,
AM_Clamp>::GetRHI();
	LightData->LTCAmpTexture = GSystemTextures.LTCAmp-
>GetRenderTargetItem().ShaderResourceTexture;
	LightData->LTCAmpSampler = TStaticSamplerState<SF_Bilinear, AM_Clamp, AM_Clamp,
AM_Clamp>::GetRHI();

	FTextureRHIRef DymmyWhiteTexture = GWhiteTexture->TextureRHI;
	LightData->RectLightTexture0 = DymmyWhiteTexture;
	LightData->RectLightTexture1 = DymmyWhiteTexture;
	LightData->RectLightTexture2 = DymmyWhiteTexture;
	LightData->RectLightTexture3 = DymmyWhiteTexture;
	LightData->RectLightTexture4 = DymmyWhiteTexture;
	LightData->RectLightTexture5 = DymmyWhiteTexture;
	LightData->RectLightTexture6 = DymmyWhiteTexture;
	LightData->RectLightTexture7 = DymmyWhiteTexture;
	static constexpr uint32 MaxRectLightTextureSlos = 8;
	static constexpr uint32 InvalidTextureIndex = 99; // #dxr_todo: share this
definition with ray tracing shaders

	{
```

```
		// IES profiles
		FTextureRHIParamRef IESTextureRHI = nullptr;
		float IESInvProfileCount = 1.0f;

		if (View.IESLightProfileResource && View.IESLightProfileResource->GetIESLightProfilesCount())
		{
			LightData->IESLightProfileTexture = View.IESLightProfileResource->GetTexture();

			uint32 ProfileCount = View.IESLightProfileResource->GetIESLightProfilesCount();
			IESInvProfileCount = ProfileCount ? 1.f / static_cast<float>(ProfileCount) : 0.f;
		}
		else
		{
			LightData->IESLightProfileTexture = GWhiteTexture->TextureRHI;
		}

		LightData->IESLightProfileInvCount = IESInvProfileCount;
		LightData->IESLightProfileTextureSampler = TStaticSamplerState<SF_Bilinear, AM_Clamp, AM_Clamp, AM_Clamp>::GetRHI();
	}

	for (auto Light : Lights)
	{
		const bool bHasStaticLighting = Light.LightSceneInfo->Proxy->HasStaticLighting() && Light.LightSceneInfo->IsPrecomputedLightingValid();
		const bool bAffectReflection = Light.LightSceneInfo->Proxy->AffectReflection();
		if (bHasStaticLighting || !bAffectReflection) continue;

		FLightShaderParameters LightParameters;
		Light.LightSceneInfo->Proxy->GetLightShaderParameters(LightParameters);

		if (Light.LightSceneInfo->Proxy->IsInverseSquared())
		{
			LightParameters.FalloffExponent = 0;
		}

		int32 IESLightProfileIndex = INDEX_NONE;
		if (View.Family->EngineShowFlags.TexturedLightProfiles)
		{
			UTextureLightProfile* IESLightProfileTexture = Light.LightSceneInfo->Proxy->GetIESTexture();
			if (IESLightProfileTexture)
			{
				int32* IndexFound = IESLightProfilesMap.Find(IESLightProfileTexture);
				if (!IndexFound)
				{
					IESLightProfileIndex = IESLightProfilesMap.Add(IESLightProfileTexture, IESLightProfilesMap.Num());
				}
				else
				{
					IESLightProfileIndex = *IndexFound;
				}
			}
		}

		LightData->Type_LightProfileIndex_RectLightTextureIndex[LightData->Count].X
```

```
= Light.LightType;
		LightData->Type_LightProfileIndex_RectLightTextureIndex[LightData->Count].Y
= IESLightProfileIndex;
		LightData->Type_LightProfileIndex_RectLightTextureIndex[LightData->Count].Z
= InvalidTextureIndex;

		LightData->LightPosition_InvRadius[LightData->Count] =
LightParameters.Position;
		LightData->LightPosition_InvRadius[LightData->Count].W =
LightParameters.InvRadius;

		LightData->LightColor_SpecularScale[LightData->Count] =
LightParameters.Color;
		LightData->LightColor_SpecularScale[LightData->Count].W =
LightParameters.SpecularScale;

		LightData->Direction_FalloffExponent[LightData->Count] =
LightParameters.Direction;
		LightData->Direction_FalloffExponent[LightData->Count].W =
LightParameters.FalloffExponent;

		LightData->Tangent_SourceRadius[LightData->Count] =
LightParameters.Tangent;
		LightData->Tangent_SourceRadius[LightData->Count].W =
LightParameters.SourceRadius;

		FVector4 SpotAngles_SourceLength_SoftSourceRadius =
FVector4(LightParameters.SpotAngles, FVector2D(LightParameters.SourceLength,
LightParameters.SoftSourceRadius));
		LightData->SpotAngles_SourceLength_SoftSourceRadius[LightData->Count] =
SpotAngles_SourceLength_SoftSourceRadius;

		const FVector2D FadeParams = Light.LightSceneInfo->Proxy-
>GetDirectionalLightDistanceFadeParameters(View.GetFeatureLevel(), Light.LightSceneInfo-
>IsPrecomputedLightingValid(), View.MaxShadowCascades);

		FVector4 DistanceFadeMAD_RectLightBarnCosAngle_RectLightBarnLength =
FVector4(FadeParams.Y, -FadeParams.X * FadeParams.Y,
LightParameters.RectLightBarnCosAngle, LightParameters.RectLightBarnLength);
		LightData-
>DistanceFadeMAD_RectLightBarnCosAngle_RectLightBarnLength[LightData->Count] =
DistanceFadeMAD_RectLightBarnCosAngle_RectLightBarnLength;

		const bool bRequireTexture = Light.LightType ==
ELightComponentType::LightType_Rect && LightParameters.SourceTexture;
		uint32 RectLightTextureIndex = InvalidTextureIndex;
		if (bRequireTexture)
		{
			const uint32* IndexFound =
RectTextureMap.Find(LightParameters.SourceTexture);
			if (!IndexFound)
			{
				if (RectTextureMap.Num() < MaxRectLightTextureSlos)
				{
					RectLightTextureIndex = RectTextureMap.Num();
					RectTextureMap.Add(LightParameters.SourceTexture,
RectLightTextureIndex);
				}
			}
			else
			{
				RectLightTextureIndex = *IndexFound;
			}
```

```
		}

		if (RectLightTextureIndex != InvalidTextureIndex)
		{
			LightData->Type_LightProfileIndex_RectLightTextureIndex[LightData-
>Count].Z = RectLightTextureIndex;
			switch (RectLightTextureIndex)
			{
			case 0: LightData->RectLightTexture0 = LightParameters.SourceTexture;
break;
			case 1: LightData->RectLightTexture1 = LightParameters.SourceTexture;
break;
			case 2: LightData->RectLightTexture2 = LightParameters.SourceTexture;
break;
			case 3: LightData->RectLightTexture3 = LightParameters.SourceTexture;
break;
			case 4: LightData->RectLightTexture4 = LightParameters.SourceTexture;
break;
			case 5: LightData->RectLightTexture5 = LightParameters.SourceTexture;
break;
			case 6: LightData->RectLightTexture6 = LightParameters.SourceTexture;
break;
			case 7: LightData->RectLightTexture7 = LightParameters.SourceTexture;
break;
			}
		}

		LightData->Count++;

		if (LightData->Count >= GRaytracingLightCountMaximum) break;
	}

	// Update IES light profiles texture
	// TODO (Move to a shared place)
	if (View.IESLightProfileResource != nullptr && IESLightProfilesMap.Num() > 0)
	{
		TArray<UTextureLightProfile*, SceneRenderingAllocator> IESProfilesArray;
		IESProfilesArray.AddUninitialized(IESLightProfilesMap.Num());
		for (auto It = IESLightProfilesMap.CreateIterator(); It; ++It)
		{
			IESProfilesArray[It->Value] = It->Key;
		}

		View.IESLightProfileResource-
>BuildIESLightProfilesTexture(IESProfilesArray);
	}
}

TUniformBufferRef<FRaytracingLightDataPacked> CreateLightDataPackedUniformBuffer(const
TSparseArray<FLightSceneInfoCompact>& Lights, const class FViewInfo& View,
EUniformBufferUsage Usage)
{
	FRaytracingLightDataPacked LightData;
	SetupRaytracingLightDataPacked(Lights, View, &LightData);
	return CreateUniformBufferImmediate(LightData, Usage);
}

#endif // RHI_RAYTRACING
```

5. Material Hit Shaders.cpp

```
#include "RayTracing/RayTracingMaterialHitShaders.h"
#include "DeferredShadingRenderer.h"
#include "PipelineStateCache.h"

#if RHI_RAYTRACING
#include "RayTracingDefinitions.h"
#include "RayTracingInstance.h"

int32 GEnableRayTracingMaterials = 1;
static FAutoConsoleVariableRef CVarEnableRayTracingMaterials(
	TEXT("r.RayTracing.EnableMaterials"),
	GEnableRayTracingMaterials,
	TEXT(" 0: bind default material shader that outputs placeholder data\n")
	TEXT(" 1: bind real material shaders (default)\n"),
	ECVF_RenderThreadSafe
);

static TAutoConsoleVariable<int32> CVarRayTracingTextureLodCvar(
	TEXT("r.RayTracing.UseTextureLod"),
	0,
	TEXT("0 to disable texture LOD.\n")
	TEXT(" 0: off\n")
	TEXT(" 1: on"),
	ECVF_RenderThreadSafe);

static bool IsSupportedVertexFactoryType(const FVertexFactoryType* VertexFactoryType)
{
	static FName LocalVfFname = FName(TEXT("FLocalVertexFactory"), FNAME_Find);
	static FName LSkinnedVfFname = FName(TEXT("FGPUSkinPassthroughVertexFactory"),
FNAME_Find);
	static FName InstancedVfFname = FName(TEXT("FInstancedStaticMeshVertexFactory"),
FNAME_Find);
	static FName NiagaraSpriteVfFname = FName(TEXT("FNiagaraSpriteVertexFactory"),
FNAME_Find);
	static FName GeometryCacheVfFname =
FName(TEXT("FGeometryCacheVertexVertexFactory"), FNAME_Find);

	return (VertexFactoryType == FindVertexFactoryType(LocalVfFname)
		|| VertexFactoryType == FindVertexFactoryType(LSkinnedVfFname)
		|| VertexFactoryType == FindVertexFactoryType(NiagaraSpriteVfFname)
		|| VertexFactoryType == FindVertexFactoryType(GeometryCacheVfFname));
}

class FMaterialCHS : public FMeshMaterialShader, public
FUniformLightMapPolicyShaderParametersType
{
public:
	FMaterialCHS(const FMeshMaterialShaderType::CompiledShaderInitializerType&
Initializer)
		: FMeshMaterialShader(Initializer)
	{
		PassUniformBuffer.Bind(Initializer.ParameterMap,
FSceneTexturesUniformParameters::StaticStructMetadata.GetShaderVariableName());
		FUniformLightMapPolicyShaderParametersType::Bind(Initializer.ParameterMap);
	}

	FMaterialCHS() {}

	virtual bool Serialize(FArchive& Ar) override
```

```
	{
		bool bShaderHasOutdatedParameters = FMeshMaterialShader::Serialize(Ar);
		FUniformLightMapPolicyShaderParametersType::Serialize(Ar);
		return bShaderHasOutdatedParameters;
	}

	void GetShaderBindings(
		const FScene* Scene,
		ERHIFeatureLevel::Type FeatureLevel,
		const FPrimitiveSceneProxy* PrimitiveSceneProxy,
		const FMaterialRenderProxy& MaterialRenderProxy,
		const FMaterial& Material,
		const FMeshPassProcessorRenderState& DrawRenderState,
		const TBasePassShaderElementData<FUniformLightMapPolicy>&
ShaderElementData,
		FMeshDrawSingleShaderBindings& ShaderBindings) const
	{
		FMeshMaterialShader::GetShaderBindings(Scene, FeatureLevel,
PrimitiveSceneProxy, MaterialRenderProxy, Material, DrawRenderState, ShaderElementData,
ShaderBindings);

		FUniformLightMapPolicy::GetPixelShaderBindings(
			PrimitiveSceneProxy,
			ShaderElementData.LightMapPolicyElementData,
			this,
			ShaderBindings);
	}

	void GetElementShaderBindings(
		const FScene* Scene,
		const FSceneView* ViewIfDynamicMeshCommand,
		const FVertexFactory* VertexFactory,
		bool bShaderRequiresPositionOnlyStream,
		ERHIFeatureLevel::Type FeatureLevel,
		const FPrimitiveSceneProxy* PrimitiveSceneProxy,
		const FMeshBatch& MeshBatch,
		const FMeshBatchElement& BatchElement,
		const TBasePassShaderElementData<FUniformLightMapPolicy>&
ShaderElementData,
		FMeshDrawSingleShaderBindings& ShaderBindings,
		FVertexInputStreamArray& VertexStreams) const
	{
		FMeshMaterialShader::GetElementShaderBindings(Scene,
ViewIfDynamicMeshCommand, VertexFactory, bShaderRequiresPositionOnlyStream, FeatureLevel,
PrimitiveSceneProxy, MeshBatch, BatchElement, ShaderElementData, ShaderBindings,
VertexStreams);
	}
};

template<typename LightMapPolicyType, bool UseAnyHitShader, bool UseRayConeTextureLod>
class TMaterialCHS : public FMaterialCHS
{ DECLARE_SHADER_TYPE(TMaterialCHS, MeshMaterial);
public:

	TMaterialCHS(const FMeshMaterialShaderType::CompiledShaderInitializerType&
Initializer)
		: FMaterialCHS(Initializer)
	{}

	TMaterialCHS() {}

	static bool ShouldCompilePermutation(EShaderPlatform Platform, const FMaterial*
Material, const FVertexFactoryType* VertexFactoryType)
	{
```

```
		// #dxr_todo: this should also check if ray tracing is enabled for the
target platform & project
		return IsSupportedVertexFactoryType(VertexFactoryType)
			&& (Material->IsMasked() == UseAnyHitShader)
			&& LightMapPolicyType::ShouldCompilePermutation(Platform, Material,
VertexFactoryType)
			&& ShouldCompileRayTracingShadersForProject(Platform);
	}

	static void ModifyCompilationEnvironment(EShaderPlatform Platform, const
FMaterial* Material, FShaderCompilerEnvironment& OutEnvironment)
	{
		OutEnvironment.SetDefine(TEXT("USE_RAYTRACED_TEXTURE_RAYCONE_LOD"),
UseRayConeTextureLod ? 1 : 0);
		OutEnvironment.SetDefine(TEXT("SCENE_TEXTURES_DISABLED"), 1);
		LightMapPolicyType::ModifyCompilationEnvironment(Platform, Material,
OutEnvironment);
		FMeshMaterialShader::ModifyCompilationEnvironment(Platform, Material,
OutEnvironment);
	}

	static bool ValidateCompiledResult(EShaderPlatform Platform, const
TArray<FMaterial*>& Materials, const FVertexFactoryType* VertexFactoryType, const
FShaderParameterMap& ParameterMap, TArray<FString>& OutError)
	{
		if
(ParameterMap.ContainsParameterAllocation(FSceneTexturesUniformParameters::StaticStructMe
tadata.GetShaderVariableName()))
		{
			OutError.Add(TEXT("Ray tracing closest hit shaders cannot read from
the SceneTexturesStruct."));
			return false;
		}

		return true;
	}
};


#define IMPLEMENT_MATERIALCHS_TYPE(LightMapPolicyType, LightMapPolicyName,
AnyHitShaderName) \
	typedef TMaterialCHS<LightMapPolicyType, false, false>
TMaterialCHS##LightMapPolicyName; \
	IMPLEMENT_MATERIAL_SHADER_TYPE(template<>, TMaterialCHS##LightMapPolicyName,
TEXT("/Engine/Private/RayTracing/RayTracingMaterialHitShaders.usf"),
TEXT("closesthit=MaterialCHS"), SF_RayHitGroup); \
	typedef TMaterialCHS<LightMapPolicyType, true, false>
TMaterialCHS##LightMapPolicyName##AnyHitShaderName; \
	IMPLEMENT_MATERIAL_SHADER_TYPE(template<>,
TMaterialCHS##LightMapPolicyName##AnyHitShaderName,
TEXT("/Engine/Private/RayTracing/RayTracingMaterialHitShaders.usf"),
TEXT("closesthit=MaterialCHS anyhit=MaterialAHS"), SF_RayHitGroup) \
	typedef TMaterialCHS<LightMapPolicyType, false, true>
TMaterialCHSLod##LightMapPolicyName; \
	IMPLEMENT_MATERIAL_SHADER_TYPE(template<>, TMaterialCHSLod##LightMapPolicyName,
TEXT("/Engine/Private/RayTracing/RayTracingMaterialHitShaders.usf"),
TEXT("closesthit=MaterialCHS"), SF_RayHitGroup); \
	typedef TMaterialCHS<LightMapPolicyType, true, true>
TMaterialCHSLod##LightMapPolicyName##AnyHitShaderName; \
	IMPLEMENT_MATERIAL_SHADER_TYPE(template<>,
TMaterialCHSLod##LightMapPolicyName##AnyHitShaderName,
TEXT("/Engine/Private/RayTracing/RayTracingMaterialHitShaders.usf"),
TEXT("closesthit=MaterialCHS anyhit=MaterialAHS"), SF_RayHitGroup);
```

```
IMPLEMENT_MATERIALCHS_TYPE(TUniformLightMapPolicy<LMP_NO_LIGHTMAP>, FNoLightMapPolicy,
FAnyHitShader);
IMPLEMENT_MATERIALCHS_TYPE(TUniformLightMapPolicy<LMP_PRECOMPUTED_IRRADIANCE_VOLUME_INDIR
ECT_LIGHTING>, FPrecomputedVolumetricLightmapLightingPolicy, FAnyHitShader);
IMPLEMENT_MATERIALCHS_TYPE(TUniformLightMapPolicy<LMP_LQ_LIGHTMAP>, TLightMapPolicyLQ,
FAnyHitShader);
IMPLEMENT_MATERIALCHS_TYPE(TUniformLightMapPolicy<LMP_HQ_LIGHTMAP>, TLightMapPolicyHQ,
FAnyHitShader);
IMPLEMENT_MATERIALCHS_TYPE(TUniformLightMapPolicy<LMP_DISTANCE_FIELD_SHADOWS_AND_HQ_LIGHT
MAP>, TDistanceFieldShadowsAndLightMapPolicyHQ, FAnyHitShader);

IMPLEMENT_GLOBAL_SHADER(FHiddenMaterialHitGroup,
"/Engine/Private/RayTracing/RayTracingMaterialDefaultHitShaders.usf",
"closesthit=HiddenMaterialCHS anyhit=HiddenMaterialAHS", SF_RayHitGroup);
IMPLEMENT_GLOBAL_SHADER(FOpaqueShadowHitGroup,
"/Engine/Private/RayTracing/RayTracingMaterialDefaultHitShaders.usf",
"closesthit=OpaqueShadowCHS", SF_RayHitGroup);
IMPLEMENT_GLOBAL_SHADER(FDefaultMaterialMS,
"/Engine/Private/RayTracing/RayTracingMaterialDefaultHitShaders.usf",
"DefaultMaterialMS", SF_RayMiss);

template<typename LightMapPolicyType>
static FMaterialCHS* GetMaterialHitShader(const FMaterial& RESTRICT MaterialResource,
const FVertexFactory* VertexFactory, bool UseTextureLod)
{
	if (MaterialResource.IsMasked())
	{
		if(UseTextureLod)
		{
			return MaterialResource.GetShader<TMaterialCHS<LightMapPolicyType,
true, true>>(VertexFactory->GetType());
		}
		else
		{
			return MaterialResource.GetShader<TMaterialCHS<LightMapPolicyType,
true, false>>(VertexFactory->GetType());
		}
	}
	else
	{
		if (UseTextureLod)
		{
			return MaterialResource.GetShader<TMaterialCHS<LightMapPolicyType,
false, true>>(VertexFactory->GetType());
		}
		else
		{
			return MaterialResource.GetShader<TMaterialCHS<LightMapPolicyType,
false, false>>(VertexFactory->GetType());
		}
	}
}

template<typename PassShadersType, typename ShaderElementDataType>
void FRayTracingMeshProcessor::BuildRayTracingMeshCommands(
	const FMeshBatch& RESTRICT MeshBatch,
	uint64 BatchElementMask,
	const FPrimitiveSceneProxy* RESTRICT PrimitiveSceneProxy,
	const FMaterialRenderProxy& RESTRICT MaterialRenderProxy,
	const FMaterial& RESTRICT MaterialResource,
	const FMeshPassProcessorRenderState& RESTRICT DrawRenderState,
	PassShadersType PassShaders,
	const ShaderElementDataType& ShaderElementData)
```

```
{
	const FVertexFactory* RESTRICT VertexFactory = MeshBatch.VertexFactory;

	checkf(MaterialRenderProxy.ImmutableSamplerState.ImmutableSamplers[0] == nullptr,
TEXT("Immutable samplers not yet supported in Mesh Draw Command pipeline"));

	FRayTracingMeshCommand SharedCommand;

	SharedCommand.SetShaders(PassShaders.GetUntypedShaders());
	SharedCommand.InstanceMask =
ComputeBlendModeMask(MaterialResource.GetBlendMode());
	SharedCommand.bCastRayTracedShadows = MeshBatch.CastRayTracedShadow;
	SharedCommand.bOpaque = MaterialResource.GetBlendMode() ==
EBlendMode::BLEND_Opaque;

	FVertexInputStreamArray VertexStreams;
	VertexFactory->GetStreams(ERHIFeatureLevel::SM5, VertexStreams);

	if (PassShaders.RayHitGroupShader)
	{
		FMeshDrawSingleShaderBindings ShaderBindings =
SharedCommand.ShaderBindings.GetSingleShaderBindings(SF_RayHitGroup);
		PassShaders.RayHitGroupShader->GetShaderBindings(Scene, FeatureLevel,
PrimitiveSceneProxy, MaterialRenderProxy, MaterialResource, DrawRenderState,
ShaderElementData, ShaderBindings);
	}

	const int32 NumElements = MeshBatch.Elements.Num();

	for (int32 BatchElementIndex = 0; BatchElementIndex < NumElements;
BatchElementIndex++)
	{
		if ((1ull << BatchElementIndex) & BatchElementMask)
		{
			const FMeshBatchElement& BatchElement =
MeshBatch.Elements[BatchElementIndex];
			FRayTracingMeshCommand& RayTracingMeshCommand = CommandContext-
>AddCommand(SharedCommand);

			if (PassShaders.RayHitGroupShader)
			{
				FMeshDrawSingleShaderBindings RayHitGroupShaderBindings =
RayTracingMeshCommand.ShaderBindings.GetSingleShaderBindings(SF_RayHitGroup);
				PassShaders.RayHitGroupShader->GetElementShaderBindings(Scene,
ViewIfDynamicMeshCommand, VertexFactory, false, FeatureLevel, PrimitiveSceneProxy,
MeshBatch, BatchElement, ShaderElementData, RayHitGroupShaderBindings, VertexStreams);
			}

			int32 GeometrySegmentIndex = MeshBatch.SegmentIndex +
BatchElementIndex;
			RayTracingMeshCommand.GeometrySegmentIndex = (GeometrySegmentIndex <
UINT8_MAX) ? uint8(GeometrySegmentIndex) : UINT8_MAX;

			CommandContext->FinalizeCommand(RayTracingMeshCommand);
		}
	}
}

void FRayTracingMeshProcessor::Process(
	const FMeshBatch& RESTRICT MeshBatch,
	uint64 BatchElementMask,
	const FPrimitiveSceneProxy* RESTRICT PrimitiveSceneProxy,
	const FMaterialRenderProxy& RESTRICT MaterialRenderProxy,
```

```
		const FMaterial& RESTRICT MaterialResource,
		EMaterialShadingModel ShadingModel,
		const FUniformLightMapPolicy& RESTRICT LightMapPolicy,
		const		typename		FUniformLightMapPolicy::ElementDataType&		RESTRICT
LightMapElementData)
{
		const FVertexFactory* VertexFactory = MeshBatch.VertexFactory;

		TMeshProcessorShaders<
			FMeshMaterialShader,
			FMeshMaterialShader,
			FMeshMaterialShader,
			FMeshMaterialShader,
			FMeshMaterialShader,
			FMaterialCHS> RayTracingShaders;

		switch (LightMapPolicy.GetIndirectPolicy())
		{
		case LMP_PRECOMPUTED_IRRADIANCE_VOLUME_INDIRECT_LIGHTING:
			RayTracingShaders.RayHitGroupShader =
GetMaterialHitShader<TUniformLightMapPolicy<LMP_PRECOMPUTED_IRRADIANCE_VOLUME_INDIRECT_LI
GHTING>>(MaterialResource, VertexFactory,
CVarRayTracingTextureLodCvar.GetValueOnRenderThread() != 0);
			break;
		case LMP_LQ_LIGHTMAP:
			RayTracingShaders.RayHitGroupShader =
GetMaterialHitShader<TUniformLightMapPolicy<LMP_LQ_LIGHTMAP>>(MaterialResource,
VertexFactory, CVarRayTracingTextureLodCvar.GetValueOnRenderThread() != 0);
			break;
		case LMP_HQ_LIGHTMAP:
			RayTracingShaders.RayHitGroupShader =
GetMaterialHitShader<TUniformLightMapPolicy<LMP_HQ_LIGHTMAP>>(MaterialResource,
VertexFactory, CVarRayTracingTextureLodCvar.GetValueOnRenderThread() != 0);
			break;
		case LMP_DISTANCE_FIELD_SHADOWS_AND_HQ_LIGHTMAP:
			RayTracingShaders.RayHitGroupShader =
GetMaterialHitShader<TUniformLightMapPolicy<LMP_DISTANCE_FIELD_SHADOWS_AND_HQ_LIGHTMAP>>(
MaterialResource, VertexFactory, CVarRayTracingTextureLodCvar.GetValueOnRenderThread() !=
0);
			break;
		case LMP_NO_LIGHTMAP:
			RayTracingShaders.RayHitGroupShader =
GetMaterialHitShader<TUniformLightMapPolicy<LMP_NO_LIGHTMAP>>(MaterialResource,
VertexFactory, CVarRayTracingTextureLodCvar.GetValueOnRenderThread() != 0);
			break;
		default:
			check(false);
		}

		FMeshPassProcessorRenderState PassDrawRenderState(Scene-
>UniformBuffers.ViewUniformBuffer, Scene->UniformBuffers.OpaqueBasePassUniformBuffer);
		PassDrawRenderState.SetBlendState(TStaticBlendState<CW_RGBA, BO_Add, BF_One,
BF_One, BO_Add, BF_Zero, BF_One>::GetRHI());
		PassDrawRenderState.SetDepthStencilState(TStaticDepthStencilState<false,
CF_DepthNearOrEqual>::GetRHI());

		TBasePassShaderElementData<FUniformLightMapPolicy>
ShaderElementData(LightMapElementData);
		ShaderElementData.InitializeMeshMaterialData(ViewIfDynamicMeshCommand,
PrimitiveSceneProxy, MeshBatch, -1, true);

		BuildRayTracingMeshCommands(
```

```
                MeshBatch,
                BatchElementMask,
                PrimitiveSceneProxy,
                MaterialRenderProxy,
                MaterialResource,
                PassDrawRenderState,
                RayTracingShaders,
                ShaderElementData);
}


void FRayTracingMeshProcessor::AddMeshBatch(const FMeshBatch& RESTRICT MeshBatch, uint64
BatchElementMask, const FPrimitiveSceneProxy* RESTRICT PrimitiveSceneProxy)
{
        // Caveat: there are also branches not emitting any MDC
        if (MeshBatch.bUseForMaterial &&
IsSupportedVertexFactoryType(MeshBatch.VertexFactory->GetType()))
        {
                // Determine the mesh's material and blend mode.
                const FMaterialRenderProxy* FallbackMaterialRenderProxyPtr = nullptr;
                const FMaterial& Material = MeshBatch.MaterialRenderProxy-
>GetMaterialWithFallback(FeatureLevel, FallbackMaterialRenderProxyPtr);

                const FMaterialRenderProxy& MaterialRenderProxy =
FallbackMaterialRenderProxyPtr ? *FallbackMaterialRenderProxyPtr :
*MeshBatch.MaterialRenderProxy;

                const EBlendMode BlendMode = Material.GetBlendMode();
                const EMaterialShadingModel ShadingModel = Material.GetShadingModel();
                const bool bIsTranslucent = IsTranslucentBlendMode(BlendMode);

                // Only draw opaque materials.
                if ((!PrimitiveSceneProxy || PrimitiveSceneProxy->ShouldRenderInMainPass())
                        && ShouldIncludeDomainInMeshPass(Material.GetMaterialDomain()))
                {
                        // Check for a cached light-map.
                        const bool bIsLitMaterial = (ShadingModel != MSM_Unlit);
                        static const auto AllowStaticLightingVar =
IConsoleManager::Get().FindTConsoleVariableDataInt(TEXT("r.AllowStaticLighting"));
                        const bool bAllowStaticLighting = (!AllowStaticLightingVar ||
AllowStaticLightingVar->GetValueOnRenderThread() != 0);

                        const FLightMapInteraction LightMapInteraction =
(bAllowStaticLighting && MeshBatch.LCI && bIsLitMaterial)
                                ? MeshBatch.LCI->GetLightMapInteraction(FeatureLevel)
                                : FLightMapInteraction();

                        // force LQ lightmaps based on system settings
                        const bool bPlatformAllowsHighQualityLightMaps =
AllowHighQualityLightmaps(FeatureLevel);
                        const bool bAllowHighQualityLightMaps =
bPlatformAllowsHighQualityLightMaps && LightMapInteraction.AllowsHighQualityLightmaps();

                        const bool bAllowIndirectLightingCache = Scene && Scene-
>PrecomputedLightVolumes.Num() > 0;
                        const bool bUseVolumetricLightmap = Scene && Scene-
>VolumetricLightmapSceneData.HasData();

                        {
                                static const auto CVarSupportLowQualityLightmap =
IConsoleManager::Get().FindTConsoleVariableDataInt(TEXT("r.SupportLowQualityLightmaps"));
                                const bool bAllowLowQualityLightMaps =
(!CVarSupportLowQualityLightmap) || (CVarSupportLowQualityLightmap->GetValueOnAnyThread()
```

```
 != 0);

                              switch (LightMapInteraction.GetType())
                              {
                              case LMIT_Texture:
                                    if (bAllowHighQualityLightMaps)
                                    {
                                          const FShadowMapInteraction ShadowMapInteraction
= (bAllowStaticLighting && MeshBatch.LCI && bIsLitMaterial)
                                                ? MeshBatch.LCI->GetShadowMapInteraction()
                                                : FShadowMapInteraction();

                                          if (ShadowMapInteraction.GetType() ==
SMIT_Texture)
                                          { Process(
                                                      MeshBatch,
                                                      BatchElementMask,
                                                      PrimitiveSceneProxy,
                                                      MaterialRenderProxy,
                                                      Material,
                                                      ShadingModel,

      FUniformLightMapPolicy(LMP_DISTANCE_FIELD_SHADOWS_AND_HQ_LIGHTMAP),
                                                      MeshBatch.LCI);
                                          }
                                          else
                                          {
                                                Process(
                                                      MeshBatch,
                                                      BatchElementMask,
                                                      PrimitiveSceneProxy,
                                                      MaterialRenderProxy,
                                                      Material,
                                                      ShadingModel,

      FUniformLightMapPolicy(LMP_HQ_LIGHTMAP),
                                                      MeshBatch.LCI);
                                          }
                                    }
                                    else if (bAllowLowQualityLightMaps)
                                    {
                                          Process(
                                                MeshBatch,
                                                BatchElementMask,
                                                PrimitiveSceneProxy,
                                                MaterialRenderProxy,
                                                Material,
                                                ShadingModel,
                                                FUniformLightMapPolicy(LMP_LQ_LIGHTMAP),
                                                MeshBatch.LCI);
                                    }
                                    else
                                    {
                                          Process(
                                                MeshBatch,
                                                BatchElementMask,
                                                PrimitiveSceneProxy,
                                                MaterialRenderProxy,
                                                Material,
                                                ShadingModel,
                                                FUniformLightMapPolicy(LMP_NO_LIGHTMAP),
                                                MeshBatch.LCI);
                                    }
```

```
                                        break;
                                default:
                                        if (bIsLitMaterial
                                                && bAllowStaticLighting
                                                && Scene
                                                && Scene->VolumetricLightmapSceneData.HasData()
                                                && PrimitiveSceneProxy
                                                && (PrimitiveSceneProxy->IsMovable()
                                                        || PrimitiveSceneProxy-
>NeedsUnbuiltPreviewLighting()
                                                        || PrimitiveSceneProxy->GetLightmapType()
== ELightmapType::ForceVolumetric))
                                        {
                                                Process(
                                                        MeshBatch,
                                                        BatchElementMask,
                                                        PrimitiveSceneProxy,
                                                        MaterialRenderProxy,
                                                        Material,
                                                        ShadingModel,

        FUniformLightMapPolicy(LMP_PRECOMPUTED_IRRADIANCE_VOLUME_INDIRECT_LIGHTING),
                                                        MeshBatch.LCI);
                                        }
                                        else
                                        {
                                                Process(
                                                        MeshBatch,
                                                        BatchElementMask,
                                                        PrimitiveSceneProxy,
                                                        MaterialRenderProxy,
                                                        Material,
                                                        ShadingModel,
                                                        FUniformLightMapPolicy(LMP_NO_LIGHTMAP),
                                                        MeshBatch.LCI);
                                        }
                                        break;
                                };
                        }
                }
        }
}

FRHIRayTracingPipelineState*
FDeferredShadingSceneRenderer::BindRayTracingMaterialPipeline(
        FRHICommandList& RHICmdList,
        const FViewInfo& View,
        const TArrayView<const FRayTracingShaderRHIParamRef>& RayGenShaderTable,
        FRayTracingShaderRHIParamRef MissShader,
        FRayTracingShaderRHIParamRef DefaultClosestHitShader
)
{
        SCOPE_CYCLE_COUNTER(STAT_BindRayTracingPipeline);

        FRHIRayTracingPipelineState* PipelineState = nullptr;

        FRayTracingPipelineStateInitializer Initializer;

        Initializer.MaxPayloadSizeInBytes = 52; //
sizeof(FPackedMaterialClosestHitPayload)
        Initializer.bAllowHitGroupIndexing = true;

        Initializer.SetRayGenShaderTable(RayGenShaderTable);
```

```
	FRayTracingShaderRHIParamRef MissShaderTable[] = { MissShader };
	Initializer.SetMissShaderTable(MissShaderTable);

	const bool bEnableMaterials = GEnableRayTracingMaterials != 0;

	TArray<FRayTracingShaderRHIParamRef> RayTracingMaterialLibrary;

	if (bEnableMaterials)
	{
		FShaderResource::GetRayTracingMaterialLibrary(RayTracingMaterialLibrary,
DefaultClosestHitShader);
	}
	else
	{
		RayTracingMaterialLibrary.Add(DefaultClosestHitShader);
	}

	int32 OpaqueShadowMaterialIndex = RayTracingMaterialLibrary.Add(View.ShaderMap-
>GetShader<FOpaqueShadowHitGroup>()->GetRayTracingShader());
	int32 HiddenMaterialIndex = RayTracingMaterialLibrary.Add(View.ShaderMap-
>GetShader<FHiddenMaterialHitGroup>()->GetRayTracingShader());

	Initializer.SetHitGroupTable(RayTracingMaterialLibrary);

	PipelineState =
PipelineStateCache::GetAndOrCreateRayTracingPipelineState(Initializer);

	const FViewInfo& ReferenceView = Views[0];

	static auto CVarEnableShadowMaterials =
IConsoleManager::Get().FindConsoleVariable(TEXT("r.RayTracing.Shadows.EnableMaterials"));
	bool bEnableShadowMaterials = CVarEnableShadowMaterials ?
CVarEnableShadowMaterials->GetInt() != 0 : true;

	for (const FVisibleRayTracingMeshCommand VisibleMeshCommand :
ReferenceView.VisibleRayTracingMeshCommands)
	{
		const FRayTracingMeshCommand& MeshCommand =
*VisibleMeshCommand.RayTracingMeshCommand;

		const uint32 HitGroupIndex = bEnableMaterials
			? MeshCommand.MaterialShaderIndex
			: 0; // Force the same shader to be used on all geometry

		// Bind primary material shader


	MeshCommand.ShaderBindings.SetRayTracingShaderBindingsForHitGroup(RHICmdList,
			View.RayTracingScene.RayTracingSceneRHI,
			VisibleMeshCommand.InstanceIndex,
			MeshCommand.GeometrySegmentIndex,
			PipelineState,
			HitGroupIndex,
			RAY_TRACING_SHADER_SLOT_MATERIAL);

		// Bind shadow shader

		if (MeshCommand.bCastRayTracedShadows)
		{
			if (MeshCommand.bOpaque || !bEnableShadowMaterials)
			{
				// Fully opaque surfaces don't need the full material, so we
bind a specialized shader that simply updates HitT.
```

```
		RHICmdList.SetRayTracingHitGroup(View.RayTracingScene.RayTracingSceneRHI,
						VisibleMeshCommand.InstanceIndex,
						MeshCommand.GeometrySegmentIndex,
						RAY_TRACING_SHADER_SLOT_SHADOW,

						PipelineState, OpaqueShadowMaterialIndex,
						0, nullptr, 0);
			}
			else
			{
				// Masked materials require full material evaluation with any-
hit shader.
				// #dxr_todo: we need to generate a shadow-specific closest
hit shader for this!

		MeshCommand.ShaderBindings.SetRayTracingShaderBindingsForHitGroup(RHICmdList,
						View.RayTracingScene.RayTracingSceneRHI,
						VisibleMeshCommand.InstanceIndex,
						MeshCommand.GeometrySegmentIndex,
						PipelineState,
						HitGroupIndex,
						RAY_TRACING_SHADER_SLOT_SHADOW);
			}
		}
		else
		{

		RHICmdList.SetRayTracingHitGroup(View.RayTracingScene.RayTracingSceneRHI,
				VisibleMeshCommand.InstanceIndex,
				MeshCommand.GeometrySegmentIndex,
				RAY_TRACING_SHADER_SLOT_SHADOW,
				PipelineState, HiddenMaterialIndex,
				0, nullptr, 0);
		}
	}

	return PipelineState;
}

#endif // RHI_RAYTRACING
```

6. RayTracingReflections.cpp

```
#include "RendererPrivate.h"
#include "GlobalShader.h"
#include "DeferredShadingRenderer.h"

#if RHI_RAYTRACING

#include "ClearQuad.h"
#include "SceneRendering.h"
#include "SceneRenderTargets.h"
#include "RenderTargetPool.h"
#include "RHIResources.h"
#include "UniformBuffer.h"
#include "VisualizeTexture.h"
#include "LightRendering.h"
#include "SystemTextures.h"
#include "RayTracing/RaytracingOptions.h"
#include "RayTracing/RayTracingDeferredMaterials.h"
```

```
#include "RayTracing/RayTracingLighting.h"
#include "RenderGraph.h"
#include "RayTracing/RayTracingLighting.h"

static float GRayTracingReflectionsMaxRoughness = -1;
static FAutoConsoleVariableRef CVarRayTracingReflectionsMaxRoughness(
	TEXT("r.RayTracing.Reflections.MaxRoughness"),

	GRayTracingReflectionsMaxRoughness,
	TEXT("Sets the maximum roughness until which ray tracing reflections will be
visible (default = -1 (max roughness driven by postprocessing volume))")
);

static int32 GRayTracingReflectionsMaxBounces = -1;
static FAutoConsoleVariableRef CVarRayTracingReflectionsMaxBounces(
	TEXT("r.RayTracing.Reflections.MaxBounces"),
	GRayTracingReflectionsMaxBounces,
	TEXT("Sets the maximum number of ray tracing reflection bounces (default = -1 (max
bounces driven by postprocessing volume))")
);

static int32 GRayTracingReflectionsEmissiveAndIndirectLighting = 1;
static FAutoConsoleVariableRef CVarRayTracingReflectionsEmissiveAndIndirectLighting(
	TEXT("r.RayTracing.Reflections.EmissiveAndIndirectLighting"),
	GRayTracingReflectionsEmissiveAndIndirectLighting,
	TEXT("Enables ray tracing reflections emissive and indirect lighting (default =
1)")
);

static int32 GRayTracingReflectionsDirectLighting = 1;
static FAutoConsoleVariableRef CVarRayTracingReflectionsDirectLighting(
	TEXT("r.RayTracing.Reflections.DirectLighting"),
	GRayTracingReflectionsDirectLighting,
	TEXT("Enables ray tracing reflections direct lighting (default = 1)")
);

static int32 GRayTracingReflectionsShadows = -1;
static FAutoConsoleVariableRef CVarRayTracingReflectionsShadows(
	TEXT("r.RayTracing.Reflections.Shadows"),
	GRayTracingReflectionsShadows,
	TEXT("Enables shadows in ray tracing reflections)")
	TEXT(" -1: Shadows driven by postprocessing volume (default)")
	TEXT(" 0: Shadows disabled ")
	TEXT(" 1: Hard shadows")
	TEXT(" 2: Soft area shadows")
);

static float GRayTracingReflectionsMinRayDistance = -1;
static FAutoConsoleVariableRef CVarRayTracingReflectionsMinRayDistance(
	TEXT("r.RayTracing.Reflections.MinRayDistance"),
	GRayTracingReflectionsMinRayDistance,
	TEXT("Sets the minimum ray distance for ray traced reflection rays. Actual
reflection ray length is computed as Lerp(MaxRayDistance, MinRayDistance, Roughness),
i.e. reflection rays become shorter when traced from rougher surfaces. (default = -1
(infinite rays))")
);

static float GRayTracingReflectionsMaxRayDistance = -1;
static FAutoConsoleVariableRef CVarRayTracingReflectionsMaxRayDistance(
	TEXT("r.RayTracing.Reflections.MaxRayDistance"),
	GRayTracingReflectionsMaxRayDistance,
	TEXT("Sets the maximum ray distance for ray traced reflection rays. When ray
shortening is used, skybox will not be sampled in RT reflection pass and will be
```

```
composited later, together with local reflection captures. Negative values turn off this
optimization. (default = -1 (infinite rays))")
);

static TAutoConsoleVariable<int32> CVarRayTracingReflectionsSortMaterials(
	TEXT("r.RayTracing.Reflections.SortMaterials"),
	0,
	TEXT("Sets whether refected materials will be sorted before shading\n")
	TEXT("0: Disabled (Default)\n ")

	TEXT("1: Enabled, using Trace->Sort->Trace\n"),
	ECVF_RenderThreadSafe);

static TAutoConsoleVariable<int32> CVarRayTracingReflectionsSortTileSize(
	TEXT("r.RayTracing.Reflections.SortTileSize"),
	64,
	TEXT("Size of pixel tiles for sorted reflections\n")
	TEXT("  Default 64\n"),
	ECVF_RenderThreadSafe);

static TAutoConsoleVariable<int32> CVarRayTracingReflectionsSortSize(
	TEXT("r.RayTracing.Reflections.SortSize"),
	5,
	TEXT("Size of horizon for material ID sort\n")
	TEXT("0: Disabled\n")
	TEXT("1: 256 Elements\n")
	TEXT("2: 512 Elements\n")
	TEXT("3: 1024 Elements\n")
	TEXT("4: 2048 Elements\n")
	TEXT("5: 4096 Elements (Default)\n"),
	ECVF_RenderThreadSafe);

static const int32 GReflectionLightCountMaximum = 64;

class FRayTracingReflectionsRGS : public FGlobalShader
{
	DECLARE_GLOBAL_SHADER(FRayTracingReflectionsRGS)
	SHADER_USE_ROOT_PARAMETER_STRUCT(FRayTracingReflectionsRGS, FGlobalShader)

	class FDenoiserOutput : SHADER_PERMUTATION_BOOL("DIM_DENOISER_OUTPUT");

	class FDeferredMaterialMode :
SHADER_PERMUTATION_ENUM_CLASS("DIM_DEFERRED_MATERIAL_MODE", EDeferredMaterialMode);

	using FPermutationDomain = TShaderPermutationDomain<FDenoiserOutput,
FDeferredMaterialMode>;

	BEGIN_SHADER_PARAMETER_STRUCT(FParameters, )
		SHADER_PARAMETER(int32, SamplesPerPixel)
		SHADER_PARAMETER(int32, MaxBounces)
		SHADER_PARAMETER(int32, HeightFog)
		SHADER_PARAMETER(int32, ShouldDoDirectLighting)
		SHADER_PARAMETER(int32, ReflectedShadowsType)
		SHADER_PARAMETER(int32, ShouldDoEmissiveAndIndirectLighting)
		SHADER_PARAMETER(int32, UpscaleFactor)
		SHADER_PARAMETER(int32, SortTileSize)
		SHADER_PARAMETER(FIntPoint, RayTracingResolution)
		SHADER_PARAMETER(FIntPoint, TileAlignedResolution)
		SHADER_PARAMETER(float, ReflectionMinRayDistance)
		SHADER_PARAMETER(float, ReflectionMaxRayDistance)
		SHADER_PARAMETER(float, ReflectionMaxRoughness)
		SHADER_PARAMETER(float, ReflectionMaxNormalBias)
```

```
		SHADER_PARAMETER_SRV(RaytracingAccelerationStructure, TLAS)

		SHADER_PARAMETER_STRUCT_REF(FViewUniformShaderParameters,
ViewUniformBuffer)
		SHADER_PARAMETER_STRUCT_REF(FSceneTexturesUniformParameters,
SceneTexturesStruct)
		SHADER_PARAMETER_STRUCT_REF(FRaytracingLightDataPacked, LightDataPacked)
		SHADER_PARAMETER_STRUCT_REF(FReflectionUniformParameters, ReflectionStruct)
		SHADER_PARAMETER_STRUCT_REF(FFogUniformParameters, FogUniformParameters)

		// Optional indirection buffer used for sorted materials
		SHADER_PARAMETER_RDG_BUFFER_UAV(StructuredBuffer<FDeferredMaterialPayload>,
MaterialBuffer)

		SHADER_PARAMETER_RDG_TEXTURE_UAV(RWTexture2D<float4>, ColorOutput)
		SHADER_PARAMETER_RDG_TEXTURE_UAV(RWTexture2D<float>, RayHitDistanceOutput)
		SHADER_PARAMETER_RDG_TEXTURE_UAV(RWTexture2D<float>,
RayImaginaryDepthOutput)
	END_SHADER_PARAMETER_STRUCT()

	static bool ShouldCompilePermutation(const FGlobalShaderPermutationParameters&
Parameters)
	{
		return ShouldCompileRayTracingShadersForProject(Parameters.Platform);
	}
};

class FRayTracingReflectionsCHS : public FGlobalShader
{
	DECLARE_GLOBAL_SHADER(FRayTracingReflectionsCHS);
	SHADER_USE_ROOT_PARAMETER_STRUCT(FRayTracingReflectionsCHS, FGlobalShader)

	static bool ShouldCompilePermutation(const FGlobalShaderPermutationParameters&
Parameters)
	{
		return ShouldCompileRayTracingShadersForProject(Parameters.Platform);
	}

	using FParameters = FEmptyShaderParameters;
};

class FRayTracingReflectionsMS : public FGlobalShader
{
	DECLARE_GLOBAL_SHADER(FRayTracingReflectionsMS);
	SHADER_USE_ROOT_PARAMETER_STRUCT(FRayTracingReflectionsMS, FGlobalShader)

	static bool ShouldCompilePermutation(const FGlobalShaderPermutationParameters&
Parameters)
	{
		return ShouldCompileRayTracingShadersForProject(Parameters.Platform);
	}

	using FParameters = FEmptyShaderParameters;
};

IMPLEMENT_GLOBAL_SHADER(FRayTracingReflectionsRGS,
"/Engine/Private/RayTracing/RayTracingReflections.usf", "RayTracingReflectionsRGS",
SF_RayGen);
IMPLEMENT_GLOBAL_SHADER(FRayTracingReflectionsCHS,
"/Engine/Private/RayTracing/RayTracingReflections.usf", "RayTracingReflectionsMainCHS",
SF_RayHitGroup);
IMPLEMENT_GLOBAL_SHADER(FRayTracingReflectionsMS,
"/Engine/Private/RayTracing/RayTracingReflections.usf", "RayTracingReflectionsMainMS",
```

```
SF_RayMiss);

void FDeferredShadingSceneRenderer::PrepareRayTracingReflections(const FViewInfo& View,
TArray<FRayTracingShaderRHIParamRef>& OutRayGenShaders)
{
	// Declare all RayGen shaders that require material closest hit shaders to be
bound

	const bool bSortMaterials =
CVarRayTracingReflectionsSortMaterials.GetValueOnRenderThread() != 0;

	const EDeferredMaterialMode DeferredMaterialMode = bSortMaterials ?
EDeferredMaterialMode::Shade : EDeferredMaterialMode::None;

	FRayTracingReflectionsRGS::FPermutationDomain PermutationVector;
	PermutationVector.Set<FRayTracingReflectionsRGS::FDeferredMaterialMode>(DeferredMa
terialMode);

	auto RayGenShader = View.ShaderMap-
>GetShader<FRayTracingReflectionsRGS>(PermutationVector);

	OutRayGenShaders.Add(RayGenShader->GetRayTracingShader());
}
#endif // RHI_RAYTRACING

void FDeferredShadingSceneRenderer::RenderRayTracingReflections(
	FRDGBuilder& GraphBuilder,
	const FViewInfo& View,
	FRDGTextureRef* OutColorTexture,
	FRDGTextureRef* OutRayHitDistanceTexture,
	FRDGTextureRef* OutRayImaginaryDepthTexture,
	int32 SamplePerPixel,
	int32 HeightFog,
	float ResolutionFraction)
#if RHI_RAYTRACING
{
	const uint32 SortTileSize =
CVarRayTracingReflectionsSortTileSize.GetValueOnRenderThread();
	const bool bSortMaterials =
CVarRayTracingReflectionsSortMaterials.GetValueOnRenderThread() != 0;

	FSceneRenderTargets& SceneContext =
FSceneRenderTargets::Get(GraphBuilder.RHICmdList);

	int32 UpscaleFactor = int32(1.0f / ResolutionFraction);
	ensure(ResolutionFraction == 1.0 / UpscaleFactor);
	ensureMsgf(FComputeShaderUtils::kGolden2DGroupSize % UpscaleFactor == 0,
TEXT("Reflection ray tracing will have uv misalignement."));
	FIntPoint RayTracingResolution = FIntPoint::DivideAndRoundUp(View.ViewRect.Size(),
UpscaleFactor);

	{
		FPooledRenderTargetDesc Desc = SceneContext.GetSceneColor()->GetDesc();
		Desc.Format = PF_FloatRGBA;
		Desc.Flags &= ~(TexCreate_FastVRAM | TexCreate_Transient);
		Desc.Extent /= UpscaleFactor;
		Desc.TargetableFlags |= TexCreate_UAV;

		*OutColorTexture = GraphBuilder.CreateTexture(Desc,
TEXT("RayTracingReflections"));

		Desc.Format = PF_R16F;
```

```
			*OutRayHitDistanceTexture = GraphBuilder.CreateTexture(Desc,
TEXT("RayTracingReflectionsHitDistance"));
			*OutRayImaginaryDepthTexture = GraphBuilder.CreateTexture(Desc,
TEXT("RayTracingReflectionsImaginaryDepth"));
		}

		// When deferred materials are used, we need to dispatch the reflection shader
twice:
		// - First pass gathers reflected ray hit data and sorts it by hit shader ID.
		// - Second pass re-traces the reflected ray and performs full shading.
		// When deferred materials are not used, everything is done in a single pass.
		const uint32 NumPasses = bSortMaterials ? 2 : 1;
		const EDeferredMaterialMode DeferredMaterialModes[2] =
		{
			bSortMaterials ? EDeferredMaterialMode::Gather :
EDeferredMaterialMode::None,
			bSortMaterials ? EDeferredMaterialMode::Shade :
EDeferredMaterialMode::None,
		};

		FRDGBufferRef DeferredMaterialBuffer = nullptr;

		FIntPoint TileAlignedResolution = RayTracingResolution;
		if (SortTileSize)
		{
			TileAlignedResolution = FIntPoint::DivideAndRoundUp(RayTracingResolution,
SortTileSize) * SortTileSize;
		}

		const uint32 DeferredMaterialBufferNumElements = TileAlignedResolution.X *
TileAlignedResolution.Y;

		FRayTracingReflectionsRGS::FParameters CommonParameters;

		CommonParameters.SamplesPerPixel = SamplePerPixel;
		CommonParameters.MaxBounces = GRayTracingReflectionsMaxBounces > -1?
GRayTracingReflectionsMaxBounces :
View.FinalPostProcessSettings.RayTracingReflectionsMaxBounces;
		CommonParameters.HeightFog = HeightFog;
		CommonParameters.ShouldDoDirectLighting = GRayTracingReflectionsDirectLighting;
		CommonParameters.ReflectedShadowsType = GRayTracingReflectionsShadows > -1 ?
GRayTracingReflectionsShadows :
(int32)View.FinalPostProcessSettings.RayTracingReflectionsShadows;
		CommonParameters.ShouldDoEmissiveAndIndirectLighting =
GRayTracingReflectionsEmissiveAndIndirectLighting;
		CommonParameters.UpscaleFactor = UpscaleFactor;
		CommonParameters.ReflectionMinRayDistance =
FMath::Min(GRayTracingReflectionsMinRayDistance, GRayTracingReflectionsMaxRayDistance);
		CommonParameters.ReflectionMaxRayDistance = GRayTracingReflectionsMaxRayDistance;
		CommonParameters.ReflectionMaxRoughness =
FMath::Clamp(GRayTracingReflectionsMaxRoughness >= 0 ? GRayTracingReflectionsMaxRoughness
: View.FinalPostProcessSettings.RayTracingReflectionsMaxRoughness, 0.01f, 1.0f);
		CommonParameters.ReflectionMaxNormalBias = GetRaytracingMaxNormalBias();
		CommonParameters.RayTracingResolution = RayTracingResolution;
		CommonParameters.TileAlignedResolution = TileAlignedResolution;

		CommonParameters.TLAS = View.RayTracingScene.RayTracingSceneRHI-
>GetShaderResourceView();
		CommonParameters.ViewUniformBuffer = View.ViewUniformBuffer;
		CommonParameters.LightDataPacked = CreateLightDataPackedUniformBuffer(Scene-
>Lights, View, EUniformBufferUsage::UniformBuffer_SingleFrame);
		CommonParameters.SceneTexturesStruct = CreateSceneTextureUniformBuffer(
SceneContext, FeatureLevel, ESceneTextureSetupMode::All,
```

```
EUniformBufferUsage::UniformBuffer_SingleFrame);
	CommonParameters.ReflectionStruct = CreateReflectionUniformBuffer(View,
EUniformBufferUsage::UniformBuffer_SingleFrame);
	CommonParameters.FogUniformParameters = CreateFogUniformBuffer(View,
EUniformBufferUsage::UniformBuffer_SingleFrame);
	CommonParameters.ColorOutput = GraphBuilder.CreateUAV(*OutColorTexture);
	CommonParameters.RayHitDistanceOutput =
GraphBuilder.CreateUAV(*OutRayHitDistanceTexture);
	CommonParameters.RayImaginaryDepthOutput =
GraphBuilder.CreateUAV(*OutRayImaginaryDepthTexture);
	CommonParameters.SortTileSize = SortTileSize;
	for (uint32 PassIndex = 0; PassIndex < NumPasses; ++PassIndex)

{
		FRayTracingReflectionsRGS::FParameters* PassParameters =
GraphBuilder.AllocParameters<FRayTracingReflectionsRGS::FParameters>();
		*PassParameters = CommonParameters;

		const EDeferredMaterialMode DeferredMaterialMode =
DeferredMaterialModes[PassIndex];

		if (DeferredMaterialMode != EDeferredMaterialMode::None)
		{
			if (DeferredMaterialMode == EDeferredMaterialMode::Gather)
			{
				FRDGBufferDesc Desc =
FRDGBufferDesc::CreateStructuredDesc(sizeof(FDeferredMaterialPayload),
DeferredMaterialBufferNumElements);
				DeferredMaterialBuffer = GraphBuilder.CreateBuffer(Desc,
TEXT("RayTracingReflectionsMaterialBuffer"));
			}

			PassParameters->MaterialBuffer =
GraphBuilder.CreateUAV(DeferredMaterialBuffer);
		}

		FRayTracingReflectionsRGS::FPermutationDomain PermutationVector;

	PermutationVector.Set<FRayTracingReflectionsRGS::FDeferredMaterialMode>(DeferredMa
terialMode);

		auto RayGenShader = View.ShaderMap-
>GetShader<FRayTracingReflectionsRGS>(PermutationVector);
		ClearUnusedGraphResources(RayGenShader, PassParameters);

		if (DeferredMaterialMode == EDeferredMaterialMode::Gather)
		{
			GraphBuilder.AddPass(
				RDG_EVENT_NAME("ReflectionRayTracingGatherMaterials %dx%d",
TileAlignedResolution.X, TileAlignedResolution.Y),
				PassParameters,
				ERenderGraphPassFlags::Compute,
				[PassParameters, this, &View, RayGenShader,
TileAlignedResolution](FRHICommandList& RHICmdList)
			{
				FRHIRayTracingPipelineState* Pipeline =
BindRayTracingDeferredMaterialGatherPipeline(RHICmdList, View, RayGenShader-
>GetRayTracingShader());

				FRayTracingShaderBindingsWriter GlobalResources;
				SetShaderParameters(GlobalResources, RayGenShader,
*PassParameters);
```

```
                        FRayTracingSceneRHIParamRef RayTracingSceneRHI =
View.RayTracingScene.RayTracingSceneRHI;
                        RHICmdList.RayTraceDispatch(Pipeline, RayGenShader-
>GetRayTracingShader(), RayTracingSceneRHI, GlobalResources, TileAlignedResolution.X,
TileAlignedResolution.Y);
                    });

                    // A material sorting pass
                    const uint32 SortSize =
CVarRayTracingReflectionsSortSize.GetValueOnRenderThread();
                    if (SortSize)
                    {
                        SortDeferredMaterials(GraphBuilder, View, SortSize,

DeferredMaterialBufferNumElements, DeferredMaterialBuffer);
                    }
                }
                else
                {
                    GraphBuilder.AddPass(
                        RDG_EVENT_NAME("ReflectionRayTracing %dx%d",
RayTracingResolution.X, RayTracingResolution.Y),
                        PassParameters,
                        ERenderGraphPassFlags::Compute,
                        [PassParameters, this, &View, RayGenShader,
RayTracingResolution, DeferredMaterialBufferNumElements,
DeferredMaterialMode](FRHICommandList& RHICmdList)
                    {
                        FRayTracingShaderBindingsWriter GlobalResources;
                        SetShaderParameters(GlobalResources, RayGenShader,
*PassParameters);

                        FRayTracingSceneRHIParamRef RayTracingSceneRHI =
View.RayTracingScene.RayTracingSceneRHI;

                        if (DeferredMaterialMode == EDeferredMaterialMode::Shade)
                        {
                            // Shading pass for sorted materials uses 1D dispatch
over all elements in the material buffer.
                            // This can be reduced to the number of output pixels
if sorting pass guarantees that all invalid entries are moved to the end.

        RHICmdList.RayTraceDispatch(View.RayTracingMaterialPipeline, RayGenShader-
>GetRayTracingShader(), RayTracingSceneRHI, GlobalResources,
DeferredMaterialBufferNumElements, 1);
                        }
                        else // EDeferredMaterialMode::None
                        {

        RHICmdList.RayTraceDispatch(View.RayTracingMaterialPipeline, RayGenShader-
>GetRayTracingShader(), RayTracingSceneRHI, GlobalResources, RayTracingResolution.X,
RayTracingResolution.Y);
                        }
                    });
                }
        }

}
#else // !RHI_RAYTRACING
{
        check(0);
}
#endif
```

7. RayTracingShadows.cpp

```
#include "DeferredShadingRenderer.h"

#if RHI_RAYTRACING

#include "ClearQuad.h"
#include "PathTracingUniformBuffers.h"
#include "SceneRendering.h"
#include "SceneRenderTargets.h"
#include "RenderGraphBuilder.h"
#include "RenderTargetPool.h"
#include "RHIResources.h"
#include "UniformBuffer.h"
#include "VisualizeTexture.h"
#include "RayGenShaderUtils.h"
#include "RaytracingOptions.h"
#include "BuiltInRayTracingShaders.h"
#include "RayTracing/RayTracingMaterialHitShaders.h"
#include "Containers/DynamicRHIResourceArray.h"
#include "SceneViewFamilyBlackboard.h"

static float GRayTracingMaxNormalBias = 0.1f;
static FAutoConsoleVariableRef CVarRayTracingNormalBias(
	TEXT("r.RayTracing.NormalBias"),
	GRayTracingMaxNormalBias,
	TEXT("Sets the max. normal bias used for offseting the ray start position along
the normal (default = 0.1, i.e., 1mm)")
);

static int32 GRayTracingShadowsEnableMaterials = 1;
static FAutoConsoleVariableRef CVarRayTracingShadowsEnableMaterials(
	TEXT("r.RayTracing.Shadows.EnableMaterials"),
	GRayTracingShadowsEnableMaterials,
	TEXT("Enables material shader binding for shadow rays. If this is disabled, then a
default trivial shader is used. (default = 1)")
);

static TAutoConsoleVariable<int32> CVarRayTracingShadowsEnableTwoSidedGeometry(
	TEXT("r.RayTracing.Shadows.EnableTwoSidedGeometry"),
	1,
	TEXT("Enables two-sided geometry when tracing shadow rays (default = 1)"),
	ECVF_RenderThreadSafe
);

class FOcclusionRGS : public FGlobalShader
{
	DECLARE_GLOBAL_SHADER(FOcclusionRGS)
	SHADER_USE_ROOT_PARAMETER_STRUCT(FOcclusionRGS, FGlobalShader)

	class FLightTypeDim : SHADER_PERMUTATION_INT("LIGHT_TYPE", LightType_MAX);
	class FDenoiserOutputDim : SHADER_PERMUTATION_INT("DIM_DENOISER_OUTPUT", 4);
	class FEnableTwoSidedGeometryDim :
SHADER_PERMUTATION_BOOL("ENABLE_TWO_SIDED_GEOMETRY");

	using FPermutationDomain = TShaderPermutationDomain<FLightTypeDim,
FDenoiserOutputDim, FEnableTwoSidedGeometryDim>;

	static bool ShouldCompilePermutation(const FGlobalShaderPermutationParameters&
Parameters)
	{
		return ShouldCompileRayTracingShadersForProject(Parameters.Platform);
	}
```

```
	BEGIN_SHADER_PARAMETER_STRUCT(FParameters, )
		SHADER_PARAMETER(uint32, SamplesPerPixel)
		SHADER_PARAMETER(float, NormalBias)
		SHADER_PARAMETER(uint32, LightingChannelMask)
		SHADER_PARAMETER(FIntRect, LightScissor)

		SHADER_PARAMETER_STRUCT(FLightShaderParameters, Light)
		SHADER_PARAMETER_STRUCT_INCLUDE(FSceneViewFamilyBlackboard,
SceneBlackboard)

		SHADER_PARAMETER_SRV(RaytracingAccelerationStructure, TLAS)
		SHADER_PARAMETER_RDG_TEXTURE_UAV(RWTexture2D<float4>, RWOcclusionMaskUAV)
		SHADER_PARAMETER_RDG_TEXTURE_UAV(RWTexture2D<float>, RWRayDistanceUAV)
		SHADER_PARAMETER_STRUCT_REF(FViewUniformShaderParameters,
ViewUniformBuffer)
	END_SHADER_PARAMETER_STRUCT()
};

IMPLEMENT_GLOBAL_SHADER(FOcclusionRGS,
"/Engine/Private/RayTracing/RayTracingOcclusionRGS.usf", "OcclusionRGS", SF_RayGen);

float GetRaytracingMaxNormalBias()
{
	return FMath::Max(0.01f, GRayTracingMaxNormalBias);
}

void FDeferredShadingSceneRenderer::PrepareRayTracingShadows(const FViewInfo& View,
TArray<FRayTracingShaderRHIParamRef>& OutRayGenShaders)
{
	// Declare all RayGen shaders that require material closest hit shaders to be
bound

	const IScreenSpaceDenoiser::EShadowRequirements DenoiserRequirements[] =
	{
		IScreenSpaceDenoiser::EShadowRequirements::Bailout,
		IScreenSpaceDenoiser::EShadowRequirements::ClosestOccluder,
		IScreenSpaceDenoiser::EShadowRequirements::PenumbraAndAvgOccluder,
		IScreenSpaceDenoiser::EShadowRequirements::PenumbraAndClosestOccluder,
	};

	for (int32 LightType = 0; LightType < LightType_MAX; ++LightType)
	{
		for (IScreenSpaceDenoiser::EShadowRequirements DenoiserRequirement :
DenoiserRequirements)
		{
			FOcclusionRGS::FPermutationDomain PermutationVector;
			PermutationVector.Set<FOcclusionRGS::FLightTypeDim>(LightType);

	PermutationVector.Set<FOcclusionRGS::FDenoiserOutputDim>((int32)DenoiserRequiremen
t);

	PermutationVector.Set<FOcclusionRGS::FEnableTwoSidedGeometryDim>(CVarRayTracingSha
dowsEnableTwoSidedGeometry.GetValueOnRenderThread() != 0);

			TShaderMapRef<FOcclusionRGS> RayGenerationShader(View.ShaderMap,
PermutationVector);
			OutRayGenShaders.Add(RayGenerationShader->GetRayTracingShader());
		}
	}
}
```

```
#endif // RHI_RAYTRACING

void FDeferredShadingSceneRenderer::RenderRayTracingShadows(
	FRDGBuilder& GraphBuilder,
	const FSceneViewFamilyBlackboard& SceneBlackboard,
	const FViewInfo& View,
	const FLightSceneInfo& LightSceneInfo,
	const IScreenSpaceDenoiser::FShadowRayTracingConfig& RayTracingConfig,
	IScreenSpaceDenoiser::EShadowRequirements DenoiserRequirements,
	FRDGTextureRef* OutShadowMask,
	FRDGTextureRef* OutRayHitDistance)
#if RHI_RAYTRACING
{
	FLightSceneProxy* LightSceneProxy = LightSceneInfo.Proxy;
	check(LightSceneProxy);

	// Render targets
	FRDGTextureRef ScreenShadowMaskTexture;
	{
		FRDGTextureDesc Desc = FRDGTextureDesc::Create2DDesc(
			SceneBlackboard.SceneDepthBuffer->Desc.Extent,
			PF_FloatRGBA,
			FClearValueBinding::Black,
			TexCreate_None,
			TexCreate_ShaderResource | TexCreate_RenderTargetable |
TexCreate_UAV,
			/* bInForceSeparateTargetAndShaderResource = */ false);
		ScreenShadowMaskTexture = GraphBuilder.CreateTexture(Desc,
TEXT("RayTracingOcclusion"));
	}

	FRDGTextureRef RayDistanceTexture;
	{
		FRDGTextureDesc Desc = FRDGTextureDesc::Create2DDesc(
			SceneBlackboard.SceneDepthBuffer->Desc.Extent,
			PF_R16F,
			FClearValueBinding::Black,
			TexCreate_None,
			TexCreate_ShaderResource | TexCreate_RenderTargetable |
TexCreate_UAV,
			/* bInForceSeparateTargetAndShaderResource = */ false);
		RayDistanceTexture = GraphBuilder.CreateTexture(Desc,
TEXT("RayTracingOcclusionDistance"));
	}

	FIntRect ScissorRect = { {0,0}, View.ViewRect.Size() };

	if (LightSceneProxy->GetScissorRect(ScissorRect, View, View.ViewRect))
	{
		// Account for scissor being defined on the whole frame viewport while the
trace is only on the view subrect
		ScissorRect.Min = ScissorRect.Min - View.ViewRect.Min;
		ScissorRect.Max = ScissorRect.Max - View.ViewRect.Min;
	}
	else
	{
		ScissorRect = { {0,0}, View.ViewRect.Size() };
	}

	// Ray generation pass for shadow occlusion.
	{
		FOcclusionRGS::FParameters* PassParameters =
GraphBuilder.AllocParameters<FOcclusionRGS::FParameters>();
```

```
		PassParameters->RWOcclusionMaskUAV =
GraphBuilder.CreateUAV(FRDGTextureUAVDesc(ScreenShadowMaskTexture));
		PassParameters->RWRayDistanceUAV =
GraphBuilder.CreateUAV(FRDGTextureUAVDesc(RayDistanceTexture));
		PassParameters->SamplesPerPixel = RayTracingConfig.RayCountPerPixel;
		PassParameters->NormalBias = GetRaytracingMaxNormalBias();
		PassParameters->LightingChannelMask = LightSceneProxy-
>GetLightingChannelMask();
		LightSceneProxy->GetLightShaderParameters(PassParameters->Light);
		PassParameters->TLAS = View.RayTracingScene.RayTracingSceneRHI-
>GetShaderResourceView();
		PassParameters->ViewUniformBuffer = View.ViewUniformBuffer;
		PassParameters->SceneBlackboard = SceneBlackboard;
		PassParameters->LightScissor = ScissorRect;

		FOcclusionRGS::FPermutationDomain PermutationVector;
		PermutationVector.Set<FOcclusionRGS::FLightTypeDim>(LightSceneProxy-
>GetLightType());
		if (DenoiserRequirements ==
IScreenSpaceDenoiser::EShadowRequirements::ClosestOccluder)
		{
			ensure(RayTracingConfig.RayCountPerPixel == 1);
			PermutationVector.Set<FOcclusionRGS::FDenoiserOutputDim>(1);
		}
		else if (DenoiserRequirements ==
IScreenSpaceDenoiser::EShadowRequirements::PenumbraAndAvgOccluder)
		{
			PermutationVector.Set<FOcclusionRGS::FDenoiserOutputDim>(2);
		}
		else if (DenoiserRequirements ==
IScreenSpaceDenoiser::EShadowRequirements::PenumbraAndClosestOccluder)
		{
			PermutationVector.Set<FOcclusionRGS::FDenoiserOutputDim>(3);
		}
		else
		{
			PermutationVector.Set<FOcclusionRGS::FDenoiserOutputDim>(0);
		}

	PermutationVector.Set<FOcclusionRGS::FEnableTwoSidedGeometryDim>(CVarRayTracingSha
dowsEnableTwoSidedGeometry.GetValueOnRenderThread() != 0);

		TShaderMapRef<FOcclusionRGS>
RayGenerationShader(GetGlobalShaderMap(FeatureLevel), PermutationVector);

		ClearUnusedGraphResources(*RayGenerationShader, PassParameters);

		FIntPoint Resolution(View.ViewRect.Width(), View.ViewRect.Height());

		GraphBuilder.AddPass(
			RDG_EVENT_NAME("RayTracedShadow (spp=%d) %dx%d",
RayTracingConfig.RayCountPerPixel, View.ViewRect.Width(), View.ViewRect.Height()),
			PassParameters,
			ERenderGraphPassFlags::Compute,
			[this, &View, RayGenerationShader, PassParameters,
Resolution](FRHICommandList& RHICmdList)
		{
			FRayTracingShaderBindingsWriter GlobalResources;
			SetShaderParameters(GlobalResources, *RayGenerationShader,
*PassParameters);

			FRayTracingSceneRHIParamRef RayTracingSceneRHI =
View.RayTracingScene.RayTracingSceneRHI;
```

```
                        if (GRayTracingShadowsEnableMaterials)
                        {
                                RHICmdList.RayTraceDispatch(View.RayTracingMaterialPipeline,
RayGenerationShader->GetRayTracingShader(), RayTracingSceneRHI, GlobalResources,
Resolution.X, Resolution.Y);
                        }
                        else
                        {
                                FRayTracingPipelineStateInitializer Initializer;

                                Initializer.MaxPayloadSizeInBytes = 52; //
sizeof(FPackedMaterialClosestHitPayload)

                                FRayTracingShaderRHIParamRef RayGenShaderTable[] = {
RayGenerationShader->GetRayTracingShader() };
                                Initializer.SetRayGenShaderTable(RayGenShaderTable);

                                FRayTracingShaderRHIParamRef MissShaderTable[] = {
View.ShaderMap->GetShader<FDefaultMaterialMS>()->GetRayTracingShader() };
                                Initializer.SetMissShaderTable(MissShaderTable);

                                FRayTracingShaderRHIParamRef HitGroupTable[] = {
View.ShaderMap->GetShader<FOpaqueShadowHitGroup>()->GetRayTracingShader() };
                                Initializer.SetHitGroupTable(HitGroupTable);
                                Initializer.bAllowHitGroupIndexing = false; // Use the same
hit shader for all geometry in the scene by disabling SBT indexing.

                                FRHIRayTracingPipelineState* Pipeline =
PipelineStateCache::GetAndOrCreateRayTracingPipelineState(Initializer);

                                RHICmdList.RayTraceDispatch(Pipeline, RayGenerationShader-
>GetRayTracingShader(), RayTracingSceneRHI, GlobalResources, Resolution.X, Resolution.Y);
                        }
                });
        }

        *OutShadowMask = ScreenShadowMaskTexture;
        *OutRayHitDistance = RayDistanceTexture;
}
#else // !RHI_RAYTRACING
{
        unimplemented();
}
#endif
```

Повний список класів:

PaperFlipbookActor
PaperGroupedSpriteActor
PaperSpriteActor
PaperTerrainActor
PaperTileMapActor
Pawn
PlacedEditorUtilityBase
ReflectionCapture
RigidBodyBase
SceneCapture
SequenceRecorderGroup
SequencerKeyActor
SequencerMeshTrail
SkeletalMeshActor
SplineMeshActor
StaticMeshActor
TargetPoint
TextRenderActor
TrafficMovingCarlights
TriggerBase
Tutorial_BP_Class
VectorFieldVolume
VisualLoggerRenderingActor
VREDAnimationPlayer
VREditorAvatarActor
VREditorBaseActor
VREditorTeleporter
VREDVariantsSelector
129 items (1 selected)

					ДП ІСз-5103.1260-с.ССВ	
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна варіантів використань	Літера / Маса / Масштаб
Розробив		Степовий Т.О.				
Перевірив		Родіонов П.Ю.				Аркуш 1 / Аркушів 1
Т. кон.					Трасування променів в реальному часі для комп'ютерних ігор	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІСз-51
Н. кон.		Тєлишева Т.О.				
Затвердив		Родіонов П.Ю.				

					ДП ІСз-5103.1260-с.ССК	
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна класів програмного забезпечення	Літера / Маса / Масштаб
Розробив		Степовий Т.О.				
Перевірив		Родіонов П.Ю.				Аркуш 1 / Аркушів 1
Т. кон.					Трасування променів в реальному часі для комп'ютерних ігор	*КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІСз-51*
Н. кон.		Тєлишева Т.О.				
Затвердив		Родіонов П.Ю.				

					ДП ІСз-5103.1260-с.ССМ		
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна компонентів програмного забезпечення	Літера	Маса / Масштаб
Розробив		Степовий Т.О.					
						Аркуш 1	Аркушів 1
Перевірив		Родіонов П.Ю.					
Т. кон.					Трасування променів в реальному часі для комп'ютерних ігор	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІС-51	
Н. кон.		Тєлишева Т.О.					
Затвердив		Родіонов П.Ю.					

					ДП ІСз-5103.1260-с.ССМ		
					Схема структурна послідовності	Літера	Маса / Масштаб
Зм.	Арк.	№ документа	Підпис	Дата			
Розробив		Степовий Т.О.					
						Аркуш 1	Аркушів 1
Перевірив		Родіонов П.Ю.					
Т. кон.					Трасування променів в реальному часі для комп'ютерних ігор	*КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІСз-51*	
Н. кон.		Тєлишева Т.О.					
Затвердив		Родіонов П.Ю.					

					ДП ІСз-5103.1260-с.КЕ		
Зм.	Арк.	№ документа	Підпис	Дата	Креслення вигляду екранних форм	Літера	Маса / Масштаб
Розробив		Степовий Т.О.					
Перевірив		Родіонов П.Ю.				Аркуш 1	Аркушів 2
Т. кон.					Трасування променів в реальному часі для комп'ютерних ігор	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІСз-51	
Н. кон.		Тєлишева Т.О.					
Затвердив		Родіонов П.Ю.					

					ДП ІСз-5103.1260-с.КЕ	
Зм.	Арк.	№ документа	Підпис	Дата	Креслення вигляду екранних форм	Літера / Маса / Масштаб
Розробив		Степовий Т.О.				
Перевірив		Родіонов П.Ю.				Аркуш 2 / Аркушів 2
Т. кон.					Трасування променів в реальному часі для комп'ютерних ігор	КПІ ім. Ігоря Сікорського кафедра АСОІУ гр. ІСз-51
Н. кон.		Тєлишева Т.О.				
Затвердив		Родіонов П.Ю.				