ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка методу оптимізованого рендерінгу динамічного освітлення та атмосферних ефектів в 3D іграх»

на здобуття освітнього ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
(код, найменування спеціальності)
освітньо-професійної програми «Інженерія програмного забезпечення»
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Ельмар КЕНГЕРЛІ
(підпис)

Виконав: здобувач вищої освіти групи ПДМ-64
Ельмар КЕНГЕРЛІ

Керівник: Олена НЕГОДЕНКО
д.т.н., доцент

Рецензент: _____
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Магістр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення

_____ Ірина ЗАМРІЙ
«_____» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Кенгерлі Ельмару Фаіговичу

1. Тема кваліфікаційної роботи: Розробка методу оптимізованого рендерінгу динамічного освітлення та атмосферних ефектів в 3D іграх

керівник кваліфікаційної роботи Олена НЕГОДЕНКО д.т.н., доцент,

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «19» 10.2023 р. №145.

2. Строк подання кваліфікаційної роботи «29» грудня 2023 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, метод оптимізованого розрахунку освітлення в 3D сцені.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Дослідження існуючих методів розрахунку освітлення
2. Аналіз алгоритмів та математичних моделей для розрахунку освітлення точки в 3D сцені
3. Розробка формули для розрахунку можливостей комп’ютера

5. Перелік графічного матеріалу: *презентація*

1. Порівняння способів рендерінгу освітлення в 3D іграх
2. Архітектура алгоритму рендерінгу освітлення
3. Опис методів розрахунку освітлення
4. Математичні моделі розрахунку освітленності точки
5. Розрахунок продуктивності системи комп'ютера

6. Дата видачі завдання «19» жовтня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	19.10-05.11.23	
2	Аналіз показників системи комп'ютера для визначення його потужності	06.11-12.11.23	
3	Дослідження існуючих методів розрахунку променів від джерела освітлення	13.11-19.11.23	
4	Розробка реалізації на основі алгоритму оптимізованого методу рендерінгу	20.11-26.11.23	
5	Побудова математичних, інформаційних та функціональних моделей оптимізованого рендерінгу для досягнення високої якості візуальних ефектів в реальному часі	27.11-03.12.23	
6	Порівняння продуктивності оптимізованого методу з існуючими алгоритмами	04.12-10.12.23	
7	Оформлення роботи: вступ, висновки, реферат	11.12-20.12.23	
8	Розробка демонстраційних матеріалів	21.12-29.12.23	

Здобувач вищої освіти _____ Ельмар КЕНГЕРЛІ
(підпис)

Керівник
кваліфікаційної роботи _____ Олена НЕГОДЕНКО
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 7 стор., 2 табл., 18 рис., 20 джерел.

Мета роботи – оптимізація рендерінгу динамічного освітлення та атмосферних ефектів в 3D іграх

Об'єкт дослідження – рендерінг освітлення в 3D іграх.

Предмет дослідження – методи та алгоритми, що використовуються для рендерінгу освітлення та атмосферних ефектів

Короткий зміст роботи: У роботі проведено дослідження методів рендерінгу освітлення в 3D сцені. Проаналізовано основні показники продуктивності системи комп'ютера. На основі досліджень було виведено алгоритм оптимізованого динамічного рендерінгу освітлення в реальному часі.

КЛЮЧОВІ СЛОВА: РЕНДЕРІНГ, 3D ІГРИ, МОДЕЛЬ ФОНГА, МЕТОДИ РОЗРАХУНКУ ОСВІТЛЕННЯ ТОЧКИ.

ABSTRACT

Text part of the master's qualification work:
7 pages, 18 pictures, 2 tables, 20 sources.

The purpose of this research is to optimize the rendering of dynamic lighting and atmospheric effects in 3D games.

The primary object of this research is the rendering of lighting in 3D games.

The subject of this research is the methods and algorithms used for rendering lighting and atmospheric effects.

Summary of the work: This research undertakes a detailed exploration of cutting-edge methods in real-time rendering, aiming to push the boundaries of graphical quality and performance. The investigation covers the theoretical underpinnings of rendering algorithms, their practical implementations, and their potential impact on the gaming industry and other visualization applications.

The study begins by discussing the significance of real-time rendering, emphasizing its crucial role in creating immersive and realistic virtual environments. It further narrows down its focus to dynamic lighting and atmospheric effects, recognizing their pivotal role in shaping the visual appeal of 3D scenes.

Throughout the research, various rendering methods are scrutinized, including directed rendering for targeted illumination, fast ray tracing for efficient scene analysis, and Monte Carlo-based shading for realistic surface interactions. The exploration encompasses their strengths, limitations, and potential applications in diverse fields.

The practical relevance of the research is highlighted, emphasizing its potential to revolutionize the gaming industry by enhancing graphical capabilities and optimizing resource utilization. The findings have broader implications, extending to applications in virtual reality, architectural visualization, and medical simulations.

In summary, this research contributes to the field of computer graphics by providing a comprehensive analysis of state-of-the-art real-time rendering techniques. The work aims to bridge the gap between theoretical advancements and practical applications, paving the way for more immersive and visually captivating virtual experiences.

KEYWORDS: RENDERING, 3D GAMES, PHONG MODEL, METHODS FOR CALCULATING LIGHTING.

ЗМІСТ

ВСТУП 11

РОЗДІЛ 1. АНАЛІЗ МОДЕЛЕЙ ТА МЕТОДІВ РЕНДЕРІНГУ ОСВІТЛЕННЯ В 3D ІГРАХ 11

1.1. Аналіз існуючих методів рендерінгу освітлення 11

1.1.1. Метод спрямованого рендерінгу 13

1.1.2. Метод підпроменю 15

1.1.3. Швидке трасування променів 17

1.1.4. Непрямий рендерінг 19

1.1.5. Трасування променів 20

1.1.6. Затінення методом монте-карло 22

1.2. Модель освітлення фонга 24

1.3. Додаткові методи оптимізації 28

1.3.1. Використання буферів глибини 29

1.3.2. Z-буфер 30

1.3.3. W-буфер 31

1.3.4. Метод двійного буферу глибини 34

РОЗДІЛ 2. РОЗРОБКА МЕТОДУ ОПТИМІЗОВАНОГО РЕНДЕРІНГУ ДИНАМІЧНОГО ОСВІТЛЕННЯ ТА АТМОСФЕРНИХ ЕФЕКТІВ В 3D ІГРАХ 37

2.1. Розрахунок інтегральних можливостей комп’ютера 37

2.2. Опис алгоритму методу 46

2.3. Опис розроблених класів 48

РОЗДІЛ 3. ПРОВЕДЕННЯ МОДЕЛЮВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ 64

ВИСНОВКИ 68

ПЕРЕЛІК ПОСИЛАНЬ..... 69

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (ПРЕЗЕНТАЦІЯ) 71

ВСТУП

В сучасному світі віртуальної реальності та комп'ютерних ігор великою мірою визначається рівнем реалізму та емоційною залученістю гравців. Однією з ключових складових цього враження є якісне рендерінгове відтворення динамічного освітлення та атмосферних ефектів в тривимірних іграх. Розробка ефективного методу оптимізованого рендерінгу стає актуальною задачею для вдосконалення якості графічного відображення та створення максимально реалістичного ігрового середовища.

У зв'язку зі стрімким розвитком галузі відомостей та технологій комп'ютерної графіки, споживачі стають все вибагливішими, вимагаючи від розробників ігор та візуалізаційних систем нових рівнів деталізації та реалізму. Динамічне освітлення та атмосферні ефекти, такі як реалістичне сонячне світло, тіні, відображення атмосферних явищ і реакція на дії гравця стають ключовими чинниками, які визначають глибину іммерсії та емоційний зв'язок із віртуальним світом.

Об`єкт дослідження – рендерінг освітлення в 3D іграх.

Предмет дослідження – методи та алгоритми, що використовуються для рендерінгу освітлення та атмосферних ефектів.

Мета роботи – оптимізація рендерінгу динамічного освітлення та атмосферних ефектів в 3D іграх.

Методи дослідження – методи теорії систем та моделей клітинних автоматів, методи теорії інформації, апарат математичної статистики, методи проектування та розробки програмного забезпечення, технології об’єктно-орієнтованого програмування.

Практична значущість результатів даного дослідження полягає в істотному вдосконаленні якості графічного відображення в тривимірних іграх та інших візуалізаційних застосунках. Розробка та оптимізація методу рендерінгу динамічного освітлення та атмосферних ефектів має невідомий потенціал для подальших досягнень у галузі комп'ютерної графіки та віртуальної реальності. Отже, практична важливість даного дослідження полягає в його потенціалі вплинути на розвиток індустрії, покращити графічні можливості, забезпечити нові ігрові враження та розширити застосування візуалізаційних технологій.

1. АНАЛІЗ МОДЕЛЕЙ ТА МЕТОДІВ РЕНДЕРІНГУ ОСВІТЛЕННЯ В 3D ІГРАХ

1.1. Аналіз існуючих методів рендерінгу освітлення

Рендерінг освітлення є одним із найважливіших завдань у комп’ютерній графіці. Він дозволяє створювати реалістичні зображення сцен, які містять джерела світла. Існує багато різних методів рендерінгу освітлення, кожен з яких має свої переваги та недоліки. Порівняння методів рендерінгу можна побачити на таблиці 1.1.

Один з підходів, який варто розглядати, — це спрямований рендеринг. Цей метод передбачає випуск променів світла з камери та відстеження їх шляху, перетинання з поверхнями об'єктів і обчислення освітленості в точках перетину. Спрямований рендеринг дозволяє моделювати реалістичне поширення світла, але при цьому може виявитися обчислювально витратним завданням.

Ще один підхід — це підпромінь, який використовується для збору інформації про освітленість з різних напрямків. Цей метод дозволяє створювати більш точні та деталізовані зображення, але вимагає більше обчислювальних ресурсів.

Іншим популярним методом є швидке трасування променів, яке спрощує обчислення, інтерполюючи освітленість для областей зображення. Цей підхід дозволяє збільшити швидкість рендерінгу, але може втратити на точності порівняно з більш складними методами.

Непрямий рендеринг використовує методи, що оцінюють освітлення в точці, використовуючи взаємодію світла з поверхнею та відбиттям. Цей метод дозволяє моделювати більш складні ефекти, такі як м'яке та поглиблене освітлення, але може вимагати значних обчислювальних ресурсів.

Трасування променів є ще однією важливою технікою, що забезпечує точне відтворення світлових ефектів. Цей метод передбачає випуск променів від джерел світла та відстеження їх перетинання з поверхнею для обчислення освітленості.

Затінення методом Монте-Карло є інноваційним підходом, що базується на використанні випадкових вибірок для апроксимації освітленості. Цей метод дозволяє враховувати велику кількість взаємодій світла та створює фотореалістичні зображення.

Кожен із цих методів має свої унікальні переваги та застосування в конкретних сценаріях. Важливо провести детальний аналіз їх ефективності, часових затрат та обчислювальних вимог для вибору оптимального методу в конкретному випадку.

Необхідно також звертати увагу на апаратні ресурси комп'ютера, такі як продуктивність відеокарти, потужність процесора та обсяг оперативної пам'яті. Оптимальне використання цих ресурсів є ключовим фактором для забезпечення ефективності рендерінгу освітлення.

Таблиця 1.1

Опис методів рендерінгу освітлення

Метод рендерінгу	**Опис**	**Переваги**
Спрямований рендеринг	Джерело світла має чітко визначене положення і напрямок. Тінь утворюється в точці, де прямий промінь світла від джерела світла перетинає об'єкт.	Швидкий і ефективний з точки зору продуктивності
Підпромінь	Рендерінг освітлення шляхом обчислення шляху, який проходить промінь світла від камери до джерела.	Дозволяє створювати реалістичні ефекти освітлення, такі як тіні та відблиски
Швидке трасування променів	Рендерінг освітлення шляхом обчислення лише деяких шляхів, які проходять промені світла.	Швидший і ефективніший з точки зору продуктивності, ніж традиційне трасування променів

Продовження таблиці 1.1

Опис методів рендерінгу освітлення

Непрямий рендеринг	Джерело світла може бути розсіяним або відбитим від інших поверхонь. Тінь утворюється в точці, де розсіяне або відбите світло від джерела світла перетинає об'єкт.	Створює більш м'яке освітлення з меншими контрастами
Трасування променів	Рендеринг освітлення шляхом обчислення шляху, який проходить промінь світла від джерела до камери.	Найреалістичніший метод рендерінгу освітлення
Затінення методом Монте-Карло	Рендеринг освітлення шляхом випадкового вибору шляхів, які проходять промені світла.	Дозволяє створювати реалістичні ефекти освітлення, такі як тіні, відблиски, розсіяне світло та поглинання світла

1.1.1. Метод спрямованого рендерінгу

Метод спрямованого рендерінгу є ключовою стратегією у комп'ютерній графіці та візуалізації, спрямованою на створення деталізованих та реалістичних зображень сцен. Цей метод використовується для моделювання імпульсів світла та їхнього поширення від джерел до точок на поверхні об'єктів. Основна ідея методу полягає в відслідковуванні шляху світла від джерела до обраної точки на

поверхні. Принципово важливою є спроба моделювати фізичні властивості світла, такі як відбиття, пропускання та розсіювання, враховуючи взаємодію світлових променів з поверхнею об'єктів у сцені.

У контексті спрямованого рендерінгу, кожен світловий промінь розглядається як важливий імпульс, який пролягає через сцену та взаємодіє з об'єктами на своєму шляху. Застосування цього методу дозволяє моделювати різноманітні ефекти, такі як блики, тіні та м'яке освітлення, що призводить до отримання реалістичних та деталізованих зображень.

Однією з ключових характеристик спрямованого рендерінгу є здатність враховувати взаємодію світлових променів із сценою та їхнім впливом на окремі об'єкти. Це дозволяє отримувати більш точні та фотореалістичні зображення, оскільки враховуються всі нюанси розсіювання та відбиття світла. Ще однією важливою частиною методу є врахування характеристик поверхонь об'єктів, таких як їхні оптичні властивості та матеріали. Це дає можливість моделювати реальні матеріали з точністю до їхньої поведінки під впливом світла.

Прямий рендеринг - це найпростіший метод освітлення в 3D-графіці. Він розглядає світло як прямі промені, що випромінюються джерелами та потрапляють на поверхні. Спрощено, уявіть себе, як кидаєте дротики від ламп - кожен проліт освітлює певну точку. Відстежуючи ці "світлові дротики" та обчислюючи їх вплив, ми визначаємо яскравість кожного елемента сцени.

Теорія ґрунтується на двох ключових рівняннях:

- Інтенсивність освітлення: світло в точці (I) дорівнює інтенсивності джерела (L), помноженій на кут між променем світла та нормаллю поверхні (θ). Світло рівномірно розподіляється по поверхні, а під більшим кутом воно слабшає.

- Ослаблення джерела: світло послаблюється зі збільшенням відстані (r) від джерела, згідно з обернено-квадратичним законом:

$$L = I_0 / r^2 \tag{1.1}$$

де:

I_0 - світло в точці;

L - інтенсивність світла в точці;

r - відстань від джерела.

Проміні стають слабшими, пролітаючи далі.

Прямий рендерінг є простим та швидким алгоритмом. Він ідеально підходить для швидкої анімації або простих сцен з одиничним джерелом світла. Проте, його обмеження є суттєвими:

- Відсутність непрямого освітлення: проміні не враховують відбиття та переломлення, не відтворюючи тіней, відбитків або м'якого свічення непрямого освітлення. Уявіть сонячну кімнату, освітлену лише прямими променями, без ніжних відбитків від стін.

- Нереалістичне освітлення: сцени часто виглядають жорсткими та штучними, без тонкої взаємодії світла та тіні, характерної для реальності.

Незважаючи на обмеження, прямий рендеринг залишається цінним інструментом, особливо як основа для більш складних методів. Розуміючи його базові принципи, ми можемо оцінити витонченість інших техніків, які на його основі створюють справді захоплюючі сцени, залиті реалістичним світлом.

1.1.2. Метод підпроменю

Підпромінь (subsurface scattering) — це явище в комп'ютерній графіці, що моделює поведінку світла при проникненні та розсіюванні в транслюцентних матеріалах, таких як шкіра, віск або мармур. Цей ефект сприяє реалістичному відображенню матеріалів, захоплюючи змішання світла під їхніми поверхнями.
Метод підпроменю, що застосовується у комп'ютерній графіці та візуалізації сцен, є важливою технікою рендерінгу освітлення, яка дозволяє отримати детальні та реалістичні зображення. Цей метод заснований на ідеї моделювання розподілу світла в сцені, враховуючи його різноманітні взаємодії з поверхнями об'єктів.

У методі підпроменю основною концепцією є розглядання кожної точки поверхні об'єкта як джерела світла, що розбивається на безліч менших областей або підпроменів. Кожен підпромінь вважається невеликим джерелом світла, яке випромінює енергію в різних напрямках. Такий підхід дозволяє враховувати різноманітність освітленості та тіней, що виникають внаслідок різних джерел світла та їх взаємодії з поверхнею об'єкта. Основна ідея полягає в тому, щоб кожен піксель на екрані відображав не тільки основне джерело світла, але й всі можливі взаємодії з іншими поверхнями та об'єктами в сцені. Для цього вводиться поняття підпроменевого освітлення, яке враховує вплив кожного підпроменя на конкретну точку на поверхні. Це дає змогу отримувати більш точні та деталізовані зображення, оскільки враховуються усі нюанси освітлення та відбиття світла в сцені. Один із ключових аспектів методу підпроменю — це моделювання взаємодії світла з матеріалами об'єктів. Різні матеріали можуть мати різні характеристики відбивання, поглиблення та розсіювання світла. Метод підпроменю дозволяє точно моделювати ці характеристики, створюючи більш реалістичні зображення. Окрім того, метод підпроменю є ефективним у вирішенні проблеми тіней та м'якого освітлення. Враховуючи взаємодію світла з різними об'єктами та їхнім впливом на один одного, цей метод дозволяє отримувати зображення з більш натуральними тінями та м'яким освітленням.

Математична модель, що керує підповерхневим розсіюванням, включає рівняння дифузії, яке описує, як світло поширюється через середовище. Рівняння враховує такі фактори, як поглиблення, розсіювання та випромінювання світла в матеріалі. Зазвичай його чисельно розв'язують через складність реальних сценаріїв.

Рівняння дифузії для SSS виглядає наступним чином:

$$\nabla_2 I(\boldsymbol{r}) + \frac{\mu_s}{\mu_a} I(\boldsymbol{r}) = S(\boldsymbol{r}) \tag{1.2}$$

де:

I(r) - інтенсивність світла в позиції r;

∇_2 - оператор Лапласа;

μ_s - коефіцієнт розсіювання;

μ_a - коефіцієнт поглиблення;

S(r) - термін джерела світла.

Це рівняння враховує баланс між поглибленням і розсіюванням, дозволяючи моделювати проникнення та дифузію світла в матеріалі. Різноманітні чисельні техніки, такі як методи скінченних різниць чи методи Монте-Карло, використовуються для розв'язання цього рівняння для реалістичного відображення.

1.1.3. Швидке трасування променів

Швидке трасування променів (FRT) є передовим методом в області комп'ютерної графіки, спрямованим на швидку та ефективну генерацію візуально реалістичних зображень. Основним принципом FRT є використання променів світла та їхньої взаємодії з об'єктами сцени для створення реалістичних вражень. Метод використовує техніку трасування променів, де для кожного пікселя на зображенні генерується промінь, що прокладається через віртуальну камеру в сцені. Поширення цього променя визначається його взаємодією з об'єктами сцени та визначає кінцевий колір для даного пікселя.

Однією з ключових особливостей FRT є його ефективність. Замість того, щоб розглядати всі можливі шляхи променів у сцені, метод використовує ряд технік та алгоритмів для визначення лише найважливіших променів, що суттєво прискорює процес генерації зображення. Це дозволяє використовувати FRT для відтворення великих та складних сцен у реальному часі. Важливою складовою методу є обробка відбиття та розсіювання світла на поверхнях об'єктів. FRT враховує такі явища, як блики, тіні та градієнти освітлення, що додає реалізму та глибини зображенню. Це дозволяє отримувати фотореалістичні результати, що добре відповідають реальним сценам.

Метод швидкого трасування променів використовується в різних областях, включаючи відеоігри, кіноіндустрію та віртуальну реальність. Його застосування дозволяє отримувати зображення високої якості із збереженням реалізму та ефективністю обчислень. Загальною ідеєю FRT є те, що він робить акцент на швидкій та ефективній генерації зображень, зберігаючи при цьому реалістичність та деталізацію. Цей метод визначається своєю здатністю розглядати та враховувати тільки суттєві аспекти світлової взаємодії в сцені, що дозволяє йому бути однією з найефективніших стратегій рендерінгу у галузі комп'ютерної графіки. Ключ до швидкого трасування променів полягає в оптимізації алгоритмів та структур даних для досягнення рендерингу в реальному часі або практично в реальному часі.

Основна математична модель трасування променів включає в себе випуск променів з камери в сцену та визначення їх взаємодії з об'єктами. Кожен промінь перевіряється на перетин з геометрією сцени, і проводяться обчислення для імітації реалістичних ефектів освітлення, відбиттів та преломлень.

Основні етапи алгоритму включають:

- Генерація променів: промені випускаються з камери через кожен піксель зображення.

- Тестування перетину: кожен промінь перевіряється на перетин з об'єктами сцени. Використовуються структури прискорення, такі як ієрархії об'ємів огорожі (BVH), для швидкого відкидання непотрібних частин сцени.

- Розрахунок тіней: якщо знайдено перетин, виконуються розрахунки для визначення кольору та інтенсивності пікселя на основі моделей освітлення, властивостей матеріалу та текстур.

- Відбиття та преломлення: промені трасуються для відбиття та преломлення, імітуючи складні світлові взаємодії.

- Глобальне освітлення: техніки, такі як трасування променів методом Монте-Карло, можуть використовуватися для ефектів глобального освітлення, захоплення непрямого освітлення.

Рівняння для променя в тривимірному просторі задається як:

$$R(t) = O + t * D \quad (1.3)$$

де:

R(t) - точка на промені при параметрі t;

O - початкова точка променя (положення камери);

D - напрямок променя.

1.1.4. Непрямий рендерінг

Непрямий рендерінг – це метод, спрямований на створення віртуальних зображень, де враховується взаємодія світла з поверхнями сцени, включаючи відбиття, пропускання та розсіювання. Його основна ідея полягає в тому, щоб не тільки враховувати прямий шлях світла від джерел до спостерігача, але і взаємодію світла з іншими поверхнями в середовищі.

Під час непрямого рендерінгу важливо враховувати різні типи взаємодії світла з поверхнею. Це може бути відбиття, коли світло відбивається від поверхні; пропускання, коли світло проникає через прозору поверхню; та розсіювання, коли світло розсіюється в різних напрямках. Основною складовою непрямого рендерінгу є моделювання різних взаємодій світла з матеріалами. Це включає в себе врахування властивостей матеріалів, таких як кольори відбиття та пропускання, а також імітацію ефектів, таких як розсіювання світла на матових поверхнях.

Однією з важливих аспектів непрямого рендерінгу є моделювання освітлення та тіней від інших об'єктів у сцені. Це дозволяє створювати реалістичні та деталізовані зображення, в яких враховується взаємодія світла з оточуючими поверхнями. У непрямому рендерінгу велику увагу приділяють розрахунку шляхів, якими може рухатися світло в сцені. Цей метод часто використовується в галузях комп'ютерної графіки, комп'ютерного зору, візуалізації та інших областях, де потрібно створювати реалістичні візуальні ефекти.

Непрямий рендеринг відіграє важливу роль в досягненні реалістичних ефектів освітлення та глобального освітлення в комп'ютерно генерованих зображеннях. Це особливо важливо в сценах з складними сценаріями освітлення, де світло не лише безпосередньо освітлює поверхні, а й відбивається та взаємодіє з іншими поверхнями в середовищі.

Математична модель для непрямого рендерингу часто базується на рівнянні рендерингу - фундаментальному рівнянні в комп'ютерній графіці, яке описує загальне вхідне випромінювання в точці на поверхні. Рівняння рендерингу виражається наступним чином:

$$L_o(p, \omega_o) = L_e(p, \omega_o) + \int_\Omega f(p, \omega_i, \omega_o) \cdot L_i(p, \omega_i) \cdot (\omega_i \cdot N) d\omega_i \quad (1.4)$$

де:

$L_o(p, \omega_o)$ - вихідне випромінювання з точки p в напрямку ω_o;

$L_e(p, \omega_o)$ - випромінюване випромінювання в точці p в напрямку ω_o;

$f(p, \omega_i, \omega_o)$ - функція розподілу відбиття для відображення властивостей поверхні;

$L_i(p,\omega i)$ - вхідне випромінювання з напрямку ω_i;

ω_i та ω_o - вхідний та вихідний напрямки світла відповідно;

N - нормаль поверхні.

Інтеграл представляє собою сумування по всіх можливих напрямках вхідного світла. Розв'язання рівняння рендерингу точно вимагає вдосконалених алгоритмів, таких як методи Монте-Карло, для імітації складної взаємодії світла в сценах.

1.1.5. Трасування променів

Трасування променів – це техніка візуалізації в комп'ютерній графіці, яка використовується для моделювання взаємодії світла з віртуальними об'єктами у тривимірній сцені.

Метод трасування променів є одним із фундаментальних підходів до створення комп'ютерних зображень. Цей метод моделює шлях світла від джерела до камери, враховуючи взаємодію з об'єктами сцени. Замість обчислення освітлення для кожного пікселя зображення, він використовує ідею променевого сліду, де для кожного променя визначається його взаємодія з об'єктами сцени. У методі трасування променів промені випускаються з камери та відбиваються або преломлюються від поверхонь об'єктів у сцені. Ці взаємодії обчислюються для кожного променя, інколи долаючи десятки та сотні відбиттів та преломлень для досягнення реалістичності.

Завдяки методу трасування променів можна моделювати важкі фізичні ефекти, такі як тіні, відблиски та розсіювання світла. Цей підхід дозволяє створювати вражаючі візуальні ефекти та реалістично відтворювати поведінку світла в різних сценаріях. Однією з ключових особливостей методу є його витрати на обчислення, які можуть бути високими при складних сценах. Тим не менше, цей метод залишається популярним завдяки своїй здатності генерувати візуально захоплюючі зображення та ефективно моделювати фізичні явища.

На відміну від растеризації, яка працює на рівні пікселів, трасування променів відстежує шлях окремих променів світла, надаючи більш фізично точне представлення освітлення та відображень.

Трасування променів полягає в імітації поведінки світлових променів, коли вони прокладають свій шлях через сцену. Процес починається з генерації первинних променів від камери через кожен піксель. Потім ці промені відстежуються у сцені, і коли вони перетинаються з об'єктами, генеруються вторинні промені для відображень, преломлень та розрахунків тіней.

Трасування променів відмінно справляється з рендерингом реалістичних ефектів освітлення, включаючи точні відображення, тіні та глобальне освітлення. Воно впорається з складними сценами з різними матеріалами та умовами освітлення, захоплюючи тонкі деталі і виробляючи високоякісні зображення, які використовуються в обчислювальній графіці (CGI) для фільмів, архітектурної візуалізації та інших галузях.

Основа трасування променів включає математичні моделі для опису поведінки світлових променів. Основне рівняння - це рівняння променя:

$$R(t) = O + t * D \quad (1.5)$$

де:

R(t) - точка на промені при параметрі t;

O - початкова точка променя (положення камери);

D - напрямок променя.

1.1.6. Затінення методом Монте-Карло

Метод Монте-Карло в затіненні — це підхід до обчислення освітленості в точці сцени, який базується на випадковому виборі напрямків для обчислення освітлення. Замість аналітичного розрахунку використовується статистичний підхід, де велику кількість випадкових променів випускають в різних напрямках і обчислюють середнє значення освітленості для точки.

Процес затінення методом Монте-Карло розпочинається з випадкового вибору напрямку променів, які виходять з точки зіткнення. Ці випадкові напрямки можуть враховувати різні властивості сцени, такі як розсіювання світла, тіні та відбиття. Затім для кожного випадкового напрямку обчислюється внесок освітленості. Оскільки метод Монте-Карло використовує стохастичний підхід, його сутність полягає в тому, щоб використовувати велику кількість випадкових променів для отримання більш точного результату. Чим більше променів використовується, тим краще моделюється освітлення та властивості матеріалів.

Однією з переваг методу Монте-Карло є його універсальність та здатність апроксимувати різноманітні освітлювальні умови. Використовуючи велику кількість випадкових променів, цей метод може ефективно моделювати різні види ефектів, такі як м'які тіні, відблиски та глобальне освітлення.

Недоліками методу Монте-Карло є високі вимоги до обчислювальних ресурсів через необхідність великої кількості випадкових семплів для досягнення точних результатів. Також, через стохастичний характер методу, результати можуть варіюватися між різними запусками програми.

Назва методу походить від методу Монте-Карло в теорії ймовірностей, де використовується випадкове вибіркове дослідження для наближеного вирішення завдань. У контексті освітлення метод Монте-Карло використовується для оцінки інтегралів освітлення та вирішення рівнянь рендерингу, надаючи статистичний підхід до обробки складних сценаріїв освітлення.

Метод Монте-Карло в освітленні використовує випадковість для точного моделювання поведінки світла в сцені. Замість використання детерміністичних алгоритмів, він використовує випадковий вибір для генерації кількох ймовірних шляхів світла. Шляхом усереднення результатів цих випадкових вибірок метод Монте-Карло виробляє високоякісні зображення з реалістичними ефектами освітлення.

Основна математична модель в Методі Монте-Карло в освітленні включає функції щільності ймовірності (PDF) і техніку інтегрування Монте-Карло. Розглянемо рівняння рендерингу:

$$L_o(p, \omega_o) = L_e(p, \omega_o) + \int_\Omega f(p, \omega_i, \omega_o) \cdot L_i(p, \omega_i) \cdot (\omega_i \cdot N) d\omega_i \quad (1.6)$$

У методі Монте-Карло в освітленні цей інтеграл розв'язується за допомогою формули інтегрування Монте-Карло:

$$\frac{1}{N} \sum_{i=1}^{N} \frac{f(p,w_i,w_o)\, Li(p,w_i)(w_i N)}{p(w_i)} \quad (1.7)$$

Тут N представляє кількість випадкових вибірок. Термін $p(w_i)$ - це функція щільності ймовірності, пов'язана з обраним напрямком w_i. Ділення на $p(w_i)$ забезпечує вагу відповідних вибірок, виправляючи нерівномірний розподіл вибірок.

1.2. Модель освітлення Фонга

Модель освітлення Фонга є однією з ключових концепцій у графіці та комп'ютерному зорі. Розроблена в 1973 році Бліном Фонгом, вона вивчає взаємодію світла та поверхонь для створення реалістичних зображень. Ця модель дозволяє досягати вражаючих візуальних ефектів, забезпечуючи деталізовану симуляцію освітлення на поверхнях об'єктів.

У моделі Фонга враховуються три основні компоненти освітлення: розсіяне, відбите та затемнення. Розсіяне освітлення відповідає за матовий ефект на поверхні. Вона залежить від кута між нормаллю до поверхні та напрямком світла. Відбите освітлення відображає світло від поверхні, узагальнюючи блики. Цей компонент моделює блискучі області на поверхні. Затемнення визначає, наскільки тінь має вплив на конкретну точку поверхні. Головна ідея моделі полягає в тому, що кожен з цих компонентів обчислюється окремо для кожної точки поверхні об'єкта. Комбінація цих компонентів дає повний зовнішній вигляд поверхні. Важливою особливістю моделі Фонга є те, що вона є досить простою та швидкою для обчислення, але в той же час забезпечує хорошу апроксимацію реального освітлення.

Основна ідея застосування моделі Фонга полягає в тому, щоб розрахувати освітленість кожної точки на поверхні об'єкта з урахуванням її орієнтації та відносної позиції до джерела світла. Це дає можливість створювати вражаючі візуальні ефекти, такі як блики, відбиття та тіні. Модель Фонга знаходить широке застосування в галузях від комп'ютерної графіки та візуалізації до анімації та виробництва фільмів.

Необхідно враховувати, що модель Фонга має свої обмеження та недоліки, основним з яких є обчислювальна складність при великій кількості об'єктів і точок. Також вона допускає деякі візуальні артефакти в деяких сценаріях. Однак несмотря на це, модель Фонга залишається важливим інструментом для створення реалістичних зображень в галузі комп'ютерної графіки.

Модель Фонга включає три основні компоненти для симуляції різних аспектів взаємодії світла: фонгівського, розсіяного та дзеркального відбивання.

Фонгівське відбивання є одним з ключових компонентів моделі освітлення Фонга, визначаючи постійне фонове освітлення, яке об'єкт отримує навіть у відсутності прямого світла. Цей аспект грає важливу роль у створенні візуально збалансованих та реалістичних зображень. Фонгівське відбивання виконує важливу функцію в забезпеченні того, що об'єкти не виглядають абсолютно чорними або неприродньо темними у відсутність прямого світла. Цей аспект сприяє створенню природного вигляду сцени, освітлюючи об'єкти навіть у тіньових або менше освітлених областях.

Формула для обчислення фонового освітлення:

$$k_a \cdot I_a \tag{1.8}$$

де:

k_a - коефіцієнт фонгівського відбивання, визначає, наскільки сильно об'єкт реагує на фонове освітлення. Високий коефіцієнт призводить до сильного фонового освітлення, тоді як низький може зробити об'єкт темнішим;

I_a - інтенсивність фонового світла, представляє рівень фонового освітлення.

Велика інтенсивність призводить до яскравого фонового освітлення, що робить об'єкт більш видимим у темних областях.

Розсіяне відбивання в моделі освітлення Фонга є ключовим елементом, що визначає, як матові поверхні взаємодіють із світлом. Цей компонент моделі допомагає створити матовий вигляд об'єктів, де світло розсіюється в різних напрямках, не утворюючи чітких відблисків.

Ключовим фактором у розсіяному відбиванні є дифузна розподіленість світла на поверхні. Матова текстура чи мікроструктура поверхні призводить до того, що світло розсіюється в усіх напрямках. Це створює м'яке та однорідне освітлення, із зменшеним відзеркаленням світла.

Математично, розсіяне відбивання визначається формулою:

$$k_d \cdot I_d \cdot (L \cdot N) \quad (1.9)$$

де:

kd - коефіцієнт розсіяного відбивання, який визначає, наскільки поверхня розсіює світло;

Id - інтенсивність розсіяного світла;

L - нормалізований вектор напрямку світла;

N - нормалізований вектор нормалі поверхні.

Ця формула описує, як світло розсіюється на матовій поверхні в залежності від кута падіння світла та орієнтації поверхні. Розсіяне відбивання взаємодіє з коліром поверхні, впливаючи на те, які частини видимого спектра світла поглинаються та які відбиваються. Крім того, цей ефект визначає м'які тіні та визначає освітлені та тіньові області на поверхні.

У сценаріях 3D-рендерингу, розсіяне відбивання грає ключову роль у створенні природного та матового вигляду матеріалів. Це особливо важливо при рендерингу реалістичних сцен, де важливо передати різні типи матеріалів, такі як тканини, дерево чи камінь.

Дзеркальне відбивання в моделі Фонга є важливим аспектом, що визначає, як гладкі поверхні взаємодіють із світлом. Цей компонент відповідає за створення блискучих точкових віддзеркалень, що спостерігаються на гладких та відбивальних матеріалах. Дзеркальне відбивання виникає від ідеї, що гладкі поверхні, такі як скло чи метал, відображають світло подібно до дзеркала. Вектор віддзеркалення R представляє напрям від точки на поверхні до дзеркально відбитого світла. Дзеркальне відбивання додає блиск і розкіш до зображень, зокрема, коли моделюються гладкі поверхні, такі як скло, метал або вода. Цей компонент особливо ефективний при створенні вражаючих відображень об'єктів, що взаємодіють з оточенням.

У моделі Фонга дзеркальне відбивання визначається формулою:

$$k_s \cdot I_s \cdot (R \cdot V)^n \quad (1.10)$$

де:

k_s - інтенсивність дзеркального світла;

I_s - вектор віддзеркалення світла;

R - вектор віддзеркалення світла;

V - вектор напрямку спостерігача;

n - Фонг-експонента, що впливає на розмір і форму блисків.

Формула Фонга для дзеркального відбивання може бути апроксимована лінійною функцією, яка полегшує обчислення, особливо в реальному часі. Лінійна аппроксимація виглядає наступним чином:

$$k_s \cdot I_s \cdot (R \cdot V) \quad (1.11)$$

Ця апроксимація, хоча менш точна, але більш ефективна для використання у графічних додатках з обмеженими ресурсами. Фонг-експонента (n) у формулі дзеркального відбивання визначає форму та розмір блисків. Великі значення n призводять до менших, але більш концентрованих блисків, тоді як менші значення роблять блиски ширшими та розсіянішими.

Повне рівняння освітлення в моделі Фонга об'єднує фонгівське, розсіяне та дзеркальне відбивання для визначення освітлення поверхні в конкретній точці в просторі. Це рівняння враховує взаємодію світла з матеріалом об'єкта та формує остаточний колір цієї точки.

Повне рівняння освітлення Фонга:

$$I_p = k_a i_a + \sum_{m \in lights} (k_d (\widehat{L_m} \widehat{N}) i_{m,d} + k_s (\widehat{R_m} \widehat{V})^a i_{m,s}) \quad (1.12)$$

де:

k_a – коефіцієнт оточуючого відбиття;

k_d – коефіцієнт розсіючого відбиття;

k_s – коефіцієнт відблискуючого відбиття;

α – коефіцієнт блиску;

$\hat{L}_m$ – вектор, направлений від поверхні освітлюваного тіла до джерела світла відбиття;

$\hat{N}$ – нормаль до поверхні освітлюваного тіла;

$\hat{V}$ – вектор, напрямлений від поверхні освітлюваного тіла до віртуального спостерігача;

$\hat{R}_m$ – відбитий поверхнею промінь світла. Він також розраховується за наступною формулою:

$$\hat{R}_m = 2\,(\hat{L}_m * \hat{N})\hat{N} - \hat{L}_m \qquad (1.13)$$

Це рівняння динамічно розраховує інтенсивність світла I_p в конкретній точці поверхні, надаючи всебічне та реалістичне представлення того, як світло взаємодіє з матеріалами. Модель Фонга знаходить широке застосування в графіці комп'ютерів, геймінгу та анімації. Її універсальність дозволяє представляти різні матеріали, від матових до високоякісних поверхонь. Хоча існують більш вдосконалені моделі, такі як Блінна-Фонга та Кук-Торранса, модель Фонга залишається фундаментальним та доступним вибором для створення реалістичних сцен.

1.3. Додаткові методи оптимізації

1.3.1. Використання буферів глибини

Буфер глибини є важливою складовою 3D-рендерингу, визначаючи видимість об'єктів та оптимізуючи процес створення зображень. Кожен піксель на екрані має своє значення глибини, визначаючи його відстань від камери. Це значення порівнюється з тим, що вже знаходиться в буфері глибини, і, якщо нове значення менше, піксель відображається, в іншому випадку - приховується.

Використання буферів глибини є ключовим елементом оптимізації рендерінгу освітлення в графіці та комп'ютерному зорі. Буфери глибини

представляють собою спеціальні структури даних, які використовуються для зберігання глибини кожної точки у тривимірному просторі на площині зображення.

Головна мета використання буферів глибини - забезпечити ефективний та швидкий процес визначення, які об'єкти перебувають на передньому плані, а які - на задньому, з огляду на перспективу та освітлення сцени. Кожній точці на екрані присвоюється значення глибини, яке визначає відстань від камери до об'єкта, який знаходиться в цій точці. Використання буферів глибини дозволяє ефективно визначати, який об'єкт повинен бути відображений у випадку перекриття поверхонь. Замість рендерингу всіх об'єктів у сцені на кожному кадрі, система може визначати глибину кожної точки та обирати лише той об'єкт, який перебуває передніше у зоровому полі глядача. Це робить можливим значне зменшення кількості обчислень, необхідних для візуалізації сцени.

Використання буферів глибини також сприяє уникненню обчислень освітлення для невидимих поверхонь, що знаходяться за іншими об'єктами. Таким чином, система може концентрувати обчислення освітлення лише на видимих областях, що робить процес рендерінгу значно ефективнішим. Однак важливо враховувати, що використання буферів глибини також може призводити до деяких артефактів, зокрема, якщо об'єкти мають тіні або освітлення, що додає складнощі в обчисленнях. Також важливо правильно налаштувати буфери глибини для уникнення проблем із затемненням та іншими артефактами.

Загалом, використання буферів глибини є важливою оптимізацією в рендерінгу освітлення, що дозволяє отримувати реалістичні та ефективні зображення за більш короткий час. Цей підхід знаходить широке застосування у відеографіці, комп'ютерних іграх, візуальнізації та інших областях, де важлива швидкість та ефективність візуалізації сцен.

Буфер глибини має кілька основних функцій. Він забезпечує коректне визначення видимості об'єктів, визначаючи, які знаходяться на передньому та які - на задньому плані. Це важливо для створення реалістичних сцен. Оптимізація

рендерингу досягається завдяки використанню буфера глибини, що дозволяє виключити обчислення освітлення та текстур для прихованих пікселів.

Існують різні алгоритми буфера глибини, такі як Z-буфер та W-буфер, а також метод двійного буфера глибини, який дозволяє розподілити роботу між кадрами. Застосування буфера глибини також вирішує проблеми зображення, такі як артефакти зіскокування та низька точність глибини.

1.3.2. Z-буфер

Використання Z-буфера є важливою оптимізацією в процесі рендерінгу освітлення в графіці та комп'ютерному зорі. Z-буфер (або буфер глибини) є особливим буфером в графічному процесорі, який відповідає за зберігання глибини кожного пікселя на екрані.

Основна ідея полягає в тому, щоб для кожної точки на екрані зберігати відстань від камери до цієї точки у Z-буфері. При рендерінгу сцени, система порівнює глибину кожного нового пікселя зі значенням, яке вже знаходиться в Z-буфері. Якщо нова глибина менша, система визначає, що новий піксель перебуває передніше і замінює значення в Z-буфері. Це дозволяє уникнути обчислень та візуалізації об'єктів, які знаходяться за іншими об'єктами. Тобто, якщо новий об'єкт перекриває частину іншого, лише видима частина нового об'єкта буде відображена, а частина, яка залишається за іншим об'єктом, буде відкинута. Використання Z-буфера спрощує реалізацію алгоритмів рендерінгу, оскільки він автоматично вирішує проблему перекриття поверхонь. Системі не потрібно вручну обчислювати, який об'єкт повинен бути відображений у випадку перекриття. Завдяки Z-буферу зменшується кількість необхідних обчислень та спрощується програмування рендерінгу. Також він дозволяє реалізувати реалістичну глибину в зображеннях, що є важливим для досягнення відчуття глибини та реалізму в графіці.

Збереження інформації про глибину для кожного пікселя також дозволяє оптимізувати обчислення освітлення та тіней. Система може ігнорувати

обчислення для невидимих областей, зменшуючи обсяг роботи і підвищуючи продуктивність.

Загалом, використання Z-буфера є ефективним засобом оптимізації рендерінгу освітлення, забезпечуючи точність та продуктивність візуалізації тривимірних сцен.

Математично цей процес може бути виражений наступним чином:

$$Z_{\text{пікселя}}(x, y) = \min(Z_{\text{пікселя}}(x, y), Z_{\text{новий\_піксель}}(x, y)) \quad (1.14)$$

Розглянемо сцену з двома об'єктами, A та B, які розташовані на різних глибинах. Нехай Z_A та Z_B - глибини цих об'єктів. При використанні Z-буфера для кожного пікселя обчислюється нова глибина $Z_{\text{новий\_піксель}}$.

Якщо $Z_{\text{новий\_піксель}} < Z_{\text{пікселя}}$, то цей піксель буде відображений.

Такий підхід дозволяє коректно визначати видимість об'єктів на сцені і забезпечує правильний порядок їх відображення.

З-буфер гарантує правильний визначення порядку відображення об'єктів на сцені, уникуючи артефактів інтерференції. За допомогою Z-буфера можливе значне зменшення обчислень для невидимих пікселів, оптимізуючи процес рендерингу.

Додаткові алгоритми та оптимізації, такі як динамічне оновлення та метод передбачення, дозволяють підтримувати актуальність Z-буфера в реальному часі та прискорювати роботу з великою кількістю об'єктів на сцені. Проте, проблеми, такі як Z-файтинг та точність глибини, вимагають уважного контролю та використання додаткових оптимізацій.

Z-буфер виявляється потужним інструментом в руках графічних розробників, забезпечуючи не тільки правильне відображення об'єктів на сцені, але й значне прискорення процесу рендерингу. Його математика та алгоритми стали ключовими елементами в графічному програмуванні, дозволяючи отримувати вражаючі та оптимізовані 3D-зображення.

1.3.3. W-буфер

Використання W-буфера в рендерінгу освітлення є ефективною стратегією оптимізації, спрямованою на збереження та опрацювання інформації про глибину об'єктів у тривимірному просторі. W-буфер, як і Z-буфер, представляє собою спеціальний буфер, який використовується для зберігання значень глибини об'єктів на екрані під час рендерінгу.

Основна концепція W-буфера полягає в тому, щоб для кожного пікселя екрану зберігати координату W, що вказує на глибину пікселя в просторі. Зазвичай W-координата визначається як обернена до Z-координати (1/Z), де Z визначає глибину. Збереження W-координати в кожному пікселі дозволяє здійснювати порівняння глибин та здійснювати обчислення для визначення того, який об'єкт перебуває на передньому плані. Під час рендерінгу, система порівнює W-координату нового пікселя зі значенням, що вже знаходиться в W-буфері. Якщо нова W-координата менша, система визначає, що новий піксель перебуває передніше і замінює значення в W-буфері.

Використання W-буфера подібно до Z-буфера, але з деякими перевагами. Він забезпечує більш точне подання глибини об'єктів, оскільки зберігає не тільки саму глибину, але із зміненою просторовою перспективою (W-координатою). Це робить його особливо ефективним для обробки перспективної глибини у тривимірних сценах.

Використання W-буфера є частиною загальної стратегії оптимізації для зменшення обчислень та оптимізації візуалізації тривимірних сцен. Цей метод дозволяє системам рендерінгу більш ефективно визначати порядок відображення об'єктів, що забезпечує більш швидкий та ефективний рендерінг.

Теорія W-буфера базується на ідеї використання четвертої координати (координати W), яка додається до векторів в сцені для перетворення у просторі зображення. Глибина кожного пікселя тепер обчислюється як відношення глибини до координати W, що дозволяє враховувати перспективні ефекти та реалістично моделювати сцени.

Математично W-буфер виражається наступним чином:

$$W_{\text{пікселя}}(x, y) = \min(W_{\text{пікселя}}(x, y), W_{\text{новий\_піксель}}(x, y)) \quad (1.15)$$

де:

$W_{\text{пікселя}}$ - існуюче значення ваги в W-буфері;

$W_{\text{новий\_піксель}}$ - нове значення ваги для розглянутого пікселя.

Такий підхід дозволяє ефективно визначати видимість пікселів у сцені. W-буфер застосовується в широкому спектрі графічних застосувань, де важливо точно визначати прозорість об'єктів та забезпечувати вірне відображення сцени. Він є популярним інструментом в галузі комп'ютерної графіки та візуалізації даних.

Однією з основних переваг W-буфера є його здатність ефективно вирішувати проблеми, пов'язані з прозорістю. Коли об'єкти частково перекривають один одного або мають прозорі елементи, W-буфер може точно визначити, який піксель повинен бути видимим, забезпечуючи правильне наложення та прозорість.

Додатково W-буфер дозволяє ефективно керувати прозорістю об'єктів на сцені. Вага W використовується для визначення ступеня прозорості, де менше значення W вказує на більшу прозорість. Це важливо для реалістичного відображення матеріалів з прозорими частинами, такими як вода чи скло.

W-буфер виявляється особливо корисним в сучасних застосунках, де реалізація складних ефектів світла та матеріалів вимагає точного визначення видимості об'єктів. Його використання дозволяє отримувати деталізовані та реалістичні зображення зі збереженням оптимальності в роботі з глибинами та вагами об'єктів.

Проте, як і з будь-яким методом, пов'язаним із зображенням, існують певні виклики та проблеми. Наприклад, додаткові обчислення, пов'язані з використанням W-буфера, можуть призводити до певного збільшення

обчислювального навантаження. Також важливо правильно керувати значеннями ваги для уникнення артефактів та некоректного відображення.

У підсумку, W-буфер представляє сучасний та потужний інструмент у світі 3D-рендерингу. Він дозволяє ефективно вирішувати проблеми прозорості, забезпечуючи точне та реалістичне відображення сцени, та ставиться в основу численних графічних застосувань, від відеоігор до комп'ютерної візуалізації даних.

1.3.4. Метод двійного буфера глибини

Використання методу двійного буфера глибини (DBB) є важливою стратегією оптимізації в області рендерінгу освітлення в комп'ютерній графіці. DBB визначається наявністю двох буферів глибини: переднього та заднього. Цей метод спрямований на поліпшення ефективності рендерінгу та зменшення артефактів, пов'язаних з видимістю та глибиною об'єктів.

В основі DBB лежить ідея використання двох буферів глибини для альтернативного запису глибини під час обчислень. Під час кожного кадру використовуються обидва буфери, причому один використовується для запису нових значень глибини, а інший залишається незмінним. Після завершення кожного кадру, передній та задній буфери обмінюються ролями, і процес повторюється. Ця стратегія робить можливим уникнення конфліктів при паралельному рендерінгу, коли паралельно відбуваються зчитування та запис в один буфер глибини. Завдяки ротації буферів після кожного кадру, забезпечується консистентність даних та уникнення колізій при взаємодії з об'єктами на сцені.
Використання DBB дозволяє ефективно контролювати доступ до буфера глибини, зменшуючи можливість помилок та артефактів, пов'язаних із зчитуванням та записом даних в один і той самий буфер одночасно. Це особливо корисно в реальних застосунках, де велика кількість об'єктів може впливати на видимість та глибину.

Такий підхід стає необхідним у великих та складних тривимірних сценах, де велика кількість об'єктів взаємодіє між собою. DBB є частиною стратегій оптимізації для забезпечення високої ефективності рендерінгу та оптимізації відображення тривимірних сцен.

Математично метод двійного буфера глибини виражається наступним чином:

$$Z_{\text{поточний}}(x, y) = \min(Z_{\text{поточний}}(x, y), Z_{\text{новий}}(x, y)) \quad (1.16)$$

де

$Z_{\text{поточний}}$ - глибина пікселя в поточному кадрі;

$Z_{\text{новий}}$ - нове значення глибини для розглянутого пікселя у наступному кадрі.

Метод двійного буфера глибини використовується для вирішення проблеми взаємодії прозорих та непрозорих об'єктів на сцені. При використанні лише одного буфера глибини виникає проблема з прозорістю, коли непрозорий об'єкт перекривається прозорим і тіні виводяться некоректно. Метод двійного буфера глибини дозволяє уникнути цієї проблеми, використовуючи відокремлені буфери для рендеренгу непрозорих і прозорих об'єктів.

Іншою важливою особливістю цього методу є ефективне використання асинхронності рендеренгу. Оскільки один буфер використовується для поточного кадру, а інший - для наступного, рендеринг нового кадру може початися, навіть якщо попередній ще не закінчено. Це дозволяє забезпечити плавніше відображення сцени, зменшуючи можливі затримки.

Приклад використання методу двійного буфера глибини може бути ілюстрований в сцені з прозорими об'єктами, такими як вікна будівель чи стекляні предмети. Один буфер використовується для збереження глибини непрозорих елементів, а інший - для прозорих. Це дозволяє забезпечити правильне наложення тіней та кольорів, що призводить до більш точного та реалістичного відображення.

Метод двійного буфера глибини також знайшов широке застосування в галузі відображення віртуальної реальності (VR) та аугментованої реальності (AR). У таких системах важливо забезпечити низьку затримку та високу якість зображення для покращеного користувацького досвіду.

Важливою перевагою методу двійного буфера глибини є його висока ефективність та відмінна сумісність з різноманітними видами сцен та об'єктів. Він дозволяє досягти високої якості графіки при збереженні оптимальності у використанні ресурсів. Такий метод стає ключовим компонентом для сучасних систем візуалізації, де важливо забезпечити оптимальне та реалістичне відображення 3D-сцен.

2 РОЗРОБКА МЕТОДУ ОПТИМІЗОВАНОГО РЕНДЕРІНГУ ДИНАМІЧНОГО ОСВІТЛЕННЯ ТА АТМОСФЕРНИХ ЕФЕКТІВ В 3D ІГРАХ

2.1 Розрахунок інтегральних можливостей комп’ютера

Визначення потужності комп'ютера в рендерінгу освітлення в 3D іграх має критичне значення для створення іммерсивного та реалістичного геймплею. 3D ігри в сучасному світі стають все більш складними та деталізованими, завдяки чому вимагають великої обчислювальної потужності для ефективного відтворення складних графічних сцен та освітлення.

Однією з ключових складових візуального досвіду в 3D іграх є освітлення. Правильно відтворене освітлення додає глибину, тіні, відблиски та інші ефекти, що роблять ігровий світ більш реалістичним та захоплюючим. Проте для досягнення високої якості освітлення потрібна потужна обчислювальна система.

У світі геймінгу визначення потужності комп'ютера впливає на гладкість гри та рівень деталізації графічних об'єктів. Чим потужніший графічний процесор та центральний процесор, тим більше деталей та ефектів освітлення можна відтворити в реальному часі. Гравець може насолоджуватися динамічним освітленням, реалістичними тінями та вражаючими візуальними ефектами, які створюють неперевершений геймплей.

Рендерінг освітлення в 3D іграх вимагає великої обчислювальної потужності через використання різних технологій та ефектів. Реалістичне освітлення включає в себе високодеталізовані тіні, відблиски, ефекти рефлексії та освітлення окружаючого середовища. Все це потребує значних обчислень, і тут важлива роль належить потужності графічної карти та її здатності до прискорення графічних обчислень.

Процес визначення потужності комп'ютера в рендерінгу освітлення включає в себе оцінку графічних параметрів, таких як роздільна здатність екрану, частота кадрів на секунду, якість текстур та інші графічні налаштування. Ці параметри

безпосередньо впливають на спроможність комп'ютера створювати вражаючі візуальні ефекти та підтримувати високий рівень геймплею.

Однак важливість визначення потужності комп'ютера в рендерінгу освітлення в 3D іграх не обмежується лише графічним аспектом. Задіяність процесора та графічної карти також визначається у сферах фізичної моделювання, штучного інтелекту, звукового супроводу та інших аспектах геймплею. Такий комплексний підхід до визначення потужності робить її ключовим аспектом створення конкурентоспроможних та захоплюючих ігор.

Загалом, визначення потужності комп'ютера в рендерінгу освітлення в 3D іграх є вирішальним для створення вражаючого та реалістичного геймплею. Висока обчислювальна потужність, ефективні графічні технології та здатність до прискорення графічних обчислень роблять можливим відтворення вражаючих візуальних ефектів у реальному часі. Це не лише підвищує рівень геймплею, але й робить ігри більш захоплюючими та деталізованими для кожного гравця.

Розрахунок можливостей комп'ютера включає оцінку трьох основних компонентів: центрального процесора (CPU), відеокарти (GPU) та оперативної пам'яті (RAM). Подана теорія розкриє основні поняття та методи оцінки продуктивності кожного з цих елементів.

2.1.1 Розрахунок інтегральних можливостей комп’ютера

Процесор, який є центральною частиною обчислювального простору в комп'ютері, відіграє ключову роль у виконанні складних обчислювальних завдань, таких як рендерінг освітлення в графіці та візуалізації. Важливість процесора в цьому контексті важко переоцінити, оскільки від його характеристик і продуктивності залежить здатність комп'ютера ефективно обробляти графічні дані та генерувати реалістичні зображення.

Сучасні графічні застосунки та ігри ставлять високі вимоги до обчислювальної потужності, і процесор має бути достатньо потужним, щоб впоратися з цими завданнями. Однією з важливих функцій процесора є виконання

алгоритмів, які визначають розподіл світла, тіні та інших атмосферних ефектів на поверхні об'єктів у віртуальному просторі.

Обчислювальна потужність процесора визначає його здатність ефективно виконувати графічні операції. Чим вища обчислювальна потужність, тим швидше можуть виконуватися розрахунки, пов'язані з рендерінгом. Це стає критичним у випадках, коли потрібно взаємодіяти з великою кількістю світлових джерел, обчислювати тіні, а також враховувати реалістичні властивості матеріалів та їх взаємодію із світлом.

У рендерінгу освітлення важливою стає також підтримка різноманітних інструкційних наборів процесором. Сучасні графічні технології використовують оптимізовані інструкції для покращення продуктивності та забезпечення швидкого виконання операцій рендерінгу.

Мнозадачність також є ключовою характеристикою процесора в контексті графічного рендерінгу. Можливість виконання паралельних завдань дозволяє ефективно обробляти великий обсяг даних, такий як інформація про кожен піксель зображення чи розрахунки для кожного світлового променя. Це особливо важливо при застосуванні методів реалістичного трасування променів, які вимагають великої кількості паралельних обчислень.

Однією з важливих властивостей процесора є також можливість ефективно взаємодіяти з графічними картами (GPU), які часто використовуються в сучасних системах для виконання графічних обчислень. Висока сумісність процесора з графічними картами дозволяє ефективно розподіляти завдання між цими компонентами та підвищує загальну продуктивність системи.

Важливість процесора в рендерінгу освітлення проявляється не лише в його технічних характеристиках, але і в здатності ефективно взаємодіяти з іншими компонентами комп'ютера. Наприклад, висока швидкість передачі даних між процесором та пам'яттю важлива для швидкого доступу до необхідної інформації, а сполучення процесора з оптимізованими алгоритмами рендерінгу робить його незамінним елементом у створенні вражаючих візуальних ефектів.

Висновок полягає в тому, що процесор є важливим компонентом у рендерінгу освітлення, а його характеристики визначають швидкість і якість графічного відтворення. Сучасні графічні технології висувають високі вимоги до обчислювальної потужності, паралельності та оптимізації роботи з іншими компонентами, щоб надати користувачеві реалістичні та захоплюючі візуальні враження.

Процесор є центральним елементом обчислювальної системи, відповідальним за виконання різноманітних завдань. У випадку рендерінгу освітлення, процесор використовується для попередніх обчислень, які визначають, як світло розподіляється на поверхнях об'єктів. Важливо мати потужний процесор, здатний ефективно виконувати обчислення освітлення великих сцен із складними геометричними структурами.

Сучасні мультимедійні процесори, такі як Intel Core і AMD Ryzen, забезпечують велику кількість обчислювальних ядер, які можуть працювати паралельно. Це особливо корисно для рендерінгу, де можна розділити завдання на багато менших частин і обчислювати їх одночасно. Також важливо враховувати тактильну частоту процесора, оскільки вона визначає швидкість обчислення на одному ядрі. Висока частота корисна для завдань, які не можуть бути ефективно розпаралельнені.

Розрахунок можливостей процесору включає оцінку декількох ключових параметрів, які визначають його продуктивність.

Тактова частота вимірюється в гігагерцах і вказує, скільки циклів обчислень процесор може виконати за одну секунду. Вища тактова частота зазвичай означає більшу швидкість обробки і кращу продуктивність. Розрахувати її можна через командний рядок, використовуючи, наприклад, команду **wmic cpu get maxclockspeed**.

Кількість ядер та потоків визначає можливості процесора паралельного розрахунку математичних операцій. Багатоядерні процесори можуть обробляти кілька завдань паралельно, покращуючи продуктивність у мультитаскінгу.

Визначити кількість ядер та потоків можна, використовуючи, наприклад, команду **wmic cpu get numberofcores, numberoflogicalprocessors**.

Кеш-пам'ять допомагає прискорити доступ до часто використовуваних даних. Розрахувати розмір кеш-пам'яті можна за допомогою команди **wmic cpu get L2CacheSize, L3CacheSize**.

Архітектура процесора (наприклад, x86, x64) та набір інструкцій впливають на його сумісність та продуктивність. Визначити архітектуру можна за допомогою команди **wmic cpu get caption**. Щодо набору інструкцій, використовуйте команду **wmic cpu get InstructionSet**.

Оцінка можливостей процесора важлива для вибору методу рендерінгу освітлення. Використання командного рядка та інших інструментів дозволяє отримати докладну інформацію про характеристики процесора та його ефективність в різних сценаріях використання.

Продуктивність процесора може бути оцінена за допомогою простої формули, яка враховує тактову частоту, кількість ядер, кількість потоків і кеш-пам'ять. Враховуючи, що ці параметри впливають на продуктивність в різних пропорціях, можна скористатися наступною формулою:

$$\text{Продуктивність} = \text{Тактова частота} \times \left(\text{Кількість ядер} + \frac{\text{Кількість потоків}}{2}\right) \times \text{Кеш-пам'ять} \quad (2.1)$$

де:

тактова частота - частота роботи процесора, вимірювана у GHz,

кількість ядер - вказує на загальну кількість фізичних ядер у процесорі,

кількість потоків - вказує на загальну кількість потоків, які може обробляти процесор (враховуючи потоки, які обробляються гіперпотоками),

кеш-пам'ять - вказує на загальний обсяг кеш-пам'яті, доступної для процесора.

Ця формула є спрощеною та не враховує різні аспекти архітектури процесора, такі як оптимізація коду, розрядність інструкцій і т. д. Однак вона може служити загальним показником продуктивності на базовому рівні.

2.1.2 Відеокарта

В сучасному світі комп'ютерної графіки та візуалізації, відеокарта (GPU) стала вирішальним компонентом для досягнення високоякісного рендерінгу освітлення в 3D сценах. Важливість відеокарти у рендерінгу освітлення визначається її здатністю обробляти та відтворювати графічні обчислення швидко та ефективно. Ось докладніше про кілька ключових аспектів, що підкреслюють важливість відеокарти в контексті рендерінгу освітлення.

Паралельна обробка: Відеокарти спроектовані для паралельної обробки великої кількості обчислень. Основні функції графічного процесора орієнтовані на одночасне виконання багатьох завдань, що забезпечує високу продуктивність при обробці складних алгоритмів рендерінгу освітлення.

Shader-програмування: Можливість програмувати шейдери на відеокарті дозволяє розробникам створювати складні візуальні ефекти та обробляти освітлення у реальному часі. Shader-програмування дозволяє налаштовувати вигляд матеріалів, взаємодію світла та тіней, що допомагає досягти реалістичного зображення.

Обчислення освітлення: Відеокарти володіють спеціалізованими обчислювальними блоками, які призначені для виконання складних обчислень, пов'язаних з розрахунками освітлення. Це включає в себе трасування променів, відбивання світла, тінювання та інші аспекти глобального освітлення.

Рендерінг текстур та мапування: Відеокарти дозволяють використовувати великі текстури для реалістичного зображення об'єктів у сцені. Мапування текстур та їх рендерінг високою швидкістю визначають кінцевий вигляд об'єктів та поверхонь.

GPU-прискорення відображення: Відеокарти забезпечують GPU-прискорення, що сприяє покращенню продуктивності при відображенні графіки. Це особливо важливо в сучасних вимогливих графічних додатках та іграх, де великі обсяги даних повинні бути оброблені швидко.

Відтворення в реальному часі: Завдяки високій швидкості обчислення та рендерінгу, відеокарти дозволяють відтворювати складні сцени в реальному часі. Це особливо важливо в ігровій індустрії та візуалізації, де важлива швидкість та реактивність графічних обчислень.

VR та AR додатки: У віртуальній реальності (VR) та доповненій реальності (AR) важливість відеокарти ще більше зростає. Вони використовуються для забезпечення високоякісного відображення та плавного взаємодії з віртуальним або доповненим оточенням.

Узагальнюючи, відеокарта є важливим елементом для досягнення високоякісного рендерінгу освітлення у сучасних графічних застосунках. Її паралельна обробка, програмування шейдерів, обчислювальні можливості та інші функції роблять її ключовим компонентом для досягнення реалістичного візуального враження.

Відеокарта є ключовим компонентом для обробки графічних даних та забезпечення високоякісного відображення графіки на екрані. Для розрахунку можливостей відеокарти важливо враховувати кілька ключових параметрів, які визначають її продуктивність та здатність обробляти графічні завдання.

CUDA-ядра є основною одиницею обчислення на відеокарті NVIDIA. Чим більше CUDA-ядер, тим більше завдань вона може обробляти паралельно. Розрахувати кількість CUDA-ядер можна використовуючи команду командного рядка Windows: **wmic path win32_videocontroller get caption, adapterram, maxclockspeed, numproc**.

Тактова частота GPU визначає швидкість обробки графічних даних. Вища тактова частота дозволяє відеокарті працювати швидше. Розрахувати тактову частоту можна за допомогою команди **nvidia-smi** або інших утиліт для моніторингу GPU.

Обсяг відеопам'яті важливий для обробки великих обсягів графічних даних. Розрахувати обсяг відеопам'яті можна також через командний рядок: wmic path **win32_videocontroller get caption, adapterram**.

Ширина шини пам'яті вказує на кількість бітів, які відеокарта може передавати за один обмін. Більша ширина шини пам'яті дозволяє ефективніше передавати дані. Розрахувати цей параметр можна за допомогою команди **nvidia-smi** або програм для моніторингу GPU.

Продуктивність відеокарти можна оцінити за допомогою формули, яка враховує кількість CUDA-ядер, тактову частоту GPU, обсяг відеопам'яті та ширину шини пам'яті.

Ось спрощена формула для розрахунку продуктивності:

$$\text{Продуктивність} = \text{Кількість CUDA-ядер} \times \text{Тактова Частота GPU} \times \text{Обсяг Відеопам'яті} \times \text{Ширина Шини Пам'яті} \quad (2.2)$$

де:

кількість CUDA-ядер - вказує на кількість обчислювальних ядер відеокарти;

тактова Частота GPU - визначається в GHz і вказує на швидкість обробки графічних даних;

обсяг відеопам'яті - вимірюється в GB і показує, скільки даних відеокарта може утримувати одночасно;

ширина шини пам'яті - вказує на кількість бітів, які відеокарта може передавати за один обмін пам'яті.

Ця формула є загальною та не враховує ряд додаткових факторів, таких як архітектура відеокарти, технології оптимізації, інструкції та інші аспекти, але допоможе у попередньому аналізі та виборі методів розрахунку рендерінгу освітлення.

2.1.3 Оперативна пам'ять

Оперативна пам'ять є важливим фактором у процесі рендерінгу освітлення в комп'ютерній графіці і візуалізації 3D сцен. Цей аспект грає критичну роль у високоефективних системах рендерінгу та графічних застосунках, де великі

обсяги даних повинні оброблятися швидко та ефективно для отримання якісних візуальних результатів.

Оперативна пам'ять, або RAM (Random Access Memory), є однією з ключових компонентів комп'ютерної системи, і її роль особливо важлива в графічних додатках, де обчислення та зберігання великої кількості даних в реальному часі є необхідними. У рендерінгу освітлення, де обробка складних алгоритмів, обчислення великої кількості векторів та зображення складних ефектів відбивання та тіней вимагають великих обсягів пам'яті, швидкість та доступність RAM стає критичним фактором.

Один з аспектів важливості оперативної пам'яті в рендерінгу освітлення полягає у здатності зберігати та маніпулювати об'ємними даними, такими як текстури, моделі та трансформації. Швидкий доступ до цих даних дозволяє ефективно виконувати різноманітні операції, такі як визначення положення об'єктів, обчислення освітлення та застосування матеріалів на поверхні об'єктів.

Оптимізований доступ до пам'яті забезпечує ефективне використання ресурсів та покращує продуктивність рендерінгу.

Іншим важливим аспектом є можливість використовувати оперативну пам'ять для зберігання проміжних результатів обчислень, що є важливим у випадках складних алгоритмів рендерінгу, таких як трасування променів чи обчислення глобального освітлення. Це дозволяє зменшити навантаження на центральний процесор (CPU) та графічний процесор (GPU), що може позитивно позначитися на продуктивності.

Оперативна пам'ять також використовується для зберігання та обробки даних про текстури. Великі текстури, які використовуються для створення реалістичних поверхонь об'єктів, вимагають значних обсягів пам'яті для своєї коректної обробки. Швидкий доступ до текстурної інформації сприяє зменшенню часу, необхідного для завантаження та обробки зображень.

У сучасних системах рендерінгу також важливо враховувати аспекти оптимізації використання пам'яті, такі як кешування даних та ефективне використання

буферів. Це допомагає уникнути затримок у відтворенні графіки та покращує загальну ефективність рендерінгу.

Також, важливо зазначити, що в сучасних системах велика кількість оперативної пам'яті часто використовується для запуску паралельних операцій, таких як одночасне виконання багатьох рендерінгових задач або одночасна

Для оцінки можливостей RAM важливо враховувати декілька параметрів, які визначають її продуктивність та здатність працювати з додатками та завданнями.

Обсяг оперативної пам'яті визначається в гігабайтах (ГБ) і вказує, скільки даних може утримувати RAM одночасно. Більший обсяг RAM дозволяє використовувати більше програм та даних одночасно. Щоб дізнатися обсяг пам'яті, використовуйте команду wmic memorychip get capacity.

Тактова частота RAM визначає швидкість передачі даних між оперативною пам'яттю та процесором. Вимірюється в мегагерцах (МГц). Чим вища тактова частота, тим швидше можна передавати дані. Дізнатися цей параметр можна через команду wmic memorychip get speed.

Кількість каналів вказує на кількість шляхів, якими дані можуть передаватися між пам'яттю та процесором. Більше каналів може покращити швидкодію. Використовуйте команду wmic memorychip get memorychannel для дізнання кількості каналів.

Продуктивність оперативної пам'яті можна оцінити за допомогою формули, яка враховує обсяг пам'яті, тактову частоту RAM, тип та покоління пам'яті, та кількість каналів. Спрощена формула для розрахунку продуктивності оперативної пам'яті:

$$\text{Продуктивність} = \text{Обсяг Пам’яті} \times \text{Тактова Частота RAM} \times \text{Кількість Каналів} \quad (2.3)$$

2.2 Опис алгоритму методу

За визначеною метою та завданням роботи було розроблено алгоритм оптимізованого рендерінгу освітлення та його блок-схема.

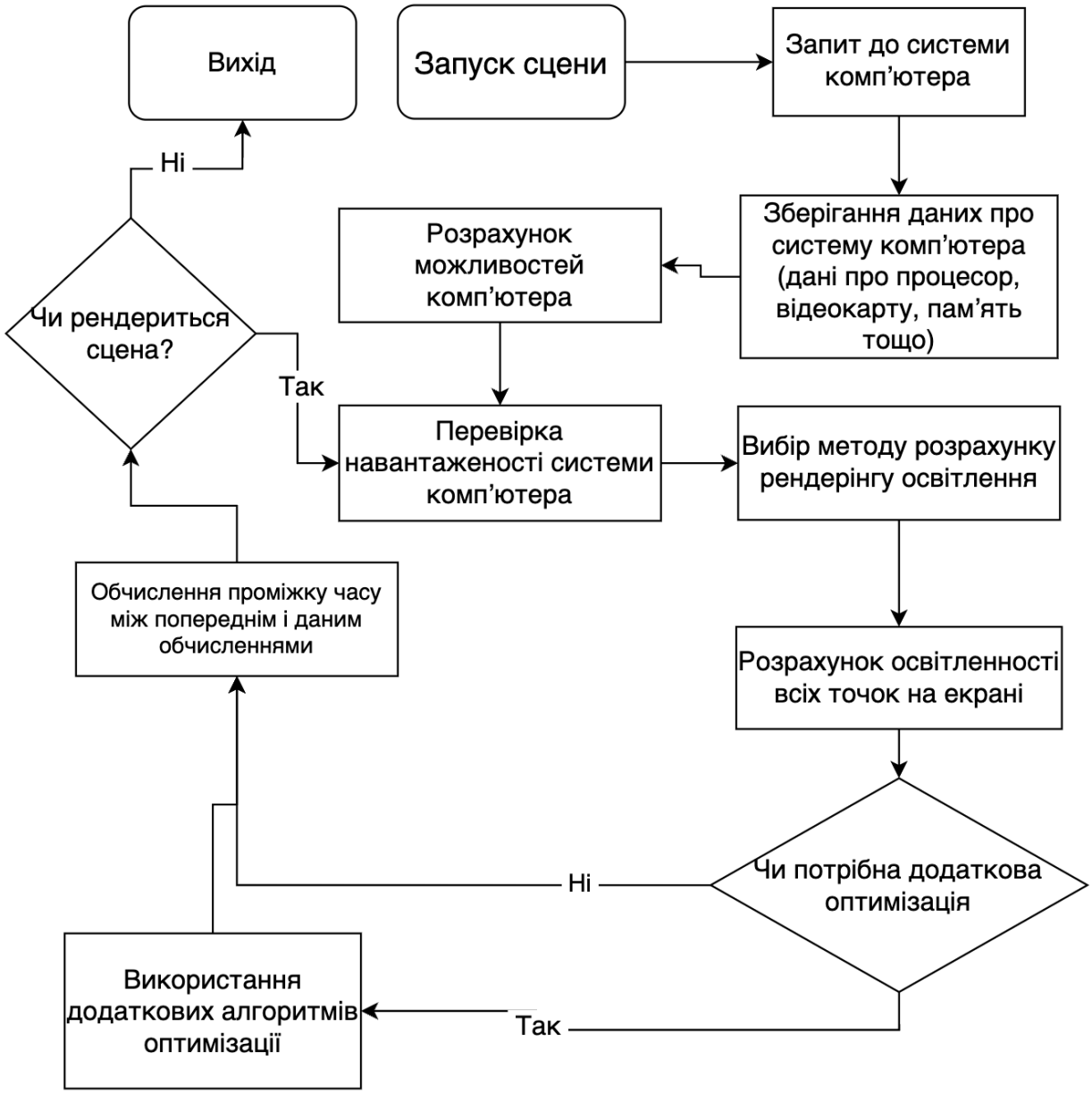


Рис. 2.1 Блок-схема алгоритму оптимізованого рендерінгу освітлення

Першим і вирішальним етапом алгоритму є попереднє визначення характеристик системи комп'ютера, які суттєво впливають на ефективність рендерінгу освітлення. Ці характеристики визначаються трьома основними

компонентами - центральним процесором (процесором), графічним процесором (відеокартою) та оперативною пам'яттю.

Далі проводиться обчислення продуктивності кожного з цих компонентів, яке виражається числовим значенням. Ці значення визначають базові методи розрахунку, за якими будуть виконуватись подальші обчислення променів від джерел освітлення. Кожному компоненту призначається ваговий коефіцієнт в залежності від його внеску у рендерінг.

Далі алгоритм визначає навантаженість кожного компонента, враховуючи його продуктивність та інші фактори, що можуть впливати на швидкість обчислень. Такий підхід дозволяє оптимально розподілити завдання між процесором, відеокартою та оперативною пам'яттю, максимізуючи загальну продуктивність.

Далі комп'ютер розпочинає цикл рендерінгу, де проводяться калькуляції освітлення для всіх точок сцени відповідно обраними методами. Цей етап включає в себе ітерації для кожного кадру гри, де величина залежить від введеного користувача або динамічно міняється у залежності від руху об'єктів на сцені.

Одним з важливих кроків є використання оптимізаційних алгоритмів, призначених для зменшення кількості обчислень на невидимих частинах сцени. Ці алгоритми виявляють області, які не видно гравцеві, та розраховують освітлення лише для видимих елементів. Це дозволяє розвантажити систему та значно зменшити навантаження на компоненти.

Цей оптимізований підхід дозволяє уникнути надмірної кількості обчислень та забезпечити гладку та ефективну роботу гри. Крім того, під час першої ітерації рендерінгу визначається час між кадрами, щоб переконатися, що комп'ютер має достатню потужність для обробки всіх обчислень встановленою швидкістю кадрів на секунду.

Визначивши загальну продуктивність системи, алгоритм на наступних ітераціях робить висновки про необхідність використання конкретних методів рендерінгу освітлення та ефективності оптимізаційних алгоритмів. Цей

ітеративний підхід дозволяє адаптуватися до змінних умов і забезпечує високу якість геймплею при оптимальному використанні ресурсів комп'ютера.

2.3 Опис розроблених класів

Для початку було вирішено почати з розробки класів для обчислення інтегральних можливостей комп’ютера.

Клас **CpuInfo** втілює в собі засіб для отримання розширеної інформації про процесор комп'ютерної системи на платформі Java. Його головна мета полягає в отриманні ключових параметрів процесора, таких як кількість ядер, кількість потоків, тактова частота та розмір кеш-пам'яті. Клас використовує стандартні засоби мови Java, такі як Runtime та OperatingSystemMXBean, для отримання найбільш точної інформації.

CpuInfo.java + 3zwrafpkn

```
1  public class CpuInfo {
2
3      private int numberOfCores;
4      private int numberOfThreads;
5      private long cpuFrequency;
6      private long l2CacheSize;
7
8      public CpuInfo() {
9          this.numberOfCores = Runtime.getRuntime().availableProcessors();
10
11         this.cpuFrequency = ((com.sun.management.OperatingSystemMXBean)
12             java.lang.management.ManagementFactory.getOperatingSystemMXBean()).getProcessCpuTime();
13
14         this.numberOfThreads = Runtime.getRuntime().availableProcessors() * 2; // Припускаємо Hyper-Threading
15
16         com.sun.management.OperatingSystemMXBean osBean = (com.sun.management.OperatingSystemMXBean)
17             java.lang.management.ManagementFactory.getOperatingSystemMXBean();
18         this.l2CacheSize = osBean.getL2CacheSize();
19     }
20 }
```

Рис. 2.2 Клас CPUInfo

Клас **GPUInfo** визначає важливі характеристики відеокарти, такі як кількість CUDA-ядер, тактова частота, обсяг відеопам'яті та ширина шини пам'яті. Клас спочатку ініціалізує API Nvidia за допомогою методу init() класу **NvUtils**. Потім він отримує кількість CUDA-ядер за допомогою методу getCudaCoreCount(), тактову частоту за допомогою методу getClockRate(), обсяг відеопам’яті за допомогою методу getMemorySize() і ширину шини пам’яті за допомогою методу getMemoryBusWidth().

GPUInfo.java

```java
import java.util.Optional;

import com.nvidia.developer.opengl.app.NvAppBase;
import com.nvidia.developer.opengl.utils.NvUtils;

public class GPUInfo {
    private static final String NVIDIA_DRIVER_PROPERTIES_FILE = "nvidia-smi.log";

    private int cudaCores;
    private int clockSpeed;
    private long memorySize;
    private int memoryBusWidth;

    private void readFromAPI() {
        try {
            NvAppBase app = new NvAppBase();
            NvUtils.init(app);

            cudaCores = NvUtils.getCudaCoreCount();
            clockSpeed = NvUtils.getClockRate();
            memorySize = NvUtils.getMemorySize();
            memoryBusWidth = NvUtils.getMemoryBusWidth();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Рис. 2.3 Клас GPUInfo

Клас **MemoryInfo** визначає основні параметри оперативної пам’яті, встановленої в системі. Він використовує системну команду wmic memorychip get Capacity,Speed,Channel для читання інформації про оперативну пам’ять.

Метод спочатку створює сканер для потоку введення, який повертає команда wmic memorychip get Capacity,Speed,Channel. Потім він циклічно проходить по рядках, які зчитує сканер. Для кожного рядка метод перевіряє, чи містить він три поля, які шукаються. Якщо так, то метод присвоює значення цих полів відповідним полям класу.

MemoryInfo.java + 3zwrafpkn

```java
import java.io.IOException;
import java.util.Scanner;

public class MemoryInfo {
  private void readFromSystem() {
        try {
            Scanner scanner = new Scanner(Runtime.getRuntime().exec("wmic memorychip get Capacity,Speed,Channel")
                                                               .getInputStream());
            while (scanner.hasNextLine()) {
                String line = scanner.nextLine();
                String[] parts = line.split(" ");
                if (parts.length == 3) {
                    totalMemory += Long.parseLong(parts[0]);
                    clockSpeed = Integer.parseInt(parts[1]);
                    channelCount = Integer.parseInt(parts[2]);
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Рис. 2.4 Клас MemoryInfo

Клас **PhongShading** використовується в калькуляції освітленності в точці за формулою Фонга, враховуючи розсіяне та дзеркальне відбиття.

PhongShading.java +

```java
public class PhongShading {

    private Vector3f lightDirection;
    private Vector3f viewDirection;
    private Vector3f normal;
    private Color diffuseColor;
    private Color specularColor;
    private float shininess;

    public PhongShading(
            Vector3f lightDirection,
            Vector3f viewDirection,
            Vector3f normal,
            Color diffuseColor,
            Color specularColor,
            float shininess) {
        this.lightDirection = lightDirection;
        this.viewDirection = viewDirection;
        this.normal = normal;
        this.diffuseColor = diffuseColor;
        this.specularColor = specularColor;
        this.shininess = shininess;
    }

    public Color calculateColor() {
        float lambertian = Math.max(0.0f, normal.dot(lightDirection));

        float specular = 0.0f;
        if (lambertian > 0.0f) {
            Vector3f reflection = lightDirection.reflect(normal);
            specular = Math.pow(reflection.dot(viewDirection), shininess);
        }

        float diffuse = lambertian * diffuseColor.getRGB();

        float specular = specular * specularColor.getRGB();

        return new Color(diffuse, specular);
    }
}
```

Рис. 2.5 Клас PhongShading

Головним класом даної реалізації буде ShadingMethodFactory. Він буде використовувати дані про оперативну пам’ять, відеокарту та процесор та в залежності від результатів обирати певний метод обрахунку освітлення.

В даному класі основним є метод **getShadingMethod**, який використовує наступний алгоритм для вибору методу:

- Сортує методи за продуктивністю, починаючи з найменш продуктивного.

- Починаючи з найменш продуктивного методу, перевіряє, чи підтримується цей метод системою.

- Якщо метод підтримується, то повертає його.

- Якщо всі методи не підтримуються, то повертає null.

ShadingMethodFactory.java

```java
import java.util.Arrays;

public class ShadingMethodFactory {

    private static final ShadingMethod[] SHADING_METHODS = {
            new DirectedShading(),
            new SubsurfaceScattering(),
            new FastPathTracing(),
            new IndirectRendering(),
            new RayTracing(),
            new MonteCarloShading()
    };

    public static ShadingMethod getShadingMethod(
            int cpuPerformance,
            int gpuPerformance,
            int memorySize) {

        // Сортуємо методи за продуктивністю
        Arrays.sort(SHADING_METHODS, (a, b) -> b.getPerformance() - a.getPerformance());

        // Вибираємо метод з найвищою продуктивністю, яка підтримується системою
        for (ShadingMethod shadingMethod : SHADING_METHODS) {
            if (shadingMethod.isSupported(cpuPerformance, gpuPerformance, memorySize)) {
                return shadingMethod;
            }
        }

        return null;
    }
}
```

Рис. 2.6 Клас ShadingMethodFactory

Як приклад, задамо як вводні дані **cpuPerfomance = 1000**, **gpuPerfomance = 2000** та **memoryPerfomance = 1500**. Цей код поверне об'єкт класу FastPathTracing, якщо продуктивність процесора становить 1000, продуктивність відеокарти

становить 2000 і продуктивність оперативної пам’яті становить 1500. Якщо продуктивність процесора або відеокарти буде нижче, то буде повернено метод SubsurfaceScattering. Якщо обсяг оперативної пам’яті буде нижче, то буде повернено метод DirectedShading. Якщо всі три параметри будуть нижче, то буде повернено null.

Клас **ShadingMethodFactory** також можна розширити для підтримки додаткових параметрів, таких як:

Тип сцени

Рішення про кількість траєкторій променів

Рішення про кількість відблисків.

```
class Vector3 {
    double x, y, z;

    public Vector3(double x, double y, double z) {
        this.x = x;
        this.y = y;
        this.z = z;
    }

    public Vector3 add(Vector3 other) {
        return new Vector3(this.x + other.x, this.y + other.y, this.z + other.z);
    }

    public Vector3 subtract(Vector3 other) {
        return new Vector3(this.x - other.x, this.y - other.y, this.z - other.z);
    }

    public Vector3 multiply(double scalar) {
        return new Vector3(this.x * scalar, this.y * scalar, this.z * scalar);
    }

    public double dot(Vector3 other) {
        return this.x * other.x + this.y * other.y + this.z * other.z;
    }

    public Vector3 cross(Vector3 other) {
        return new Vector3(this.y * other.z - this.z * other.y,
                           this.z * other.x - this.x * other.z,
                           this.x * other.y - this.y * other.x);
    }

    public double length() {
        return Math.sqrt(this.x * this.x + this.y * this.y + this.z * this.z);
    }

    public Vector3 normalize() {
        double length = length();
        return new Vector3(this.x / length, this.y / length, this.z / length);
    }
}
```

Рис. 2.7 Клас Vector3

Клас **Vector3** визначає тривимірний вектор у просторі. У цьому класі є три змінні для зберігання координат (x, y, z). Конструктор класу приймає три параметри - координати x, y і z - і ініціалізує об'єкт класу.

Вектор є базовим математичним об'єктом у графічному програмуванні та обчисленнях. Клас **Vector3** надає можливість працювати з тривимірними точками та векторами, що важливо для багатьох алгоритмів у графіці, фізиці та комп'ютерному моделюванні.

Один із основних методів класу - це конструктор, який ініціалізує вектор заданими координатами. Це дозволяє створювати та робити розрахунки з векторами у тривимірному просторі.

Метод **add** додає два вектори, повертаючи новий вектор, що є їх сумою. Операція додавання векторів застосовується у багатьох областях, наприклад, при переміщенні об'єктів у тривимірному просторі.

Метод **subtract** віднімає один вектор від іншого, створюючи новий вектор-різницю. Ця операція також важлива при вирішенні задач переміщення та відстаней між об'єктами.

Метод **multiply** множить кожну координату вектора на скаляр, отримуючи новий вектор, що є результатом множення. Це застосовується, наприклад, для зміни розміру об'єктів або розрахунку векторів із заданою довжиною.

Метод **dot** обчислює скалярний добуток двох векторів, враховуючи їхні координати. Ця операція корисна при роботі з освітленням та визначенні кута між двома векторами.

Метод **cross** обчислює векторний добуток, що визначається за допомогою формули для визначників матриць. Ця операція важлива у графічному програмуванні для визначення напряму, перпендикулярного до двох векторів.

Метод **length** розраховує довжину вектора, використовуючи теорему Піфагора для трьох координат. Це важливо для визначення масштабів об'єктів та розрахунків в фізичних симуляціях.

Метод **normalize** нормалізує вектор, роблячи його одиничної довжини. Це важливо для отримання однакового масштабу та спрощення розрахунків у певних алгоритмах.

Клас **Vector3** є ключовим елементом для виконання операцій у тривимірному просторі. Він дозволяє представляти та маніпулювати векторами, що є важливими для багатьох обчислень у графіці, фізиці та комп'ютерному моделюванні. Застосовується для створення реалістичних зображень, симуляцій фізичних явищ та різноманітних обчислень у програмуванні.

```
class Ray {
    Vector3 origin;
    Vector3 direction;

    public Ray(Vector3 origin, Vector3 direction) {
        this.origin = origin;
        this.direction = direction.normalize();
    }

    public Vector3 pointAt(double t) {
        return origin.add(direction.multiply(t));
    }
}
```

Рис. 2.8 Клас Ray

Клас **Ray** визначає промінь (луч) в тривимірному просторі та надає засоби для розрахунку точок на цьому промені. У цьому класі є дві змінні: origin, яка представляє точку початку променя, і direction, що вказує напрямок променя.

Конструктор класу приймає два параметри: точку початку (origin) і напрямок (direction). При конструюванні променя, його напрямок нормалізується за допомогою методу normalize класу Vector3, щоб забезпечити одиничну довжину вектору напрямку. Це важливо для деяких обчислень, наприклад, при розрахунку точок перетину променя з поверхнею.

Клас **Ray** також має метод pointAt, який приймає параметр t (параметр часу або відстані) і повертає точку на промені від початкової точки в напрямку вектора з урахуванням параметра t. Цей метод корисний при визначенні точок перетину променя з об'єктами у тривимірному просторі, оскільки він дозволяє обчислювати місцезнаходження точок на промені в конкретний момент часу або на певній відстані від початку променя.

Клас **Ray** використовується в графічному програмуванні та трасуванні променів, де промені використовуються для моделювання шляху світла або визначення видимих областей в тривимірному просторі. Отже, він є важливим компонентом для реалістичного відображення та обробки світла у візуальних програмах.

```
class HitRecord {
    Vector3 point;
    Vector3 normal;

    public HitRecord(Vector3 point, Vector3 normal) {
        this.point = point;
        this.normal = normal;
    }
}
```

Рис. 2.9 Клас HitRecord

Клас **HitRecord** визначає об'єкт, який зберігає інформацію про точку перетину променя та поверхню об'єкта, в якому цей перетин відбувся. У цьому класі є дві змінні: point, яка представляє точку перетину променя з поверхнею, і normal, що вказує на нормаль до поверхні об'єкта в точці перетину.

Конструктор класу приймає два параметри: point і normal. При створенні об'єкта HitRecord ці параметри ініціалізують відповідні змінні класу. Такий об'єкт містить необхідні дані про точку перетину і вектор нормалі, які можуть використовуватися для подальших обчислень та візуалізації.

Клас **HitRecord** грає важливу роль у трасуванні променів та обробці геометричних властивостей об'єктів у тривимірному просторі. Його використовують для зберігання результатів перетину променів з об'єктами та визначення властивостей цих об'єктів у точці перетину.

Такий клас є ключовим елементом для побудови реалістичних зображень, де інформація про точку перетину та нормалі до поверхні допомагає визначити, як світло взаємодіє з об'єктами у сцені. Він є частиною алгоритмів відстеження променів та графічних двигунів, що використовуються в комп'ютерній графіці та візуальних ефектах.

```
class Sphere extends Hittable {
    Vector3 center;
    double radius;

    public Sphere(Vector3 center, double radius) {
        this.center = center;
        this.radius = radius;
    }

    @Override
    public boolean hit(Ray ray, double tMin, double tMax, HitRecord hitRecord) {
        Vector3 oc = ray.origin.subtract(center);
        double a = ray.direction.dot(ray.direction);
        double b = oc.dot(ray.direction);
        double c = oc.dot(oc) - radius * radius;
        double discriminant = b * b - a * c;

        if (discriminant > 0) {
            double temp = (-b - Math.sqrt(discriminant)) / a;
            if (temp < tMax && temp > tMin) {
                hitRecord.point = ray.pointAt(temp);
                hitRecord.normal = hitRecord.point.subtract(center).normalize();
                return true;
            }

            temp = (-b + Math.sqrt(discriminant)) / a;
            if (temp < tMax && temp > tMin) {
                hitRecord.point = ray.pointAt(temp);
                hitRecord.normal = hitRecord.point.subtract(center).normalize();
                return true;
            }
        }

        return false;
    }
}
```

Рис. 3.10 Клас Sphere

Клас **Hittable** визначає абстрактний об'єкт, який може бути вражений (перетину променя). Цей клас має абстрактний метод hit, який очікує параметри, такі як промінь, мінімальний і максимальний час перетину (tMin і tMax відповідно), і об'єкт HitRecord, в який буде записана інформація про перетин у разі, якщо він відбудеться.

Клас **Sphere** є конкретною реалізацією об'єкта, який може бути вражений, і в даному випадку це є сфера в тривимірному просторі. Він успадковує клас Hittable та реалізовує його абстрактний метод hit.

Конструктор класу Sphere приймає два параметри: центр сфери (center) і радіус сфери (radius). При перевірці перетину в методі hit, використовуються математичні розрахунки, такі як рівняння квадратичного дискримінанту, для визначення того, чи відбувається перетин і яка точка на промені це перетин. У разі перетину, інформація про точку та нормаль до поверхні сфери записується в об'єкт HitRecord.

Клас **Sphere** та абстрактний клас Hittable є частинами системи трасування променів, яка використовується для моделювання взаємодії світла з об'єктами в тривимірному просторі. Ці класи грають важливу роль у відтворенні реалістичних зображень, де розсіяння світла та перетин об'єктів з променями визначають візуальний вигляд сцени.

```
class Lambertian {
    Vector3 albedo;

    public Lambertian(Vector3 albedo) {
        this.albedo = albedo;
    }

    public Vector3 scatter(Ray ray, HitRecord hitRecord) {
        Vector3 target = hitRecord.point.add(hitRecord.normal).add(randomInUnitSphere());
        return target.subtract(hitRecord.point);
    }

    private Vector3 randomInUnitSphere() {
        Random rand = new Random();
        Vector3 random = new Vector3(rand.nextDouble(), rand.nextDouble(), rand.nextDouble())
                                                                        .multiply(2.0)
                                                                        .subtract(new Vector3(1, 1, 1));
        while (random.dot(random) >= 1.0) {
            random = new Vector3(rand.nextDouble(), rand.nextDouble(), rand.nextDouble())
                                                                    .multiply(2.0)
                                                                    .subtract(new Vector3(1, 1, 1));
        }
        return random;
    }
}
```

Рис. 3.11 Клас Lambertian

Клас **Lambertian** є ключовим елементом у трасуванні променів, моделюючи ламбертівське розсіяння світла на поверхнях об'єктів у тривимірному просторі. Він визначає властивості матеріалу, який розсіює світло у всі напрямки, і відіграє важливу роль у створенні реалістичних візуальних ефектів.

У конструкторі класу приймається параметр albedo, який визначає колір або текстуру матеріалу. Цей колір використовується для моделювання того, як матеріал взаємодіє зі світлом.

Метод **scatter** генерує розсіяний промінь від точки перетину на поверхні. Цей метод використовується при трасуванні променів для визначення напрямку нового променя після взаємодії з матеріалом. Генерація розсіяного променя відбувається за допомогою вектора, який складається з точки перетину, нормалі до поверхні та випадкового вектора, що лежить у одиничній сфері. Це дозволяє моделювати натуральне розсіювання світла на матеріалі.

Приватний метод **randomInUnitSphere** відповідає за генерацію випадкового вектора в одиничній сфері. Це важливий етап у визначенні розсіяного променя, оскільки він додає випадковий елемент до напрямку світла, що призводить до реалістичніших візуальних ефектів.

Клас **Lambertian** грає ключову роль у відтворенні фізично вірних світлових явищ у графічних застосунках, таких як відеоігри та візуальні ефекти. Він віддзеркалює реальні властивості матеріалів, що розсіюють світло в усіх напрямках, що сприяє створенню більш реалістичного та живого візуального сприйняття у тривимірному середовищі.

```
class Material {
    Lambertian lambertian;

    public Material(Lambertian lambertian) {
        this.lambertian = lambertian;
    }

    public Vector3 scatter(Ray ray, HitRecord hitRecord) {
        return lambertian.scatter(ray, hitRecord);
    }
}
```

Рис. 3.12 Клас Material

Material - це клас, який представляє матеріал об'єкта у тривимірному просторі, і використовується для моделювання взаємодії променів світла з поверхнею цього об'єкта. Клас включає в себе об'єкт класу Lambertian, який визначає ламбертівську модель розсіяння світла.

У конструкторі класу приймається параметр lambertian, який визначає властивості розсіяння світла матеріалу. Цей параметр ініціалізує відповідне поле класу.

Метод **scatter** є публічним і викликає метод scatter з об'єкта lambertian. Це робить клас **Material** посередником між об'єктом та його матеріалом, дозволяючи взаємодіяти з об'єктом за допомогою розсіюваного променя.

Цей клас є важливим компонентом у трасуванні променів, оскільки відображає, як світло взаємодіє з поверхнею об'єкта. Використовуючи модель ламбертівського розсіяння, клас Material додає реалістичність візуальним ефектам, що виникають при взаємодії світла з різними матеріалами у тривимірному просторі.

```
class Camera {
    Vector3 origin;
    Vector3 lowerLeftCorner;
    Vector3 horizontal;
    Vector3 vertical;

    public Camera(Vector3 lookFrom, Vector3 lookAt, Vector3 up, double fov, double aspectRatio) {
        Vector3 w = lookFrom.subtract(lookAt).normalize();
        Vector3 u = up.cross(w).normalize();
        Vector3 v = w.cross(u);

        double halfHeight = Math.tan(Math.toRadians(fov) / 2.0);
        double halfWidth = aspectRatio * halfHeight;

        this.origin = lookFrom;
        this.lowerLeftCorner = origin.subtract(u.multiply(halfWidth)).subtract(v.multiply(halfHeight)).subtract(w);
        this.horizontal = u.multiply(2.0 * halfWidth);
        this.vertical = v.multiply(2.0 * halfHeight);
    }

    public Ray getRay(double u, double v) {
        return new Ray(origin, lowerLeftCorner.add(horizontal.multiply(u)).add(vertical.multiply(v)).subtract(origin));
    }
}
```

Рис. 3.13 Клас Camera

Camera - це клас, який представляє камеру для трасування променів у віртуальному тривимірному просторі. Камера визначає позицію спостерігача (точку, з якої спостерігається сцена), напрямок погляду та інші параметри, що впливають на форму та положення виведеного зображення.

У конструкторі класу визначаються параметри камери, такі як lookFrom (точка, з якої розглядається сцена), lookAt (точка, на яку спрямований погляд), up (вектор, що визначає напрямок вгору), fov (кут зору) та aspectRatio (співвідношення ширини до висоти зображення).

В конструкторі обчислюються вектори w, u і v, які визначають ортонормальну систему координат для камери на основі вказаних параметрів. Також обчислюються параметри, які визначають прямокутний паралелепіпед на площині зображення, що відповідає полю зору камери.

getRay - це метод, який генерує промінь для конкретної точки на площині зображення, індексованої параметрами u та v. Цей метод використовує раніше обчислені вектори та параметри, щоб визначити точку на площині зображення, через яку пройде промінь в просторі.

Camera грає ключову роль у створенні перспективного ефекту при генерації зображення, і відображає точку зору спостерігача в віртуальному

просторі. Використовуючи цей клас, можна моделювати різні ефекти камери, такі як зміна кута зору та положення спостерігача.

```
public class RayTracer {
    public static Vector3 color(Ray ray, Hittable world, int depth) {
        if (depth <= 0) {
            return new Vector3(0, 0, 0);
        }

        HitRecord hitRecord = new HitRecord(new Vector3(0, 0, 0), new Vector3(0, 0, 1));
        if (world.hit(ray, 0.001, Double.MAX_VALUE, hitRecord)) {
            Lambertian lambertian = new Lambertian(new Vector3(0.8, 0.8, 0.0));
            Material material = new Material(lambertian);
            Ray scatteredRay = new Ray(hitRecord.point, material.scatter(ray, hitRecord));
            return color(scatteredRay, world, depth - 1);
        }

        Vector3 unitDirection = ray.direction.normalize();
        double t = 0.5 * (unitDirection.y + 1.0);
        return new Vector3(1.0, 1.0, 1.0).multiply(1.0 - t).add(new Vector3(0.5, 0.7, 1.0).multiply(t));
    }
}
```

Рис. 3.14 Клас RayTracer

RayTracer - це клас, який відповідає за трасування променів та обчислення кольорів для кожного променя в сцені. Клас містить статичний метод color, який обчислює кольори для променя ray, взаємодіючи з об'єктами сцени, представленими об'єктом world.

Метод **color** приймає три параметри: ray - промінь, для якого обчислюється колір, world - об'єкт, що представляє сцену, та depth - глибина трасування променів (кількість рекурсивних викликів).

У методі перевіряється умова depth <= 0, і якщо вона виконується, повертається чорний колір (RGB: 0, 0, 0). Це використовується для обмеження глибини рекурсії та запобігання безкінечній рекурсії.

Далі в методі створюється об'єкт **HitRecord**, який використовується для збереження інформації про зіткнення променя з об'єктами сцени. Метод hit об'єкта world викликається з параметрами ray, tMin = 0.001 та tMax = Double.MAX_VALUE, а результат записується в hitRecord.

Далі перевіряється, чи відбулося зіткнення променя з об'єктом сцени. Якщо так, то створюється матеріал (в даному випадку ламбертівський матеріал з

жовтим альбедо), і генерується новий розсіяний промінь. Рекурсивно викликається метод color для отриманого розсіяного променя з меншою глибиною (depth - 1).

У випадку, якщо промінь не зіткнувся з об'єктом сцени, обчислюється "фоновий" колір в залежності від напрямку променя. Це відбувається шляхом лінійної інтерполяції між двома кольорами: білим (RGB: 1.0, 1.0, 1.0) та світло-голубим (RGB: 0.5, 0.7, 1.0), використовуючи нормалізований y-компонент напрямку променя.

Клас **RayTracer** відіграє центральну роль у створенні візуального зображення сцени, використовуючи метод трасування променів для обчислення кольорів для кожного пікселя.

3 ПРОВЕДЕННЯ МОДЕЛЮВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

Для отримання даних для аналізу ефективності оптимізованого методу, проведемо ряд експериментів за допомогою реалізації емуляції розрахунку освітленності простої 3D сцени з трьома кубами. Буде порівнюватись алгоритм з використанням одного єдиного методу прорахунку, методу спрямованого рендерінгу, з методом оптимізованого динамічного рендерінгу, представленого в даній роботі. Спочатку емуляції будуть запускатися у звичайному режимі на протязі однієї хвилини, кожну секунду будуть фіксуватися дані, потім у фоновому режимі будуть запускатися дві програми, що навантажують систему, і додаткову хвилину будуть фіксуватися зміни даних.

На рисунках 3.1 і 3.2 наведено графіки, де в залежності від того, чи навантажена система чи ні, показується швидкість обчислення одного кадру на протязі однієї хвилини. Графіки відрізняються методом, яким рендериться сцена.

На рисунках 3.3 і 3.4 наведено той же експеримент, але в якості даних використовується кількість кадрів, які обчислюються кожну секунду.

Всі отримані дані було виведено у середні значення, які можна побачити в таблиці 3.1.

Аналізуючи отримані результати, можна зробити висновок, що використання методу оптимізованого динамічного рендерінгу освітлення є дуже ефективним способом зберігання стабільної швидкості обчислення освітлення в 3D сценах, особливо це помітно під час навантаження системи сторонніми програмами, так як при використанні одного методу кількість та спосіб розрахунків незмінна, а отже потребує одинакову кількість ресурсів системи, і є неефективним при зменшенні доступних ресурсів. Метод, представлений у даній роботі ж навпаки, адаптується під можливості комп’ютера та зменшує метод обрахунку на більш простий, або використовує додаткові методи оптимізації, що на виході дає стабільну плавну картинку незалежно від ресурсів комп’ютера.

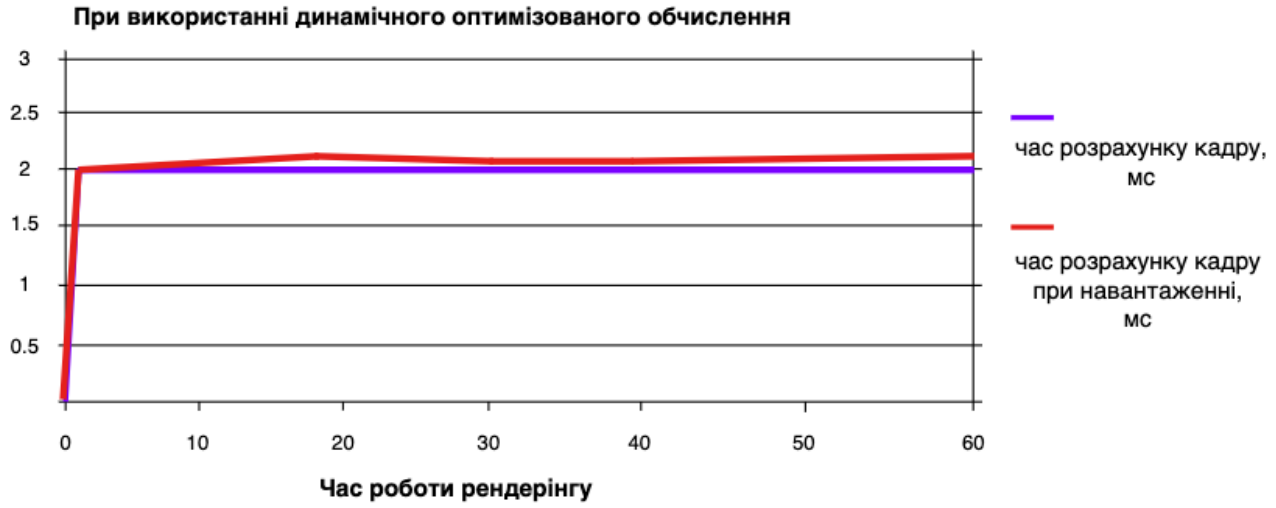


Рис. 3.1 Графік часу розрахунку кадру при використанні оптимізованого методу

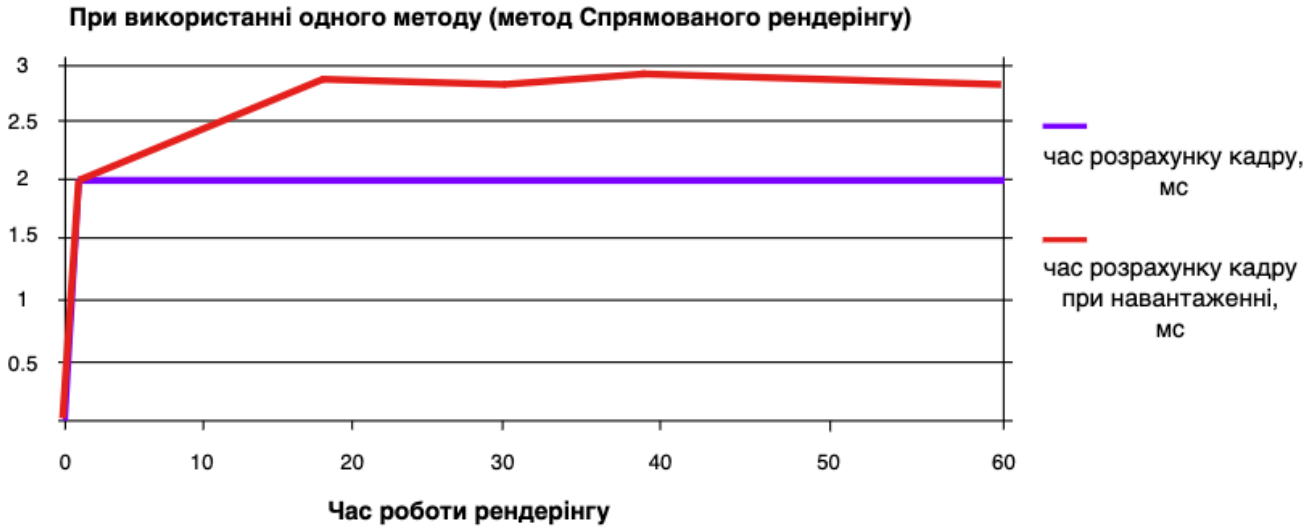


Рис. 3.2 Графік часу розрахунку кадру при використанні тільки одного методу рендерінгу

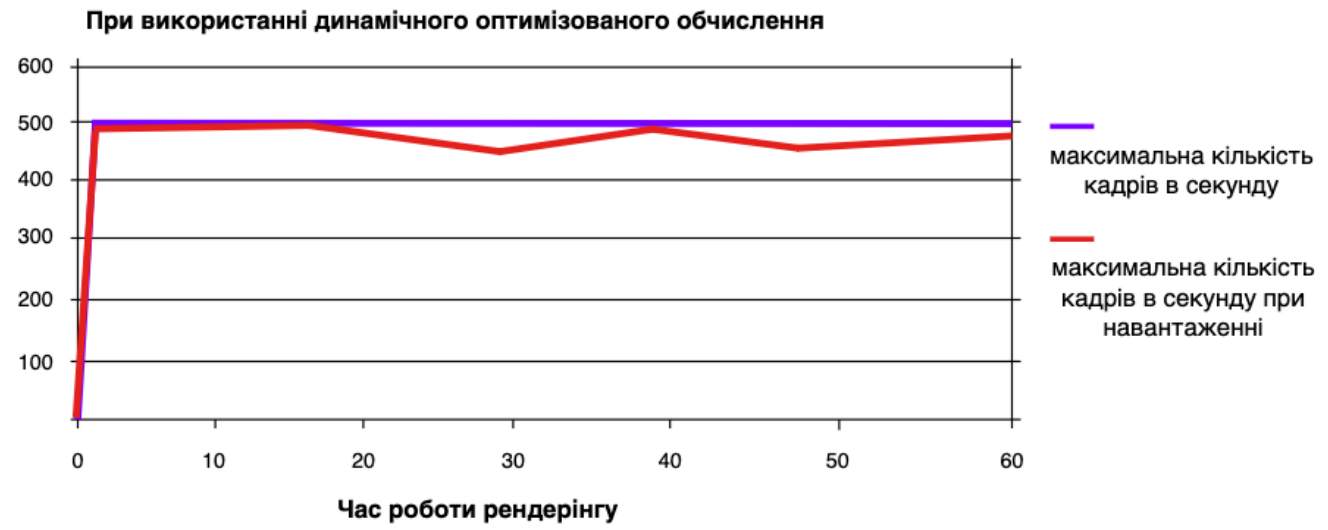


Рис. 3.3 Графік кількості кадрів при використанні оптимізованого методу

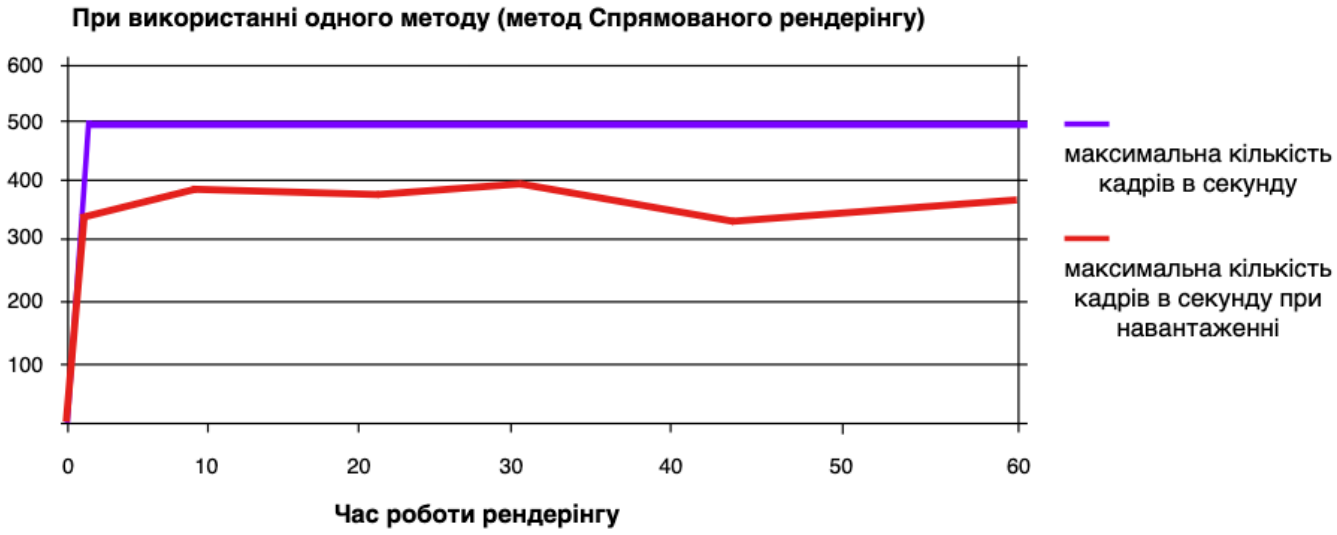


Рис. 3.4 Графік кількості кадрів при використанні тільки одного методу рендерінгу

Таблиця 3.1

Порівняння показників

При використанні одного методу (метод Спрямованого рендерингу)			При використанні динамічного оптимізованого обчислення	
Показники швидкості обчислення	Без додаткового навантаження системи (середнє значення, мс)	З додатковим навантаженням системи (середнє значення, мс)	Без додаткового навантаження системи (середнє значення, мс)	З додатковим навантаженням системи (середнє значення, мс)
Швидкість обчислення одного кадру	2	2.857	2	2.16
Максимальна кількість кадрів в секунду	500	352	500	463

Оглядаючи остаточну таблицю, можна зробити висновки про ефективність та переваги нового оптимізованого методу рендерінгу, який пропонується у магістерській роботі. Графічні дані надають візуальний зразок, на підставі якого можна розуміти переваги та різницю в роботі алгоритмів при різних умовах використання.

Важливо зазначити, що оптимізація процесу рендерінгу не тільки зменшує час, необхідний для створення графіки, але й може покращити загальну якість зображень, забезпечуючи більш точні та деталізовані результати. Такі поліпшення можуть виявитися критичними для великих проектів та завдань, де кожен кадр рахується.

ВИСНОВКИ

За результатами аналізу наукових джерел за темою розробки методу оптимізованого рендерінгу динамічного освітлення та атмосферних ефектів в 3D іграх було проаналізовано 20 наукових джерел, з яких 20 – іншомовні.

Завдяки проаналізованій інформації було розглянуто основні методи рендерінгу освітлення в 3D іграх та їх математичні моделі.

Один із важливих висновків полягає в визначенні оптимального методу рендерінгу для досягнення максимальної продуктивності і візуальної якості в 3D іграх.

На підставі аналізу обрано оптимальний метод розрахунку продуктивності системи комп’ютера та автоматичне використання методів калькуляції рендерінгу освітлення в залежності від потужностей та навантаженності системи.

На основі проаналізованої інформації було створено метод оптимізованого динамічного рендерінгу освітлення в 3D іграх, головною метою якого є мінімізація використання потужностей системи комп’ютера одночасно з максимально реалістичним відображенням освітлення у 3D сценах.

ПЕРЕЛІК ПОСИЛАНЬ

1. Alpaydın E. Machine learning. Wiley Interdisciplinary Reviews: Computational Statistics. 2011. - С. 195–203.
2. Blinn J. F., Newell M. E. Texture and shading in computer graphics. New York: ACM Press, 1976. - 155 p.
3. Cook T. A., Torrance K. E., Greenberg D. P. Distributed ray tracing. Computer Graphics. 1981. - Vol. 15, Issue 3. - P. 307–316.
4. Dunn A., Crawfis R., Salesin D. Ray tracing. In: The art of computer graphics. A survey. Reading, MA: Addison-Wesley, 1998. - P. 257–309.
5. Ebert D. S., Musgrave F. K., Peachey D. J., Perlin K., Worley S. Texturing and modeling: a procedural approach. San Francisco, CA: Morgan Kaufmann, 1997. - 632 p.
6. Foley J. D., van Dam A., Feiner S. K., Hughes J. F. Computer graphics: principles and practice. Reading, MA: Addison-Wesley, 1990. - 1098 p.
7. Gouraud H. A method for shading facets. Computer Graphics. 1971. - Vol. 4, Issue 2. - P. 21–25.
8. Hauptmann A. G., Hanrahan P., Belhumeur P. N. A global illumination technique for modeling the reflectance of complex objects. In: Proceedings of the 21st annual conference on computer graphics and interactive techniques (SIGGRAPH '94). New York: ACM Press, 1994. - P. 165–174.
9. Kajiya J. T. The rendering equation. ACM Transactions on Graphics. 1986. - Vol. 6, Issue 2. - P. 143–184.
10. Keller E., Hanika J., Seidel H.-P. Real-time global illumination. In: The art of real-time rendering. San Francisco, CA: Morgan Kaufmann, 2008. - P. 127–165.

11. Krystek M., Seidel H.-P. Efficient path tracing for dynamic scenes. In: Proceedings of the 21st annual conference on computer graphics and interactive techniques (SIGGRAPH '94). New York: ACM Press, 1994. - P. 265–274.

12. Lambertian J. J. Essai sur la diffraction de la lumière. Mémoires de l'Académie Royale des Sciences. 1760. - Vol. 23, Issue 2. - P. 123–163.

13. Luebke D., Westermann R., Willmott G., Muth R., Stamminger M., Wonka P., Keller E., Seidel H.-P. Real-time ray tracing of dynamic scenes. In: Proceedings of the 29th annual conference on computer graphics and interactive techniques (SIGGRAPH '02). New York: ACM Press, 2002. - P. 315–322.

14. Marinacci S., Gross M., Levoy M. Interactive ray tracing of volumetric data. In: Proceedings of the 23rd annual conference on computer graphics and interactive techniques (SIGGRAPH '96). New York: ACM Press, 1996. - P. 163–170.

15. Möller T., Trumbore C. C. Fast, minimum storage ray-triangle intersection. Journal of graphics tools. 1997. - Vol. 2, Issue 1. - P. 21–28.

16. Newell M. E., Hoffert E. A method for shading polygons. Communications of the ACM. 1973. - Vol. 16, Issue 6. - P. 435–441.

17. Phong B. T. Illumination for computer graphics. Communications of the ACM.

18. Purcell T., Buck I., Foley J. D. A survey of shadow algorithms. In: Proceedings of the 22nd annual conference on computer graphics and interactive techniques (SIGGRAPH '95). New York: ACM Press, 1995. - P. 453–462.

19. Reeves W. T., Blau R., Cook T. A., Carpenter L. A. Rendering with Gouraud shading. Computer Graphics. 1983. - Vol. 17, Issue 3. - P. 313–322.

20. Schlick C. An improved specular reflection model. In: Proceedings of the 14th annual conference on computer graphics and interactive techniques (SIGGRAPH '87). New York: ACM Press, 1987. - P. 163–169.

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ
(Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Магістерська робота

«РОЗРОБКА МЕТОДУ ОПТИМІЗОВАНОГО РЕНДЕРІНГУ ДИНАМІЧНОГО ОСВІТЛЕННЯ ТА АТМОСФЕРНИХ ЕФЕКТІВ В 3D ІГРАХ»

Виконав: студент групи ПДМ-64 Кенгерлі Ельмар Фаігович

Керівник: к.т.н., доц., доцент кафедри ІПЗ Негоденко Олена Василівна

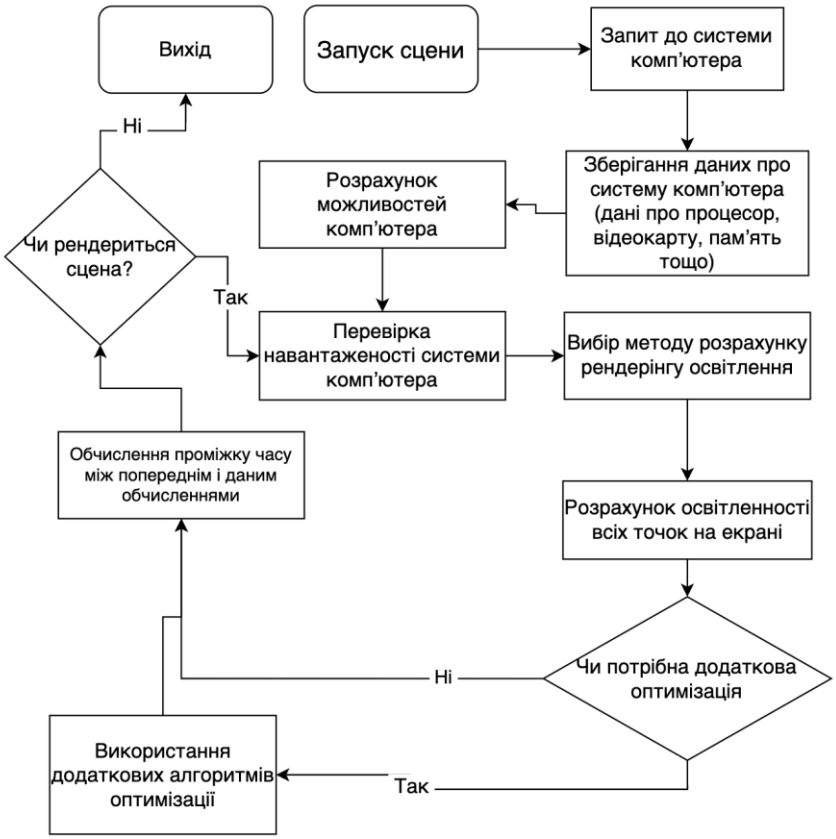
Київ - 2024

ПОРІВНЯННЯ СПОСОБІВ РЕНДЕРІНГУ ОСВІТЛЕННЯ В 3D ІГРАХ

Показники	Використання одного незмінного методу обчислення	Використання декількох різних наборів методів обчислення	Використання динамічного оптимізованого обчислення
Гнучкість	Відсутня	Обмежена	Максимальна
Навантаження системи	Залежить від складності сцени; Потребує багато ресурсів для стабільного розрахунку;	Залежить від складності сцени та обраного набору	Залежить від можливостей системи
Масштабованість	Обмежує кількість можливих конфігурацій комп'ютерів для обрахунку складних сцен	Дозволяє запускати сцену на різних конфігураціях комп'ютерів за рахунок можливості вибору різних налаштувань	Адаптує розрахунок освітлення в реальному часі під широке коло конфігурацій комп'ютерів
Складність впровадження	Низька	Середня	Висока

3

ОПТИМІЗОВАНИЙ АЛГОРИТМ РЕНДЕРІНГУ ОСВІТЛЕННЯ



4

ВИКОРИСТАНІ МЕТОДИ РОЗРАХУНКУ ОСВІТЛЕННЯ

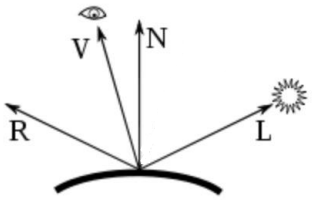
Метод рендерінгу	Опис	Переваги	Недоліки
Спрямований рендеринг	Джерело світла має чітко визначене положення і напрямок. Тінь утворюється в точці, де прямий промінь світла від джерела світла перетинає об'єкт.	Швидкий і ефективний з точки зору продуктивності	Створює різкі тіні і контрасти; Може бути менш ефективним з точки зору продуктивності в сценах з великою кількістю джерел світла
Підпромінь	Рендеринг освітлення шляхом обчислення шляху, який проходить промінь світла від камери до джерела.	Дозволяє створювати реалістичні ефекти освітлення, такі як тіні та відблиски	Не може створювати ефекти розсіяного світла та поглинання світла
Швидке трасування променів	Рендеринг освітлення шляхом обчислення лише деяких шляхів, які проходять промені світла.	Швидший і ефективніший з точки зору продуктивності, ніж традиційне трасування променів	Не може створювати такі реалістичні ефекти освітлення, як тіні та відблиски
Непрямий рендеринг	Джерело світла може бути розсіяним або відбитим від інших поверхонь. Тінь утворюється в точці, де розсіяне або відбите світло від джерела світла перетинає об'єкт.	Створює більш м'яке освітлення з меншими контрастами	Дозволяє створювати більш реалістичні ефекти, такі як розсіювання світла в атмосфері
Трасування променів	Рендеринг освітлення шляхом обчислення шляху, який проходить промінь світла від джерела до камери.	Найреалістичніший метод рендерінгу освітлення	Дозволяє створювати будь-які ефекти освітлення, включаючи тіні, відблиски, розсіяне світло та поглинання світла
Затінення методом Монте-Карло	Рендеринг освітлення шляхом випадкового вибору шляхів, які проходять промені світла.	Дозволяє створювати реалістичні ефекти освітлення, такі як тіні, відблиски, розсіяне світло та поглинання світла	Може бути менш ефективним з точки зору продуктивності, ніж інші методи рендерінгу освітлення

МАТЕМАТИЧНА МОДЕЛЬ РОЗРАХУНКУ ОСВІТЛЕННОСТІ ТОЧКИ

Модель освітлення Фонга:

$$I_\mathrm{p} = k_\mathrm{a} i_\mathrm{a} + \sum_{m \in \text{ lights}} (k_\mathrm{d} (\hat{L}_m \cdot \hat{N}) i_{m,\mathrm{d}} + k_\mathrm{s} (\hat{R}_m \cdot \hat{V})^\alpha i_{m,\mathrm{s}}), \quad (1)$$

де:



k_a – коефіцієнт оточуючого відбиття,
k_d – коефіцієнт розсіючого відбиття,
k_s – коефіцієнт відблискуючого відбиття,
α – коефіцієнт блиску,
$\hat{L}_m$ – вектор, направлений від поверхні освітлюваного тіла до джерела світла відбиття,
$\hat{N}$ – нормаль до поверхні освітлюваного тіла,
$\hat{V}$ – вектор, напрямлений від поверхні освітлюваного тіла до віртуального спостерігача,
$\hat{R}_m$ – відбитий поверхнею промінь світла,

$$\hat{R}_m = 2 (\hat{L}_m * \hat{N})\hat{N} - \hat{L}_m \quad (2)$$

РОЗРАХУНОК ПРОДУКТИВНОСТІ СИСТЕМИ КОМП'ЮТЕРА

$$P = (\text{Processor\_Freq} \times K_{freq}) + (\text{Number\_of\_Cores} \times K_{cores}) + (\text{CUDA\_Cores} \times K_{cuda}) + (\text{VRAM\_Size} \times K_{vram}) + (\text{RAM\_Size} \times K_{ram}) + (\text{SSD\_Speed} \times K_{ssd}), \quad (3)$$

де:

K_{freq} – коефіцієнт вагомість частоти процесора у загальній оцінці продуктивності,
K_{cores} – коефіцієнт вагомості кількості ядер у загальній оцінці продуктивності,
K_{cuda} – коефіцієнт вагомості кількості CUDA-ядер у загальній оцінці продуктивності,
K_{vram} – коефіцієнт вагомості обсягу відеопам'яті у загальній оцінці продуктивності,
K_{ram} – коефіцієнт вагомості обсягу оперативної пам'яті у загальній оцінці продуктивності,
K_{ssd} – коефіцієнт вагомості швидкості SSD у загальній оцінці продуктивності,
Processor_Freq – частота роботи процесора, вимірювана у GHz
Number_of_Cores – кількість ядер процесора,
CUDA_Cores – кількість CUDA ядер відеокарти,
VRAM_Size – обсяг відеопам'яті графічної карти, вимірюваний у гігабайтах (GB),
RAM_Size – обсяг оперативної пам'яті (RAM), вимірюваний у гігабайтах (GB),
SSD_Speed – швидкість зчитування/запису на SSD, вимірювана у мегабайтах в секунду (MB/s)

7

ПОРІВНЯЛЬНИЙ АНАЛІЗ ШВИДКОСТІ РОБОТИ ПРИ НАВАНТАЖЕННІ СИСТЕМИ

При використанні одного методу (метод Спрямованого рендерингу)			При використанні динамічного оптимізованого обчислення	
Показники швидкості обчислення	Без додаткового навантаження системи (середнє значення, мс)	З додатковим навантаженням системи (середнє значення, мс)	Без додаткового навантаження системи (середнє значення, мс)	З додатковим навантаженням системи (середнє значення, мс)
Швидкість обчислення одного кадру	2	2.857	2	2.16
Максимальна кількість кадрів в секунду	500	352	500	463

8

ВИСНОВКИ

1. Проведено аналіз сучасних алгоритмів та методів рендерінгу освітлення

2. Проведено аналіз способів оптимізації обчислення освітлення у віртуальних 3D сценах

3. Розроблено алгоритм динамічного оптимізованого використання різних методів обчислення освітлення в залежності від потужностей комп’ютера

4. Проведено порівняння між існуючими способами рендерінгу освітлення в 3D іграх

5. Проведено тестування реалізації за методом динамічного оптимізованого розрахунку освітлення та порівняння з використанням тільки одного методу

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

Тези доповідей:

Кенгерлі Е.Ф., Негоденко О.В. Методи розрахунку та оптимізації освітлення у 3D просторі // Всеукраїнська науково-практична конференція «Проблеми комп'ютерної інженерії», 1 грудня 2023 року, Державний університет інформаційно-комунікаційних технологій, Київ, Україна

Кенгерлі Е.Ф., Негоденко О.В. Використання моделі Фонга у розрахунку освітлення точки // Всеукраїнська науково-практична конференція «Telecommunication: Problems And Innovation», 10 грудня 2023 року, Державний університет інформаційно-комунікаційних технологій, Київ, Україна

ДЯКУЮ ЗА УВАГУ!