МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики
(повне найменування назва факультету)

Кафедра інформаційних систем
(повна назва кафедри)

Магістерська робота

КОМП'ЮТЕРНЕ МОДЕЛЮВАННЯ СЦЕН З УРАХУВАННЯМ ЗАЛОМЛЕННЯ ПРОМЕНІВ СВІТЛА

Виконала: студентка групи ПМІМ-22с
спеціальності
122 – Комп'ютерні науки
(шифр і назва спеціальності)

(підпис) Ковальчук С. А.
(прізвище та ініціали)

Керівник (підпис) Стельмащук В. В.
(прізвище та ініціали)

Рецензент (підпис) Борачок І.В.
(прізвище та ініціали)

Львів – 2022

ДЕКАН

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет Прикладної математики та інформатики

Кафедра

Інформаційних систем

Спеціальність

122 Комп'ютерні науки

(шифр і назва)

«ЗАТВЕРДЖУЮ»

Завідувач кафедри проф. Шинкаренко Т.А.

"05" 09 **2022 року**

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТЦІ

Ковальчук Софії Андріївній

(прізвище, ім'я, по батькові)

1 .Тема роботи Комп'ютерне моделювання сцен з урахуванням заломлення променів світла

керівник роботи *Стельмащук Віталій Володимирович, канд. фіз.-мат. наук*,

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені Вченою радою факультету від " 13 " вересня 2022 року

№ 15

2. Строк подання студентом роботи 12 грудня 2022р.

3. Вихідні дані до роботи

4. Зміст магістерської роботи (перелік питань, які потрібно розробити)

А) огляд етапів створення 3D-об'єктів

Б) огляд основних методів рендерингу зображень

В) теоретичний огляд методу трасування променів та його оптимізацій

Г) програмна реалізація методу трасування променів та його оптимізацій у вигляді динамічної бібліотеки

Д) розробка десктопного застосунку для демонстрації роботи методу трасування променів

Е) провести аналіз отриманих результатів роботи алгоритму

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Рисунки, що ілюструють роботу алгоритму;

Знімки екрану роботи десктопної аплікації;
Результати рендерингу сцен;

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 04. 09. 2022р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Огляд літератури: етапи створення 3D-об'єктів, основні методи рендерингу зображень	вересень	виконано
2	Теоретичний розгляд методу трасування променів та його можливих оптимізацій	вересень	виконано
3	Імплементація методу трасування променів у вигляді динамічної бібліотеки побудованої за допомогою мови програмування Python	жовтень	виконано
4	Створення десктопного тестового застосунку засобами мови програмування C# та технології WPF для перевірки коректності імплементації обраного методу	жовтень	виконано
5	Тестування програмних засобів (динамічної бібліотеки та тестового застосунку)	листопад	виконано
6	Аналіз отриманих результатів роботи реалізованого алгоритму	листопад	виконано
7	Оформлення тексту магістерської роботи	листопад-грудень	виконано

Студентка _____ Ковальчук С. А.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Стельмащук В. В.
(підпис) (прізвище та ініціали)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

CG – Computer Graphics

GPS – Система глобального позиціювання

INS – Інерціальна навігаційна система

JSON – JavaScript Object Notation, укр. запис об'єктів JavaScript

OBJ – формат файлу опису геометрії

PPM – Portable Pixel Map (формат збереження кінцевого рендеру 3D-сцени)

RT – Ray Tracing

RC – Ray Casting

Scanline rendering – Алгоритм або візуалізація «Scanline»

WPF – Windows Presentation Foundation

XML – Extensible Markup Language

ВСТУП

На сьогодні комп’ютерне моделювання вважається однією з провідних галузей, яка використовується авіа-, машинобудівними фірмами під час конструювання та виготовлення виробів, які потребують високої точності, а основною задачею комп’ютерного моделювання є візуалізація, тобто створення зображення на основі опису (моделі) того, що потрібно зобразити. Поступово ця галузь увійшла у різні сфери повсякденного життя. До основних сфер застосування її технологій можна віднести такі:

- графічний інтерфейс користувача;
- спецефекти, візуальні ефекти, цифрова кінематографія;
- цифрове телебачення, відеоконференції;
- цифрова фотографія;
- цифровий живопис;
- візуалізація наукових та ділових даних;
- комп’ютерні ігри, системи віртуальної реальності (наприклад, тренажери з керування автомобілем, літаком тощо);
- системи автоматизованого проектування;
- комп’ютерна томографіях [4].

Цей перелік постійно розширюється, бо використання комп’ютерного моделювання у різних сферах знань дає поштовх новим напрямкам.

Моделювання графічних об’єктів використовує математичні методи, досягнення у галузях фізики, природничих наук. Для реалістичного зображення сцен необхідно враховувати як геометрію складових об’єктів сцени, так і фізичні властивості матеріалів, з яких виготовлені об’єкти, біологічну будову живих об’єктів тощо [4].

Очевидно, що середовище, яке нас оточує, ми бачимо завдяки тому, що всі предмети відбивають світло, пропускають його через себе, випромінюють. Для

передачі та збереження кольору в комп’ютерній графіці використовуються різні форми його представлення. У загальному випадку колір – набір чисел, які представляють собою координати у певній колірній моделі.

Для моделювання тривимірних об’єктів необхідно враховувати закони поширення світла у середовищах, властивості об’єктів відбивати, заломлювати, випромінювати світло [4].

У 2019 році компанія NVIDIA випустила відеокарти з підтримкою трасування променів у режимі реального часу [3]. Трасування променів (з англ. Ray Tracing, RT) – це технологія відтворення тривимірної графіки шляхом симуляції фізичної поведінки світла. Використовуючи її, відеокарта детально моделює проходження променів від джерел освітлення і їх взаємодію з об'єктами. При цьому, враховуються властивості поверхонь об'єктів, на підставі чого обчислюються точки початку розсіювання, особливості відображення світла, утворення тіней, тощо [27]. На жаль, попри науково-технічний прогрес, відеокарти даного типу досі не мають широкого використання у повсякденному житті багатьох із нас та події останніх років лише уповільнюють їхню доступність на світовому ринку бодай на 5 років.

У своїй роботі я вирішила проаналізувати існуючі методи рендерингу зображень для розуміння етапу застосування методів та технік візуалізації законів поширення світла, створити простими засобами власний аналог методу трасування променів для візуалізації просторових 3D-моделей без зайвих втрат якості фінального продукту для різних призначень: як для рендеру сцен з метою подальшої публікації, як зображення, так і для гри від 1-ї особи в режимі реального часу та побудувати прикладний програмний конвертор для відображення 3D-моделей з урахуванням заломлення променів світла з метою тестування створеного аналогу, дослідити час рендерингу на противагу існуючим методам і зробити певні висновки.

1 3D-МОДЕЛЮВАННЯ. ЕТАПИ СТВОРЕННЯ 3D-ОБ'ЄКТІВ

3D-моделювання – це процес створення моделі об'єкту у тривимірному просторі [9]. Створення 3D-моделі ж здійснюється в декілька етапів [7]:

1) Розробка ідеї та реалізація ескізу.
2) Скульптуринг (або моделювання) моделі.
3) Накладання текстур.
4) Додання джерел освітлення та розташування тіней.

Розглянемо детальніше кожний з етапів.

1.1 Розробка ідеї та реалізація ескізу

Перш за все, автор подає ідею, яку в подальшому планує реалізувати. До того, як починати моделювання на комп'ютері, деякі художники малюють ескіз на папері, роблять вибір щодо анатомії моделі, кольорів, текстур. Якщо ж проектується об'єкт, який існує у реальному світі, ескіз можна відобразити за фотографіями чи відео. Опісля, як замовник або режисер затвердять фінальні ескізи, їх передають спеціалістам з моделювання.

У випадку створення мультфільму група художників має розробити розкадровку, що надалі стає у нагоді як для надання зовнішнього вигляду персонажу, так і для анімації його рухів та міміки. [5]

1.2 Створення моделі

Існує доволі багато видів моделювання, проте для побудови моделей, що згодом використовуються для відеоконтенту, гри, тощо, використовують метод полігонального моделювання.

Полігональне моделювання – низькорівневе моделювання, яке дозволяє візуалізувати об'єкт за допомогою полігональної сітки [6].

Полігональна сітка складається з простих фігур (полігонів). Полігон являє собою трикутник або чотирикутник, який має вершини, ребра та грані. Модель, створену методом полігонального моделювання, можна назвати фігурою, яка складається з полігонів з різним ступенем перспективного спотворення, за рахунок чого об’єкт має певну форму. [11]

Регулюючи кількість полігонів, з якої складається об’єкт, можна регулювати рівень згладженості моделі. Наприклад, сфера з більшої кількості полігонів виглядає більш гладкою, ніж та, яка складається меншої кількості полігонів (рис. 1.1).

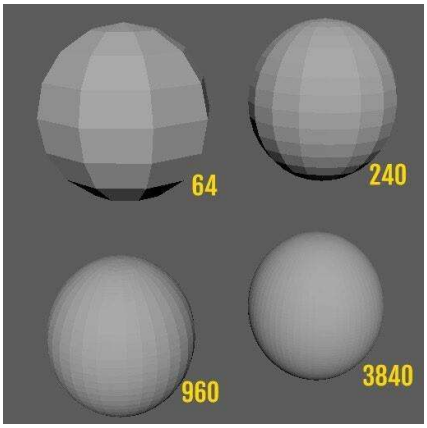


Рисунок 1.1 – Сфери, що складаються з різної кількості полігонів [10]

Також, чим більшою є кількість полігонів у моделі, тим більш деталізованою є модель. За таким критерієм моделі ділять на [8]:

- високополігональні (high-poly) – з великою кількістю полігонів;
- середньополігональні (mid-poly) – з середньою кількістю полігонів;
- низькополігональні (low-poly) – з малою кількістю полігонів.

Високополігональні моделі є більш фотореалістичними, ніж середньо- та низькополігональні, тому можуть використовуватися для створення фільмів та мультфільмів. Проте даний вид моделей потребує багато програмних та апаратних ресурсів. Рендеринг таких моделей займає багато часу, адже кожний

кадр підлягає попіксельному прорахуванню, тому на створення відеоконтенту може бути витрачено навіть декілька років.

Для комп'ютерних ігор високополігональні моделі не використовуються, тому що кадри б провантажувались доволі повільно. Щоб уникнути перевантаження програми та відеокарти комп'ютера, розробники виконують оптимізацію візуальної частини гри під можливості комп'ютерів користувачів.

За подібністю до високополігональної моделі створюють низькополігональну з полігонів-трикутників або полігонів-сфер. Для побудови низькополігональної моделі залишають лише ті полігони, які впливають на силует та форму. Елементи, які не можна буде побачити, видаляють [10]. До моделі додають джерела світла, регулюють світлі та темні ділянки (шейдинг). Далі обидві моделі аналізують на напрямки нормалей. Нормалі – це вектори, які використовуються для визначення того, як світло відбивається від поверхні. Програма будує промені за напрямками нормалей низькополігональної моделі. Коли ці промені стикаються з високополігональною моделлю, програма обчислює, як відобразити ці промені, щоб вони слідували у напрямку нормалей високополігональної моделі. Інформацію про напрямки нормалей програма зберігає у текстуру під назвою «карта нормалей» [10]. Процес аналізу та зберігання називають «запікання» (рис. 1.2).

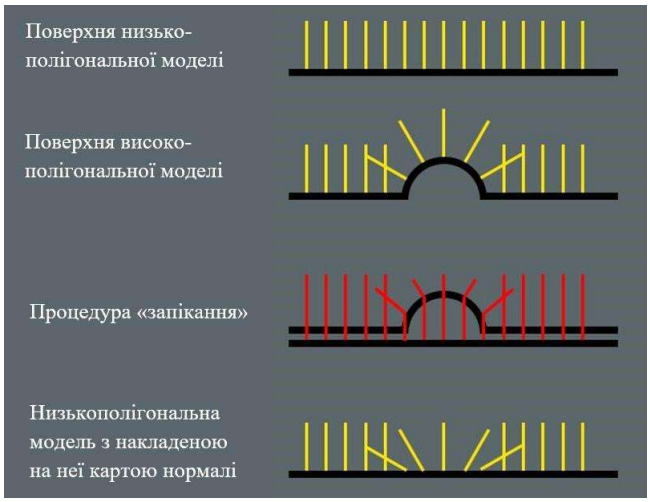


Рисунок 1.2 – Процес«запікання» [10]

Далі карту нормалей накладають на низькополігональну модель. Таким чином виникає оптична ілюзія: низькополігональна модель відбиває світло так само, як і високополігональна (рис. 1.3).

а б в

Рисунок 1.3 – Приклад оптимізації моделі: а – високополігональна модель; б – низькополігональна модель; в – низькополігональна модель з накладеною на неї картою нормалей [12]

Полігональне моделювання не має прив’язки до реальних одиниць вимірювання, тобто даний метод моделювання не підходить для створення моделей з точними розмірами, наприклад, з креслення або плану будівлі. Проте для створення відеоконтенту точність не має великого значення, більшу роль відіграє художня візуалізація об’єктів.

Якщо для створення відеоряду необхідне моделювання сцен, наприклад вулиці або місцевості, використовують метод автоматичної 3D-реконструкції за відеоматеріалами. Даний метод дозволяє отримати 3D-модель міської сцени у режимі реального часу.

Відеоматеріали знімаються багатокамерною системою в поєднанні з INS (інерціальною навігаційною системою) і GPS-вимірами. Для досягнення роботи у реальному часі виконують реконструкцію карт нормалей з наборів зображень з

подальшим злиттям зображень з картами нормалей. Алгоритм є простим і швидким, може бути реалізований на графічних процесорах. В результаті ми можемо отримати стисле і геометрично коректне представлення тривимірної сцени [13].

1.3 Накладання текстур

Накладання текстур на 3D-об’єкт – це відтворення фізичних якостей текстур та матеріалів, з яких виготовлено об’єкт для надання зображенню більшої реалістичності.

Текстура – це растрове зображення, що застосовується до полігональної моделі шляхом накладення з метою надання моделі фактурності, рельєфності і потрібного колірного забарвлення.

Якість текстурування об’єкта визначається такими одиницями як тексель. Тексель – це сукупність пікселів, що припадають на одиницю текстури. Формат і роздільна здатність картинки текстури, що використовується, безпосередньо визначають якість підсумкового результату.

Розрізняють декілька видів текстурування [14]:

- Рельєфне текстурування;
- MIP-текстурування;

Рельєфне текстурування – технологія роботи з 3D-графікою, що дозволяє створити поверхню об’єкта, що моделюється, реалістичною [10]. Рельєфне текстурування нагадує процес накладення текстури на полігон. Відмінність полягає в тому, що під час звичайного накладення текстури виконується робота з кольором і змінюється лише колірне сприйняття полігона, а під час рельєфного текстурування додається відчуття рельєфу, об’ємності плоскому полігону. Ця техніка може додати деталізацію сцені без створення додаткових полігонів, що

також є частиною оптимізації моделей для комп'ютерних ігор. Існують наступні види рельєфного текстурування [10]:

1) Створення рельєфної структури (bump mapping) – спосіб, що дозволяє надати поверхні об'єкта, що моделюється, ефект рельєфу та її деталізування. Даний ефект створюється шляхом зміщення пікселів за допомогою одноканальної карти висот (рельєф поверхні відображається у градаціях сірого кольору) і джерела світла. В результаті отримують ділянки з різним ступенем освітленості. Як правило, bump mapping застосовують, щоб створити не надто складні горбисті поверхні, пласкі виступи або западини [15].

2) Normal mapping – це метод зміни нормалі пікселя на базі кольорової карти нормалей. При цьому інформація про зміни зберігається у текселях. Даний метод є найбільш точним завдяки застосуванню трьох каналів текстур в карті нормалей [10].

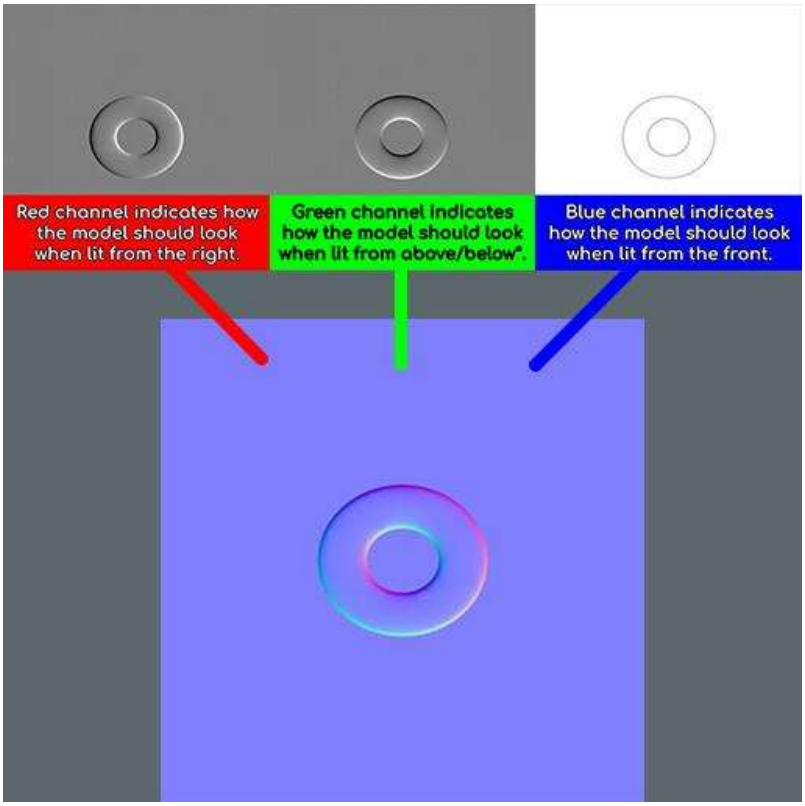


Рисунок 1.4 – Приклад зображення карти нормалей як набору із 3 сірих текстур (зліва направо перше, друге та третє зображення), що зберігаються на одному результуючому зображенні [10]

Перше зображення (див. рис. 1.4) повідомляє нашому ядру алгоритму, як ця модель має відбивати світло, якщо її освітити з правого боку, і воно зберігається в червоному каналі текстури нормальної карти.

Друге зображення (див. рис. 1.4) вказує механізму, як модель має відбивати світло при освітленні знизу, і воно зберігається в зеленому каналі нормальної текстури карти.

Примітка. Деякі програми використовують освітлення згори замість знизу, тому ми можемо використовувати «ліві» і "праві" нормальні карти, проте це може спричинити деякі проблеми у подальшому.

Третє зображення (див. рис. 1.4) повідомляє програмі, як модель має відбивати світло, коли її освітлюють спереду, і зберігається в синьому каналі текстури нормальної карти. Оскільки більшість речей виглядають білими, коли освітлені спереду, нормалі карти зазвичай виглядають блакитними.[10]

3) Parallax occlusion mapping – метод локального трасування променів, який використовується з метою визначення висот і видимості текселя. Завдяки цьому методу створюються більш сильні глибини рельєфу. Однак він не дає можливості ретельної деталізації об'єктів [16].

MIP-текстурування – метод, за якого під час накладення текстур застосовуються копії однієї і тієї ж ілюстрації текстури з різним ступенем промальовування деталей [16].

1.4 Розташування джерел освітлення та накладання тіней

Не зважаючи на матеріал (текстури) моделі, без ділянок, які освітлюються та на які падає тінь, кадри виглядатимуть доволі штучно та без об'єму. Складність додання світла до кадру полягає у достовірній імітації фізики розповсюдження світла, тому що промені мають багато властивостей, які важко візуалізувати. Наприклад, взаємодія світла з різними видами поверхонь, відбиття, заломлення, додання тіней, розсіювання, огинання об'єктів, зміна інтенсивності, яскравості, насиченості кольору матеріалу.

На перших етапах розвитку комп'ютерної графіки освітлення та його властивості виставляли вручну, тобто доступними були лише джерела прямого світла, які визначали лише напрям розповсюдження світла та на які ділянки об'єкта буде освітлено. Решту ефектів змінювали та доповнювали вручну. Згодом стало доступним глобальне освітлення, коли усі розрахунки розповсюдження світла у середовищі робить комп'ютер, причому для кожного окремого променя. Мова йде про технологію трасування променів.

Трасування променів (ray tracing) – технологія побудови зображення тривимірних моделей в комп'ютерних програмах, при яких відстежується зворотна траєкторія поширення променя. Спеціальний алгоритм відстежує шлях променя від об'єкта освітлення, а потім створює симуляцію того, як він взаємодіє з об'єктами: відбивається, заломлюється і так далі [17]. Спеціалісту залишається лише розставити джерела світла, їх напрям, віддаленість, а також налаштувати температуру світла та інтенсивність, а все інше візуалізує програма.

Накладання тіней (шейдинг) – процес, що здійснюють за допомогою шейдера – програми, що застововується у тривимірній графіці для визначення остаточних параметрів об'єкту або зображення. Шейдер може включати опис поглинання та розсіювання світла довільної складності, накладання текстури, відбиття та заломлення, затемнення, зміщення поверхні та ефекти пост-обробки.

Шейдери, що програмуються, є дуже ефективними і дозволяють за допомого простих геометричних форм візуалізувати складні з вигляду поверхні [23].

1.5 Висновки до розділу

В результаті огляду етапів та засобів створення 3D-об’єктів загалом проаналізовано метод створення 3D-моделі – на основі полігонального моделювання. Виявлено, що в залежності від сфери використання та типу рендерингу (повільний процес візуалізації для моделювання або рендеринг у реальному часі для комп’ютерних ігор) модель потребує оптимізації. Для полегшення рендерингу моделі у реальному часі виконують перетворення з високополігональної у низькополігональну модель. Відсутність деталізації компенсується текстуруванням (накладанням карт нормалей, рельєфу та ін.), шейдингом (поділом на світлі та темні ділянки сцени, доданням тіней), а також розташуванням та візуаізацією джерел освітлення за допомогою технології трасування променів.

2 АНАЛІЗ ОСНОВНИХ МЕТОДІВ РЕНДЕРИНГУ ЗОБРАЖЕНЬ

2.1 Аналіз існуючих методів побудови зображення

Для рендерингу зазвичай використовують 5 основних обчислювальні методів візуалізації. Кожнен з них має свій власний набір переваг і недоліків. Серед існуючих методів найбільш застосовуваними є:

- Scanline rendering and rasterization
- Ray casting
- Ray tracing
- Radiosity
- Z-buffer [18]

Scanline rendering – це метод, в основі якго лежить алгоритм для визначення поверхні, яку бачить глядач в комп'ютерній графіці, що працює на основі рядків за рядками замість полігонів та пікселів. Кожен багатокутник, що підпорядковується візуалізації, спершу сортують за верхніми координатами, у яких вони створюються спочатку, а потім кожен рядок сканування зображення обчислюється з використанням перетину лінії сканування з полігонами на передній частині відсортованого списку. Далі цей список оновлюється, щоб відкинути невидимі полігони, оскільки лінія сканування просувається вниз. [19]

Переваги:

- Сортування вершин вздовж нормальної площини сканування зменшує кількість порівнянь між ребрами;
- Відсутність потреби переводу координат всіх вершин з основної пам'яті в робочу;

Можливість інтегрування алгоритму з багатьма іншими графічними методами, такими як модель відображення Фонга або алгоритм Z- буфера;

Недоліки:

- За допомогою даного методу немає можливості отримати реальні відображення та заломлення;
- Застарілість алгоритму;
- Поступається альтернативним методам у реалістичному рендерингу зображення;
- Складна реалізація, оскільки система працює з об’єктним кодом;

Ray casting – це алгоритм прокладання променів, у якому змодельована геометрія аналізує кожен рядок та кожен піксель з точки зору назовні об’єкта. У місці, де перетинається об’єкт, значення кольору в точці може бути оцінено за допомогою декількох методів. У найпростішому випадку значення кольору об’єкта в точці перетину стає значення цього пікселя. Також колір може бути визначений з текстурної карти. Більш складним методом є зміна значення кольору за допомогою коефіцієнта освітленості, але без розрахунку відношення до імітованого джерела світла. Інший метод робить розрахунок на кут падіння світлових променів від джерела світла, а також виходячи із зазначених інтенсивностей джерел світла, обчислює значення пікселя. [20]

Переваги:

- Легко реалізувати;
- Інтуїтивно створює алгоритм Line of Sight;

Недоліки:

- Повільний порівняно з іншими методами;
- При відливанні лише декількох променів, квадрати поблизу джерела будуть відвідуватися багато разів;
- Багато артефактів, навіть у звичайних ситуаціях;

Radiosity – метод радіосигналу синтезу зображень у середині 1980-х років. У даному методі система дивиться лише на баланс світла (або енергії) в такому закритому середовищі, в якому вся енергія, що випромінюється або відбивається від даної поверхні, враховується шляхом відображення або поглинанням іншими поверхнями. За допомогою цього методу можна визначити величину поверхневого радіозв'язку та швидкість, з якою енергія виходить з поверхні. Для обчислення радіозв'язку для кожної поверхні використовуються значення кількості взаємодій енергії між ними. Зазвичай, за допомогою відповідних маніпуляцій, радіосигнал може генерувати зображення на льоту на відстані 15-20 кадрів в секунду. Розрахунки радіозв'язку не залежать від точки зору але збільшують обчислення, що корисні для будь-яких точок зору. Якщо відбувається невелика перестановка об'єктів радіозв'язку в сцені, однакові дані радіосигналу можуть бути повторно використані для ряду кадрів, що робить радіозв'язок ефективним способом поліпшення відливання від променів, без серйозного впливу на загальний час візуалізації. Через це радіозв'язок є основним компонентом провідних методів візуалізації в реальному часі і використовується від початку до кінця для створення великої кількості відомих останніх анімаційних 3D-мультфільмів. [21]

Переваги:

- обчислює дифузні взаємозв'язки між поверхнями;
- забезпечує перегляд незалежних рішень для швидкого відображення довільних переглядів;
- відтворює відносно реалістичні зображення;

Недоліки:

- 3D сітка вимагає більше пам'яті, ніж оригінальні поверхні;
- алгоритм відбору проб поверхні є більш сприятливим до артефактів зображення, ніж трасування променів;

- не враховує дзеркальних відображень або ефектів прозорості

Z-buffer – алгоритм, який використовується для визначення видимої поверхні та є основою процесу візуалізації сканованої лінії. Головна ідея використання цього алгоритму полягає в тому, щоб перевірити відстань від спостерігача кожної поверхні задля розробки найближчої поверхні кожного об’єкта. Якщо два об’єкти мають різні значення z-глибини вздовж однієї проектованої лінії, то більше значення знаходиться попереду, а позаду залишається ближча поверхня чи об’єкт. Застосування цього підходу дозволяє нам відображати сцени за допомогою візуалізації сканованої лінії. [22]

Переваги:

- простий у використанні;
- може бути легко реалізований в об’єкті або зображенні;
- може виконуватися швидко, навіть з багатьма полігонами;

Недоліки:

- займає багато пам’яті;
- неможливо створити прозорі поверхні без додаткового коду;

2.2 Обґрунтування обраного методу

Темою курсової роботи було обрано створення модулю для застосування у побудові зображення 3D моделей з відстеженням зворотної траєкторії променя, в основі реалізації якого лежить метод трасування променів, що має такі переваги:

- Елегантність алгоритму;
- Хороша апроксимація відображень у порівнянні з дослідженими методами;
- Здатність працювати з великою кількістю моделей водночас;
- Найбільш реалістичний і вживаний спосіб візуалізації на сьогоднішні;

- Імітування відбиття світла в реальному світі;
- Точність надання прямого освітлення, тіней, дзеркального відображення та ефекту прозорості.
- Ефективне використання ресурсів пам’яті;

У методі трасування променів, кожен піксель сцени, один чи більше променів світла відстежуються від камери до найближчого 3D-об’єкта. Світловий промінь проходить через задане число “відскоків”, що може включати в себе відбиття або заломлення променів в залежності від матеріалу моделі у 3D-сцені. Кожен колір пікселів обчислюється алгоритмічно на основі взаємодії світлового променя з об’єктами на його шляху (трасі). Ray tracing здатний більшого фотореалізму ніж інші методи рендерингу зображення, але його єдиним недоліком є складна обчислювальна спроможність.

2.3 Висновки до розділу

В цьому розділі були розібрані основні існуючі методи побудови зображення та їх переваги та недоліки. Був обґрунтований вибір теми дипломного проекту, та роз’яснено домінування обраного методу рендерингу зображення відносно альтернативних методів.

3 ТЕХНІКА ВІЗУАЛІЗАЦІЇ ВІДСТЕЖЕННЯ ПРОМЕНІВ

3.1 Огляд техніки візуалізації відстеження променів

Трасуванням променів називають метод обчислення видимості поміж точками. Транспортні алгоритми призначені для імітації способу поширення світла через простір при взаємодії з об’єктами. Вони застосовуються для обчислення кольору точки сцени. Трасування променів не вважають легким транспортним алгоритмом, це лише техніка обчислення видимості поміж точками. [22]

Загалом растрове зображення складається з пікселів. Один із способів відтворення 3D-сцени полягає в тому, щоб якось повертати це растове зображення вздовж площини зображення віртуальної камери, і знімати промені (Рис 3.1).

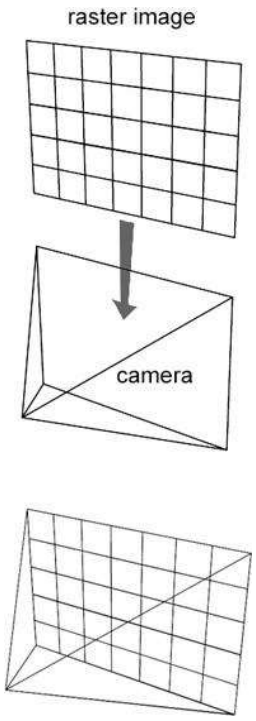


Рис. 3.1 – рух растового зображення перед площиною зображення камери [23]

За допомогою закидання променів, що походять від ока (положення камери), та які проходять через центр кожного пікселя ми знаходимо об’єкт зі сцени на якому ці промені перетинаються.

Якщо піксель щось і «бачить», то він бачить саме об’єкт, що знаходиться

прямо по напрямку, вказаного променем. Напрямок променя можна побудувати, якщо простежити лінію від походження камери до центру пікселя (Рис 3.2).

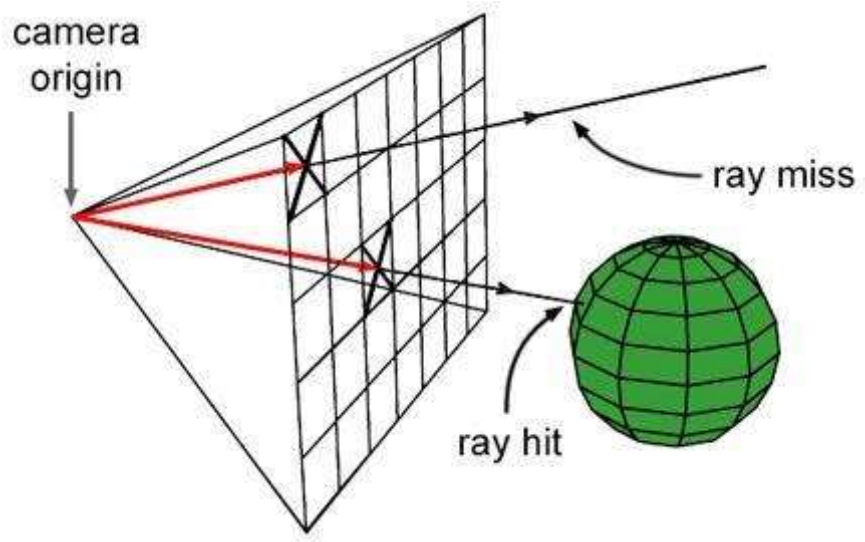


Рис. 3.2 – промінь може потрапити в геометрію сцени або пропустити її [23]

Тепер, коли ми знаємо, що бачить піксель, треба лише повторити цей процес для кожного пікселя зображення. Налаштовуючи колір пікселя відповідно до кольору об’єкта, в котрому кожен промінь проходить через центр кожного пікселя, ми можемо сформувати зображення сцени. Цей метод вимагає циклічного перегляду всіх пікселів у зображенні і і перетворення променя на сцену для кожного з цих пікселів. На другому етапі, етапі перетину, потрібно виконати цикл над усіма об’єктами сцени, щоб перевірити, чи перетинає промінь будь-яких з цих об’єктів [див 23].

Слід зауважити, що деякі промені можуть взагалі не перетинати будь-яку геометрію. Наприклад, як показано на малюнку 3.2, один з променів не перетинає сферу. У цьому випадку, як правило, треба залишити чорний колір пікселя, або встановити його на довільний інший колір. В наведеному вище коді ми ми задаємо колір пікселя кольором об’єкта в точці перетину.

Об’єкти в реальному світі виглядають дуже складно. Їх яскравість змінюється залежно від кількості світла, яке вони отримують, деякі з них блискучі, інші матові тощо. Мета фото-реалістичної візуалізації полягає не лише у точному зображенні геометрії, але й в імітуванні зовнішнього вигляду об’єктів переконливо (Рис 3.3).

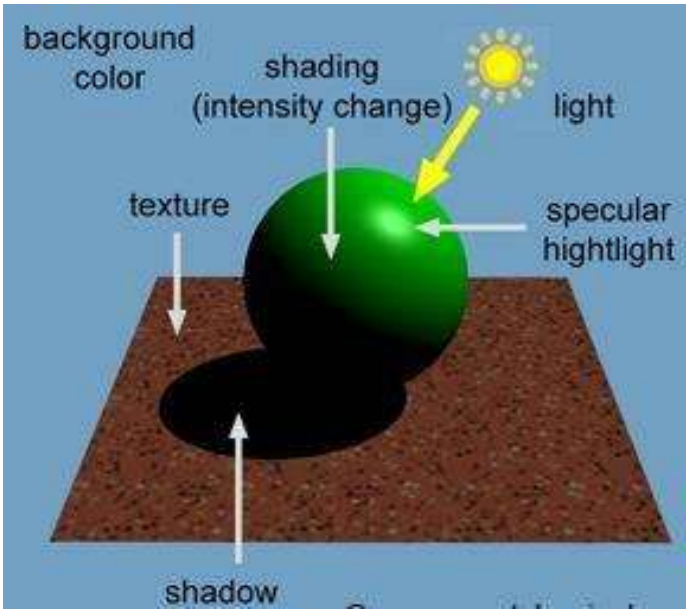


Рис. 3.3 – ілюстрація, що показує, як важко відтворити реальність [23]

Цей процес передбачає щось більш складне, ніж просто повернення постійного кольору. У комп’ютерній графіці, завдання визначення фактичного кольору об’єкта в будь-якій заданій точці на його поверхні, знаючи про ймовірні зовнішні (кількість світла, що отримує об’єкт) та внутрішні параметри (матерія об’єкта, блискучі кольори), називається затіненням.

Трасування променів вважається орієнтованим на зображення. Зовнішні петлі повторюють всі пікселі зображення і внутрішній цикл перебирає об’єкти сцени. Алгоритм растеризації є відносно орієнтованим на об’єкт. Він вимагає циклічного перегляду всіх геометричних примітивів сцени та проектування цих примітивів на екран.

Трасування променів слід використовувати для двох етапів

візуалізації: видимість та затінення. Це пов’язано з тим, що растеризація підходить тільки для видимості, і краща ніж трасування променів лише у швидкодії.

Зазвичай рендеринг займається обчисленням видимості між точкою в просторі і першою видимою поверхнею в заданому напрямку, або видимістю між двома точками. Перша задача призначена для вирішення проблеми видимості, друга - для вирішення таких проблем, як затінення. Растеризація дуже добре підходить для пошуку першої видимої поверхні, але неефективна для вирішення задачі видимості між двома точками. Трасування променів дозволяє ефективно обробляти обидва випадки. Пошук першої видимої поверхні корисний для вирішення проблеми видимості. Таким чином, і трасування променів, і растерізація підходять для вирішення цієї задачі. Затінення вимагає вирішення проблеми видимості між поверхнями, яке використовується для обчислення тіней,коли використовуються світлові області, і глобальні ефекти освітлення, такі як відображення, заломлення, непрямі відображення та непрямі дифузії. Таким чином, для цієї конкретної частини процесу візуалізації трасування променів є більш ефективним, ніж растеризація. Але майте на увазі, що будь-яка техніка, яка обчислює видимість між точками, може бути використана для затінення та вирішення проблеми видимості.

3.2 Етапи обчислення фотореалістичних 3d-зображень за допомогою трасування променів

Використання трасування променів для обчислення фото-реалістичних зображень може бути поділено на 3 етапи:

- Відливання променів на сцену;
- Тестування перехресть променевої геометрії;
- Затінення;

Перше, що потрібно зробити, щоб створити зображення за допомогою

трасуванням променів, це подати промінь для кожного пікселя зображення. Ці промені називаються камерами або первинними променями. Вторинні використовуються для пошуку точок сцени, що знаходяться в тіні, або для обчислення ефектів затінення, таких як відображення або заломлення. Коли первинний промінь відкидається в сцену, наступний крок полягає в тому, щоб знайти його перетин з будь-яким об’єктом у сцені.

Тестування променя на перетин з будь-яким об’єктом у сцені вимагає циклічного перегляду всіх об’єктів сцени і перевірки поточного об’єкта на промінь

Геометрія в 3D може бути визначена різними способами. Прості форми, такі сфера, диски, площини, можуть бути визначені математично або параметрично. Форми більш складних об’єктів можуть бути описані лише за допомогою полігонових сіток, поверхонь підрозділів поверхонь NURBS. Основна проблема з другою категорією об’єктів полягає в тому, що для кожної підтримуваної поверхні необхідно реалізувати методи перетину променевої геометрії. Наприклад, поверхні NURBS можна промалювати прямо, хоча рішення для цього дуже відрізняється від того, що використовується для тестування перетину між променем і багатокутною сіткою. Таким чином, для того щоб підтримувати всі типи геометрії, треба написати одну з перехресних процедур променевої геометрії для кожного підтримуваного типу, але це потенційно збільшує складність коду програми. Альтернативне рішення полягає в перетворенні кожного типу геометрії в одне і те ж внутрішнє уявлення, яке майже завжди буде триангулярною сіткою багатокутника (Рис 3.4).

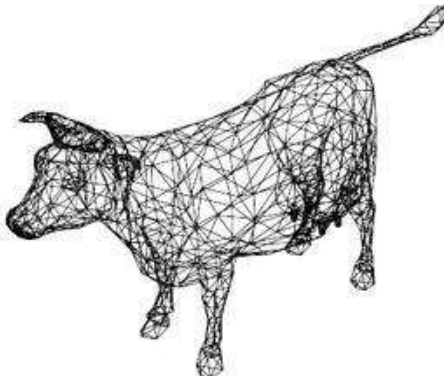


Рис. 3.4 – триангулярні сіткі зручніші для трасування променів, ніж інші типи геометрії [23]

Трапляється, що перетворення майже будь-якого типу поверхні на сітку багатокутника дуже просте. Таким чином перетворення полігональної сітки в триангулярну також досить просте. Крім цього, трикутники є вигідними з той точки зору, що вони можуть бути використані як базовий геометричний примітив для трасування променів та растеризації. Обидва алгоритми люблять трикутники через те, що вони мають цікаві геометричні властивості, яких не мають інші типи.

Таким чином, полігональні сітки або інші типи поверхонь перетворюються на трикутники. Це можна зробити або перед завантаженням геометрії в пам’ять програми, або під час візуалізації. Тепер не тільки кожен промінь камери повинен бути протестований на кожному об’єкті сцени, але і на кожному окремому трикутнику, який складається з кожного багатокутника в сцені.

Це означає, що час, необхідний для відтворення сцени трасування променів, прямо пропорційний кількості трикутників, які містить сцена. Всі трикутники повинні бути збережені в пам’яті, і кожен з них повинен бути перевірений на додавання променя в сцену. Висока обчислювальна вартість трасування променів є головним недоліком алгоритму [23].

3.3 Опис алгоритму трасування

Для реалізації алгоритму трасування променів і отримання результуючого зображення, яке складається з базових геометричних примітивів у просторі R^3, стануть у нагоді наступні програмні моделі:

- Модель променю у тривимірному декартовому просторі;
- Модель променевого емітера – це камера;
- Математичні моделі графічних примітивів (просторові фігури);
- Модель точкового джерела світла.

Для побудови первинного променю і визначення перших перетинань з об'єктами сцени введемо поняття джерела променів і площини. В основі моделі променевого емітера лежить механізм спрощеної камери-обскури (рис. 3.5) з нескінченно малим отвором, крізь який світло потрапляє на область огляду сцени.

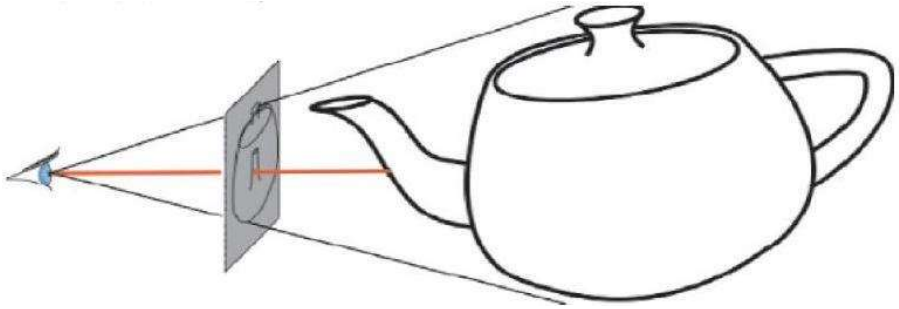


Рис. 3.5. Принцип побудови зображення. Механізм спрощеної камери-обскури [23]

Визначимо плоску область огляду як масив пікселей $\{n \times m\}$, де n і m - кількість пікселей в ширину та висоту відповідно, використовуючи попередньо визначені структури даних.

Встановимо початок координат у точку $O(x_0, y_0, z_0)$, центр камери у точку $C(x_c, y_c, z_c)$. Визначимо фокусний центр камери точкою $F(x_F, y_F, z_F)$. Вектор $\vec{\omega} = \overrightarrow{CF}$ визначає напрямок камери.

Для коректного визначення напрямку розповсюдження променю від позиції камери до центру кожного пікселя плоскої області огляду нам потрібно

сформувати ортогональний базис $\{\vec{u}, \vec{v}, \vec{\omega}\}$ за допомогою операції векторного перемноження:

$$\vec{u} = \vec{\omega} \times \vec{y}_{(0,1,0)}$$
$$\vec{v} = \vec{u} \times \vec{\omega}$$

Напрямок первинного променю $\vec{R}$ (рис. 3.6) визначається наступними формулами [див 23]:

$$\alpha = \tan\left(\frac{fovx}{2}\right) \cdot \left(\frac{i - \left(\frac{n}{2}\right)}{\frac{n}{2}}\right)$$
$$\beta = \tan\left(\frac{fovy}{2}\right) \cdot \left(\frac{\left(\frac{m}{2}\right) - j}{\frac{m}{2}}\right)$$
$$\vec{R} = \overrightarrow{OC} + \frac{\vec{\omega} + \alpha\vec{u} + \beta\vec{v}}{|\vec{\omega} + \alpha\vec{u} + \beta\vec{v}|}$$

де α, β - величини зміщення променю по осям X, Y області огляду; i, j - цілочисельні координати пікселя, для якого генерується промінь; *fovx*, *fovy* - горизонтальний і вертикальний кути огляду камери;

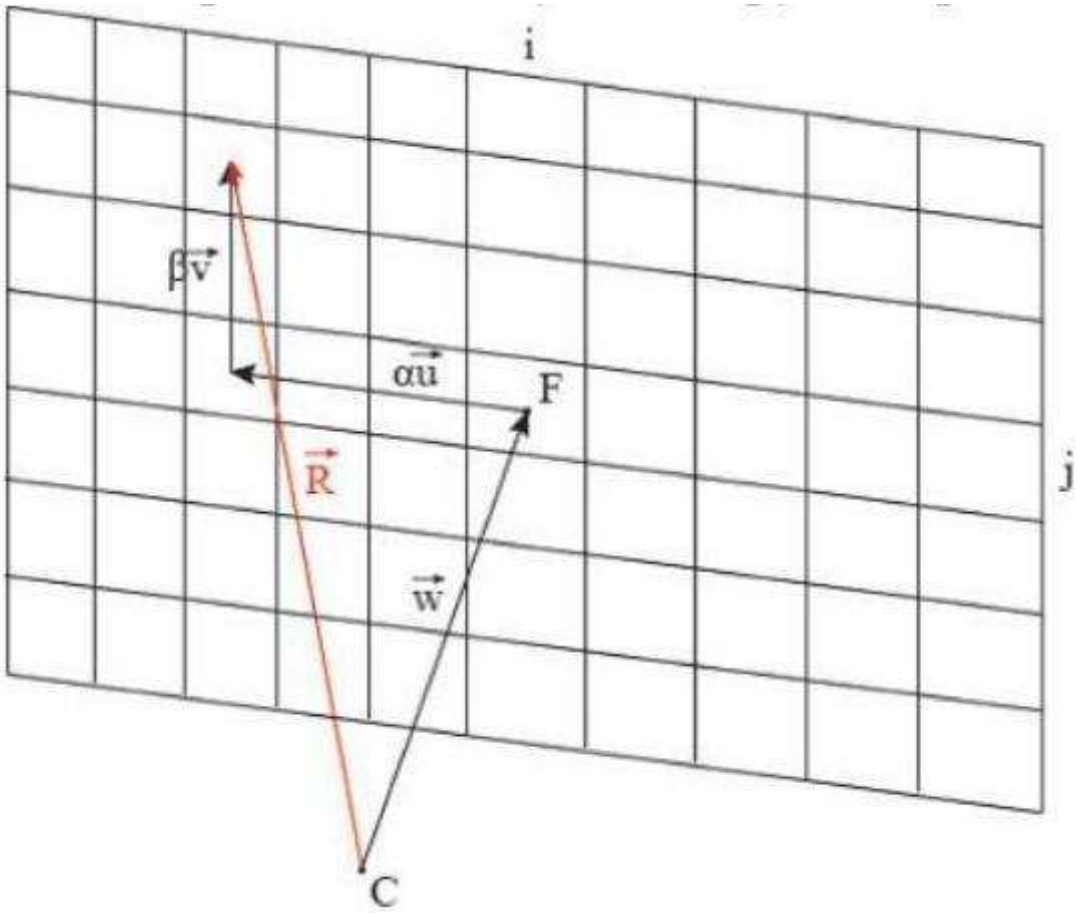


Рис. 3.6. Обчислення напрямку розповсюдження променю від позиції камери до центру кожного пікселя координатної площини

Нехай $$\vec{d} = \frac{\vec{\omega} + \alpha\vec{u} + \beta\vec{v}}{|\vec{\omega} + \alpha\vec{u} + \beta\vec{v}|^2}.$$

Таким чином можна записати векторне рівняння для пучка променів, які виходять з точки O з направляючими векторами $\vec{d}$:

$$\vec{R}_{ij}(t) = \overrightarrow{OC} + t\vec{d},\ t \geq 0, \quad (1)$$

де t - відстань від початку променю до будь-якої точки на ньому;

$\vec{d}$ - вектор напрямку розповсюдження променю.

Наведемо нижче послідовність рівнянь для визначення перетинів з наступними об'єктами, які використовуються у демонстраційній програмі: площина та сфера.

Розглянемо векторні рівняння площини, яка проходить через фіксовану точку Q перпендикулярно до вектору нормалі $\vec{n}$:

$$\vec{n} \cdot \overrightarrow{QP} = 0, \quad (2)$$

де $P(x_p, y_p, z_p)$ – довільна точка площини.

Вирішення системи рівнянь (1), (2)

$\begin{cases} \overrightarrow{OC} + t\vec{d} \\ \vec{n} \cdot \overrightarrow{QP} = 0 \end{cases}$ визначає значення параметра t, який відповідає точці перетинання променю площиною:

$$t = \frac{\vec{n} \cdot \overrightarrow{OQ}}{\vec{n} \cdot \vec{d}}.$$

Якщо $t < 0$, площина знаходиться позаду камери і промінь її не перетинає. Якщо $t \geq 0$, то точка перетину знаходиться на відстані $|\overrightarrow{OC} + t\vec{d}|$ від центру розміщення променевого емітера. Якщо $\vec{n} \cdot \vec{d} = 0$, то промінь проходить паралельно площині і не перетинає її.

Аналогічним чином вирішується задача про перетин променю та сфери з радіусом R та центром у точці $Q_d(x_s, y_s, z_s)$:

$$(x - x_s^2) + (y - y_s^2) + (z - z_s^2) = R^2.$$

Значення параметра t, при якому промінь перетинає сферу, визначаються коренями квадратного рівняння:

$$at^2 + bt + e = 0$$

де $a = |\vec{Q}_s|^2; b = 2\left((\overrightarrow{Q_sC}) \cdot \vec{d}\right); e = |\overrightarrow{Q_sC}|^2 - R^2$.

В залежності від значень a, b, e можливі наступні варіанти вирішення рівняння відносно t :

- Рівняння має два додатних дійсних корені. Промінь перетинає сферу у двох точках (входить і виходить з неї). Відповідно далі необхідно працювати з перетинами з найменшою дистанцією.
- Рівняння має два однакових додатних або від'ємних дійсних корені.
- Промінь проходить по дотичній до сфери.
- Рівняння має додатний і від'ємний дійсні корені. Променевий емітер знаходить у самій сфері і промінь тільки виходить з неї.
- Рівняння має комплексні корені. Промінь не перетинає сферу [23].

Встановивши мінімальне і максимальне значення параметра t, визначимо найближчу і найвіддаленішу площини відтинання тривимірної сцени, тим самим завершивши конфігурацію обсіченої піраміди огляду камери.

Реалізувавши алгоритм «трасування променів, описаний на попередньому етапі, сформуємо базову сцену, використовуючи підготовлені графічні примітиви. Зробимо наступні налаштування: площина: точка $(0, -1, 0)$, вектор нормалі $(0, 1, 0)$; сфера: точка $(0, 0, 0)$, одиничний радіус; джерело світла: точка $(-7, 10, -10)$. Встановимо відповідно перерахованим об'єктам коричневий, зелений та білий кольори.

Представлене вихідне зображення є результатом перевірки перетинання променів з об'єктами сцени. Для досягнення ефекту об'єму можна використати відомі моделі освітлення і затінення, зокрема:

- Ізотропні моделі дифузного освітлення Ламберта і Фонга [див. 23], остання з яких доповнює розсіяне освітлення поверхні блискучою складовою (скло, вода тощо);

- Моделі затінення поверхонь, яка візуалізує тіньовий об'єм.

У найпростішій своїй реалізації технологія тіньового об'єму у трасуванні променів полягає в генерації «тіньового променю» з точки перетину сцени первинним променем у напрямку джерела світла. Без врахування коефіцієнта затухання світла, якщо «тіньовий промінь» досягнув джерела світла, не перетинаючи при цьому інших об'єктів сцени, то точка освітлюється. В іншому випадку, якщо знайдено хоча б один перетин – тоді точка знаходиться у тіні. При правильній програмній реалізації моделей освітлення і затінення результуючі зображення наближаються до реалістичних (рис. 3.7).

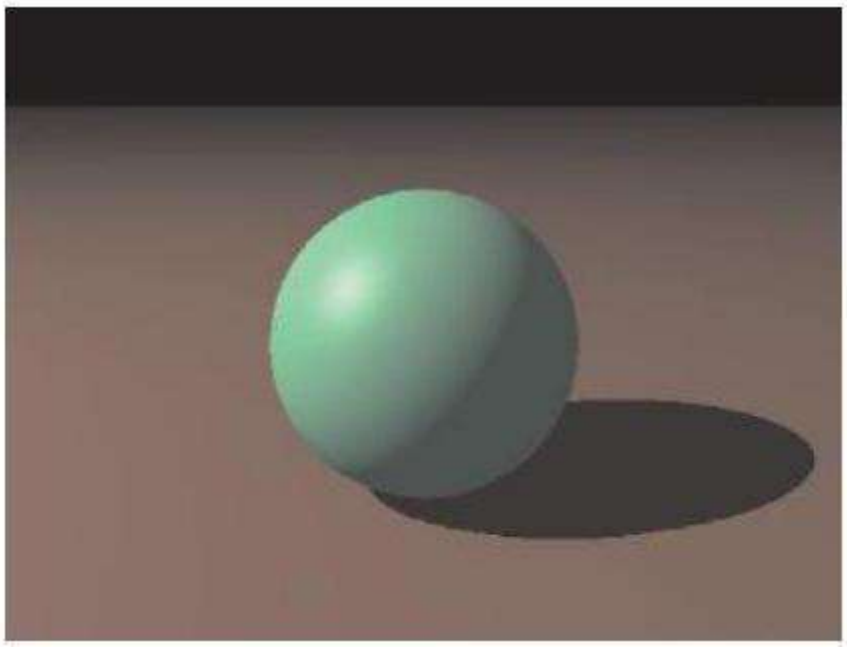


Рис. 3.7. Результат генерації тіней і роботи моделі освітлення Бліна-Фонга. Результат роботи побудованої бібліотеки на базі описаного алгоритму

Для розрахунку характеристик кольорової компоненти відбиття (віддзеркалення) у точці перетину первинного променю і об'єкту необхідно створити новий промінь у цій точці і направити його по нормалі, що відновлена з цієї точки по відношенню до поверхні об’єкта, потім рекурсивно повторити загальну процедуру отримання кольору для первинного променю. У даній роботі використовується модель ідеальних віддзеркалень. Саме тому для виключення

нескінченного формування променів для взаємного розрахунку кольору відбиття об'єктів, поверхні яких будуть наділені такою властивістю, необхідно ввести обмежувальне правило: поверхня повинна відбивати не всю енергію променю, а тільки певний відсоток.

Після імплементації алгоритму генерації додаткових променів для формування віддзеркалень ускладнимо сцену, додавши примітив (сферу) і присвоївши шаховий шаблон площини для надання наглядності роботи нового алгоритму, отримаємо наступний результат (рис. 3.8).

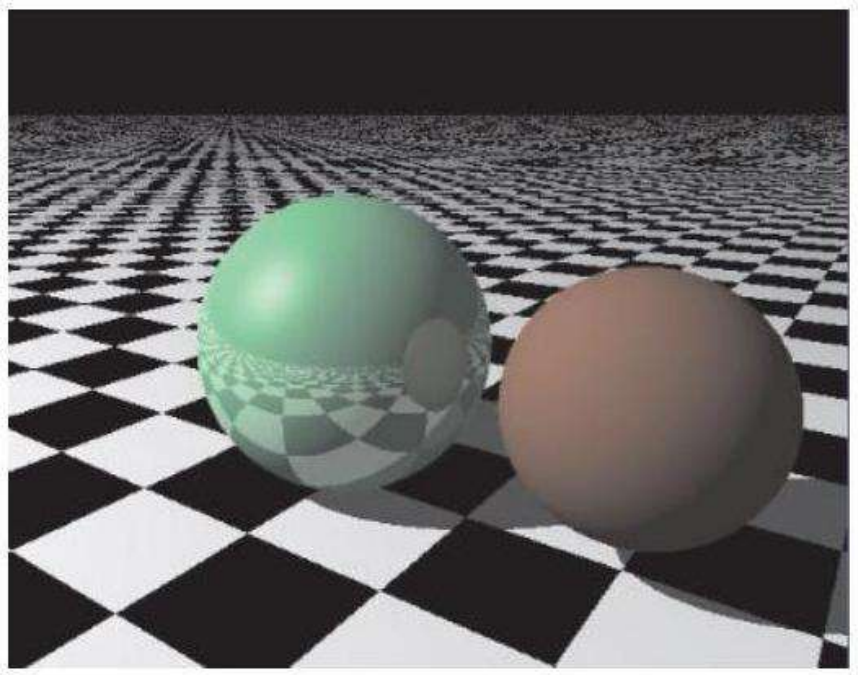


Рис. 3.8. Вихідне зображення після генерації дзеркальних відбиттів на поверхні сфери. Результат роботи демонстраційної програми Серйозним недоліком методу є невисока продуктивність. Метод растеризації і сканування рядків використовує когерентність даних, щоб розподілити обчислення між пікселями. В той час як метод трасування променів кожного разу починає процес визначення кольору пікселів по-новому, розглядаючи кожного разу трасований промінь окремо.

Якщо через кожен піксель області огляду проводити по одному променю, то гладкі лінії, в тривимірному просторі, після побудови проекції стануть східчастими на екрані. Щоб уникнути цього, необхідно генерувати декілька

променів на кожен піксель, рахувати для кожного колір і знаходити його середнє значення.

До переваг методу трасування променів можна віднести можливість рендерингу гладких об’єктів без апроксимації їх полігональними поверхнями.

3.4 Висновки до розділу

У комп’ютерній графіці концепція «зйомки» променів чи світла від очей спостерігача називається трасуванням шляху. Термін трасування променів також може бути використаний, але концепція трасування шляху дозволяє припустити, що цей спосіб створення зображень, що генеруються комп’ютером, спирається на шлях від світла до камери, або навпаки. Моделюючи це фізично реалістичним чином, ми можемо легко моделювати оптичні ефекти, такі як акустика або відображення світла іншою поверхнею в сцені. Підсумовуючи, важливо пам’ятати, що процедуру візуалізації можна розглядати як два окремі процеси в цілому. Перший з них визначає, чи видно точку на певному пікселі, а другий – тінь. Алгоритм елегантний і потужний, але змушує нас торгувати часом візуалізації для точності або навпаки.

4 РОЗРОБКА ПРОГРАМНИХ ЗАСОБІВ ДЛЯ ВІДСТЕЖЕННЯ ШЛЯХУ ПРОМЕНІВ НА ЗАДАНІЙ СЦЕНІ

Щоб застосувати наведену вище техніку на практиці, я вирішила побудувати динамічну бібліотеку для застосування у різних сферах: стиснення дизайнерських файлів у 3D-моделюванні, рендеринг сцени під час гри від першого обличчя у ґеймдевелопменті та протестувати отримані результати на створених власноруч та готових програмних рішеннях.

4.1 Обрані технології

4.1.1 Динамічні бібліотеки

Спершу я визначилась з відображенням виконуваних файлів. В результаті застосування вище зазначеної техніки, я зупинилась на побудові власної динамічної бібліотеки, яка з використанням невеликої кількості коду (див. Додаток А) може бути застосована в багатьох галузях, даючи якісний результат рендерингу (візуалізації) 3D-сцен в зображення в цілому для дизайнерських цілей, так і здатна швидко візуалізовувати зміну положення в грі від першої особи при використанні у засобах для розробки ігор.

Але для початку хотіла б конкретизувати, для чого обрала саме відображення даного методу як динамічну бібліотеку. Статичне компонування виконуваних файлів має низку недоліків.

1) Якщо декілька різних прикладних програм використовують спільний код (наприклад, коди функцій бібліотеки мови С), кожний виконуваний файл буде мати окрему копію цього коду; в результаті такі файли займатимуть більше місця на диску і в пам’яті. Варто мати можливість зберігати на диску і завантажувати в пам’ять лише одну копію спільного коду.

2) У разі кожного оновлення прикладну програму потрібно повністю перекомпілювати, перекомпонувати, повторно тестувати і перевстановити.

3) Неможливо реалізувати динамічне завантаження програмного коду під час виконання, наприклад, якщо потрібно реалізувати модульну структуру програми, подібно до того, як це зроблено в ядрі Linux.

Для вирішення цих і подібних проблем було запропоновано концепцію динамічного компонування з використанням динамічних або розділюваних бібліотек (dynamic-link libraries (DLL), shared libraries). Динамічна бібліотека – набір функцій та програмних ресурсів, скомпонованих разом в бінарний файл, який можна динамічно завантажити в адресний простір процесу, що використовує ці функції. Динамічне завантаження (dynamic loading) – завантаження під час виконання процесу, зазвичай реалізоване як відображення файла бібліотеки в його адресний простір. Динамічне компонування (dynamic linking) – компонування образу виконуваного файлу під час виконання процесу з використанням динамічних бібліотек.

Розглянемо переваги і недоліки використання динамічних бібліотек.

Переваги

- Оскільки бібліотечні функції містяться в окремому файлі, розмір виконуваного файлу стає меншим. Якщо врахувати, що є динамічні бібліотеки, які використовують майже всі застосування у системі (стандартна бібліотека мови C у Linux, бібліотека підсистеми Win32 у Windows), то очевидно, що так заощаджують дуже багато дискового простору.
- Якщо динамічну бібліотеку використовують кілька процесів, у пам'ять завантажують лише одну її копію, після чого сторінки коду бібліотеки відображаються в адресний простір кожного з цих процесів. Це дає змогу ефективніше використовувати пам'ять.
- Оновлення застосування може бути зведене до встановлення нової версії динамічної бібліотеки без необхідності перекомпонування тих його частин, які не змінилися.

- Динамічні бібліотеки дають змогу застосуванню реалізувати динамічне завантаження модулів на вимогу. На базі цього може бути реалізований розширюваний API застосування. Для додавання нових функцій до такого API стороннім розробникам достатньо буде створити і встановити нову динамічну бібліотеку, яка підлягає певним правилам.
- Динамічні бібліотеки дають можливість спільно використовувати ресурси застосування (наприклад, така бібліотека може містити спільний набір піктограм), крім того, вони дають змогу спростити локалізацію застосування (якщо всі рядки, які використовуються програмою, помістити в окрему DLL, для заміни мови застосування достатньо буде замінити тільки цю DLL).
- Оскільки динамічні бібліотеки є двійковими файлами, можна організувати спільну роботу бібліотек, розроблених із використанням різних мов програмування і програмних засобів, що спрощує створення застосувань на основі програмних компонентів (отже, динамічне компонування лежить в основі компонентного підходу до розробки програмного забезпечення). Недоліки.

Динамічні бібліотеки не позбавлені недоліків (хоча вони тільки в окремих випадках виправдовують використання статичного компонування).

- Використання DLL сповільнює завантаження застосування. Що більше таких бібліотек потрібно процесу, то більше файлів треба йому відобразити у свій адресний простір під час завантаження, а відображення кожного файлу забирає час. Для прискорення завантаження рекомендують укрупнювати DLL, об'єднуючи кілька взаємозалежних бібліотек в одну загальну.
- У деяких ситуаціях (наприклад, під час аварійного завантаження системи із дискети) використання спільних системних DLL неприйнятне через нестачу дискового простору для їхнього зберігання (такі системні DLL, подібно до стандартної бібліотеки мови C, можуть займати кілька мегабайтів дискового простору, при цьому застосування часто потребують усього по кілька функцій із них). У такій ситуації найчастіше використовують версії застосувань, статично

скомпоновані таким чином, щоб у їхні виконувані файли був включений зі стандартних бібліотек код лише тих функцій, які їм потрібні.

Хоч динамічні бібліотеки мають деякі недоліки, загалом переваги використання цього способу позитивно впливають на загальний результат обробки вхідних даних.[2]

4.1.2 Мова програмування Python

Для розробки динамічної бібліотеки я вирішила використати мову програмування Python через низку переваг у порівнянні з іншими мовами, які ми вивчали протягом навчання у ВНЗ. Python - інтерпретована об'єктно-орієнтована мова. Її також називають мовою програмування високого рівня. Розширення імені файлів: .py, .pyc, .pyw, .pyo. Це забезпечує пряме і просте програмування як для малих, так і для великих програм. Основний акцент робиться на повторному використанні коду, читанні та використанні пробілів. Python використовує вирази в основному подібні до мови C та її методів та введення тексту. Python підтримує декілька парадигм програмування, таких як функціональне програмування, імператив та процедур.

Загалом Python можна використовувати для розробки різних додатків, таких як веб-додатки, графічні додатки на основі інтерфейсу, програми для розробки програмного забезпечення, наукові та числові програми, мережеве програмування, ігри та 3D-додатки та інші бізнес-програми. Це створює інтерактивний інтерфейс і можливість простої розробки додатків [24].

Використання Python також допомагає легко отримувати доступ до бази даних. Вона використовується для стандартного API бази даних і вільно доступна для завантаження.

Також одним з найважливіших критеріїв вибору стало те, що Python використовується для спрощення складного процесу розробки програмного забезпечення, оскільки це мова програмування загального призначення. Вона

використовується для розробки такого складного додатку, як науковий та цифровий додаток, а також для настільних і веб-додатків [24].

4.1.3 Тестова реалізація з використанням програмних рішень мови програмування C#

Але для демонстрації коректної роботи динамічної бібліотеки я вирішила обрати власноруч створену аплікацію мовою програмування C#. Щодо обраної мови: я хотіла відобразити, що бібліотека є ефективною не залежно від мови програмування, в якій ми хочемо її застосувати. Також здебільшого програмні десктопні реалізація для 3D-моделювання часто написані саме цією мовою. Я ж вирішила використати технологію WPF з застосуванням цієї мови для побудови невеликого компресора візуалізації фінальних моделей, який може зекономити час у сфері дизайну. На противагу веб-реалізації, використання програмної аплікації, яку можна завантажити на персональний комп’ютер, зекономить ресурси при обробці завантажених файлів, які можуть бути громіздкими.

4.2 Динамічна бібліотека на базі програмної реалізації техніки візуального відстеження променя мовою Python

Спершу я склала завдання на зміст функцій DLL. Оскільки необхідний функціонал вже описано у 3.2, слідкуючи за методами, я спершу реалізувала базові класи для реалізації алгоритму:

- Точка, Вектор, Промінь, Перетин

- Зображення

- Полігон трикутників, площина, чотирикутник як похідний клас від площини

- Сфера, Циліндр

- Колір, Матеріал

- Сцена, Світло, Базовий двигун рендерингу заданих об’єктів.

Кожна функція, додана в бібліотеці – з параметрами. Дотримувалась загальних вимог до параметрів: в основному використовувала стандартні типи даних.

В процесі розробки єдиний спосіб створити DLL з python-файлів – за допомогою компілятора, який перетворить python-код у C, по суті, я зробила багато зайвої роботи, але це допомогло мені набути новий досвід у роботі з Python. Все ж я не зупинилась, та перетворила файл з розширенням .py у .pyx. Далі завантажила Cython для спрощення інтеграції з кодом мовою C та перетворила запрограмований файл.

Останнім кроком створила окремий проєкт, використовуючи середовище Visual Studio: New Project → Windows Desktop → Dynamic Link Library (DLL) та додала до нього отриманий код. В результаті отримала папку Debug з наступними файлами (файлу з розширенням .lib у кінцевій програмі отримано не було, на противагу при реалізації бібліотеки іншою мовою (до прикладу C++)).

4.3 Побудова WPF-аплікації для тестування коректної роботи бібліотеки

У ході курсу Програмної Інженерії, який вивчають студенти ВНЗ, я дізналась базові основи для побудови власної десктопної аплікації з використанням засобів Windows. Оперуючи отриманими знаннями, створила просту аплікацію з 2 сторінками для порівняння роботи методу трасування променів у засобах рендерингу 3D-моделей та роботи удосконаленої моделі цього методу, засобами, які наведено у наступному пункті (дивись 4.4).

Для реалізації базових операцій додала меню, яке дозволяє Очистити поле даних, додати новий файл (у форматах JSON, XML), зберегти існуючий результат (у форматі PPM) або зберегти результат, конвертувавши його в зображення

обраного типу (JPG, PNG, BMP).

Саме формати JSON та XML були обрані, як вхідні, оскільки не потребують застосування додаткового програмного забезпечення на персональному комп'ютері для їх використання, а також попереднього досвіду роботи у 3D-моделюванні для швидкої маніпуляції з об'єктами. У файлах цих форматів на вхід ми подаємо основну інформацію про розмір нашої сцени, розташування основних об'єктів на ній у формі масиву, розташування камери (глядача) та інформацію про джерела освітлень: розташування, тип джерела (чи передає тепле або холодне освітлення). Формат зберігання PPM був обраний, оскільки, хоч він і є рідкістю у наш час, бо зазвичай не використовується, проте з його використанням я отримую більш достовірну інформацію про час рендеру, не витрачаючи додаткових зусиль на переформатування вихідних даних. Інформація про конкретне зображення зберігається в PPM-файлі в текстовому форматі. Кожен піксель файлу позначений певним числом (від 0 до 65536) і містить інформацію про висоту й ширину зображення, а також про всі пробіли. Реалізацію можна переглянути у додатку А [див 1].

Також є можливість перемикатись між методами з меню та переглянути вікно довідки, щоб отримати загальну інформацію про візуалізацію та стиснення вхідного файлу.

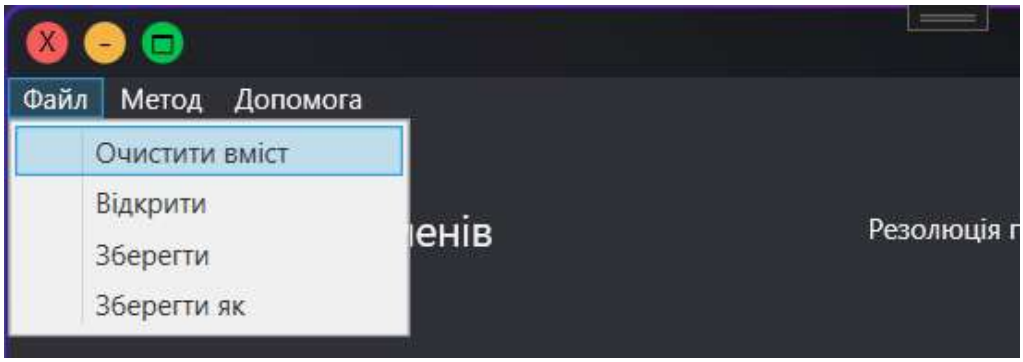


Рис. 4.1 – базові операції для роботи з вхідними/вихідними файлами тестової програми

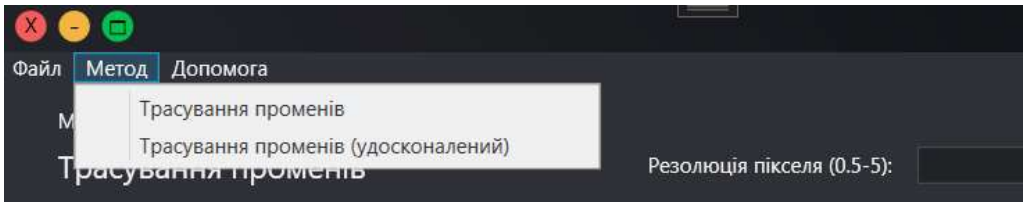


Рис. 4.2 – існуючі методи (удосконалений більш детально розглянутий у розділі 5)

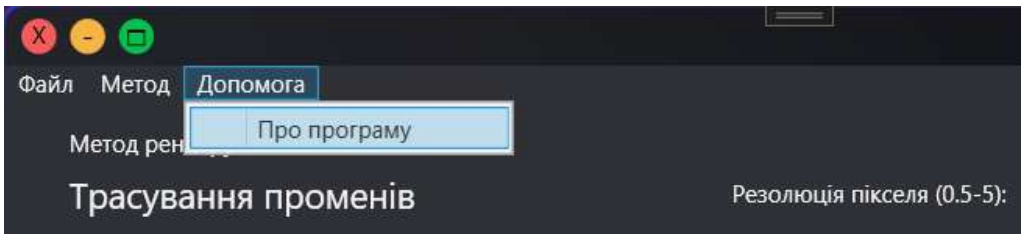


Рис. 4.3 – можливість переглянути вікно з довідковою інформацією про метод, який використовується для рендерингу сцени

Окрім завантаження, необхідно було надати користувачу можливість обрати резолюцію пікселя, або загальну щільність [див. 23]. Натиснувши кнопку “Почати” користувач отримує вихідний рендер заданої моделі:

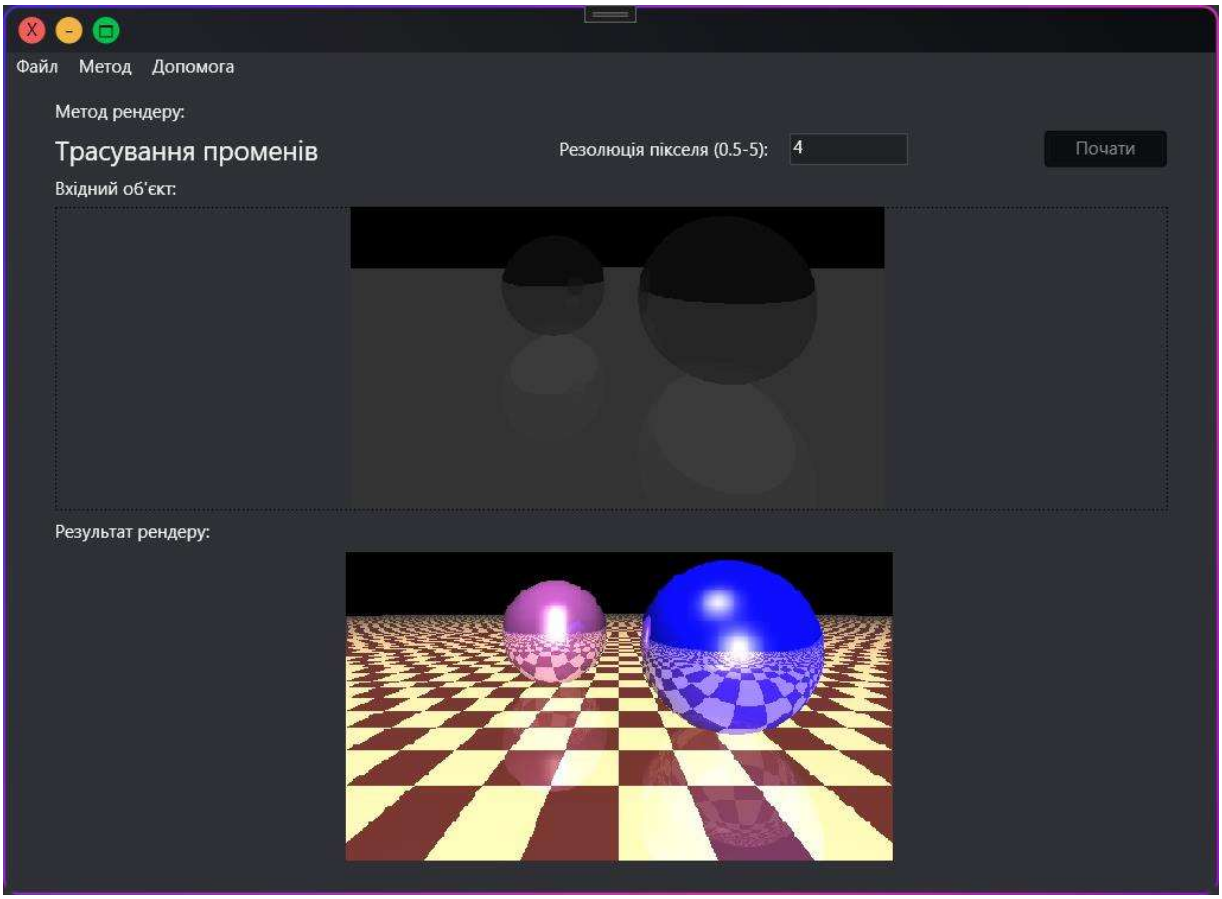


Рис. 4.4 – результат роботи WPF-аплікації

Примітка. Через високу резолюцію фінальний рендер не чітко відображається через стиснення при спробі зробити скріншот аплікації.

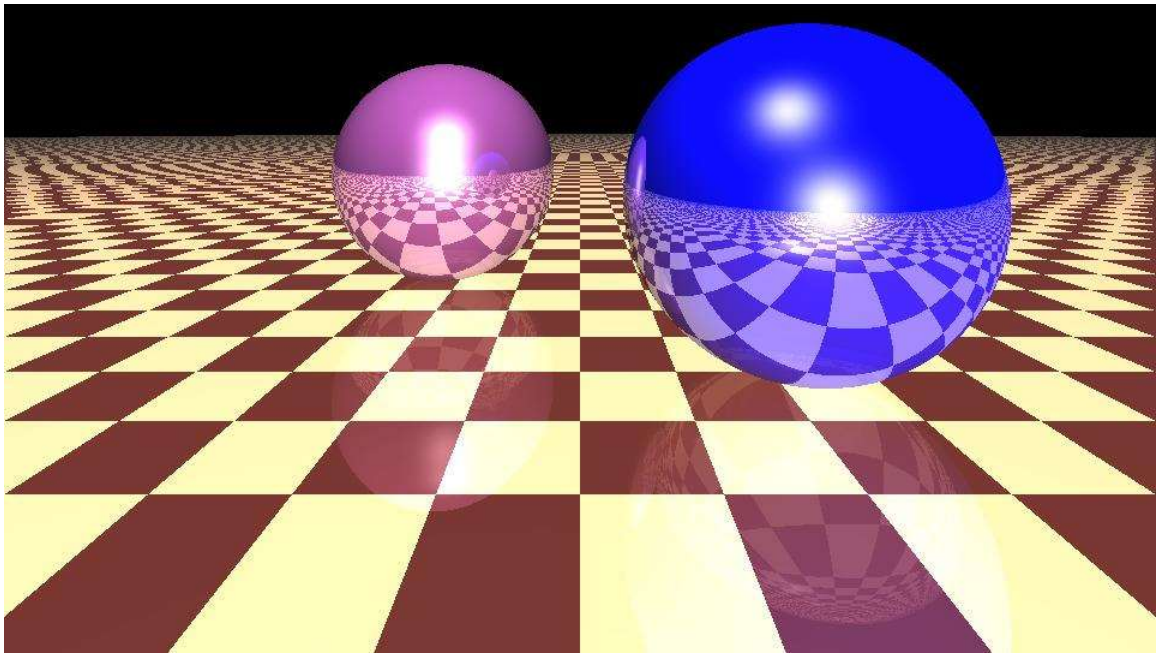


Рис. 4.5 – результат рендеру з Рис. 4.4

4.4 Методи оптимізації системи

Як було проаналізовано, метод трасування променів – це повністю функціональний алгоритм, але потребує значних обчислювальних витрат. Тому в ході досліджень та тестування були реалізовані деякі ідеї для прискорення роботи трасування.

Найбільш очевидний спосіб пришвидшення роботи трасування променів полягає в тому, щоб трасувати декілька променів одночасно. Оскільки кожен промінь, який виходить з камери, незалежний від усіх інших, та більшість структур потрібні лише для перегляду, можна трасувати по одному променю на кожне ядро центрального процесора без яких-небудь складностей через проблем з синхронізацією.

Паралелізація. Насправді, алгоритм трасування променів вважається надзвичайно паралельним, тому що саме їх суть дозволяє дуже просто їх розпаралелювати. Для реалізації розпаралелювання у бібліотеці я обрала додання аналогічного методу, який візуалізовує шлях променів, розділяючи поле на рівні горизонтальні лінії, задані на етапі тестування. З недоліків використання даного удосконалення варто зазначити, що при візуалізації горизонтальних площин на сцені кінцевий результат може бути викривлений у випадку, якщо площина потрапляє на межі горизонтального поділу сцени.

Кешування даних. Зазвичай, трасування променів більше всього часу витрачає на обрахунок значень для променів сфери, але деякі значення розташування сфери, після їх виявлення, залишаються незмінними. Можна їх обрахувати один раз під час завантаження сцени та зберігати їх в самих сферах. Це було додано на етапі реалізації минулого розділу, при створенні класів для збереження деяких просторових фігур.

Оптимізація затінення. Якщо точка об’єкту в тіні знаходиться відносно джерела світла, тому що на її шляху знаходиться інший об’єкт, то ми отримуємо велику ймовірність того, що сусідня з нею точка, через той самий об’єкт, також знаходиться в тіні відносно джерела світла (Рис 5.1). [23]

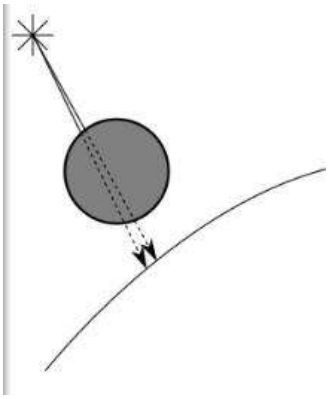


Рис. 5.1 – дві затіненні точки відносно одного джерела світла та об’єкту затінення [23]

Тобто коли ми шукаємо об’єкти між джерелом світла та заданою точкою,

ми повинні спершу перевірити, чи не накладається на поточну точку тінь від найближчого до неї об’єкту, що наклав тінь на попередню точку, відносно того ж самого джерела світла. У випадку, коли це так, ми можемо завершити, а якщо ні, треба продовжити звичайним методом перевіряти інші об’єкти.

Аналогічно, при обчисленнях перетину між променем світла та об’єктами нам не потрібно знати найближчий перетин, достатньо лише знати, що існує хоча б один перетин. Отже, використовується спеціальна версія алгоритму з урахуванням методу Монте-Карло, яка повертає результат, коли знаходиться перший перетин, і для цього потрібно повертати лише булеве значення.

Просторові структури. Обрахування перетину кожного променя з кожною сферою досить велика втрата часу. Існує велика кількість структур, яка дозволяє легко відкидати цілі групи об’єктів, що не потребують обчислень перетинів.

Уявімо, що є декілька сфер, які знаходяться досить близько один до одного. Можна обрахувати центр та радіус найменшої сфери, яка містить в собі всі ці сфери. Якщо промінь не перетинає цю граничну сферу, то можна бути впевненим, що він не перетинає сфери, які містяться в головній. Це можна зробити за допомогою одної перевірки перетину, але якщо промінь перетинає сферу, то все одно треба перевіряти наступні сфери на перетин з ним.

Субдискретизація. Існує доволі простий спосіб зробити трасування променів в N разів швидше, тобто обраховувати в N разів менше пікселів.

Уявімо, що трасуються промені для пікселів (10, 100) та (12, 100), та вони падають на один об’єкт. Можна логічно припустити, що промінь для пікселя (11, 100) також буде припадати на той же об’єкт, пропустити початковий пошук перетинів з усією сценою, та перейти до обрахування кольору в цій точці. [25]

Якщо зробити так в горизонтальному та вертикальному напрямках, то можна виконувати максимум на 75% менше початкових обрахувань перетинів

променів зі сценою.

Але в даному методі оптимізації існує один суттєвий недолік. Можна дуже просто пропустити дуже тонкий об’єкт, та результати використання оптимізації не будуть схожі на ті, що отримуються без неї. Головна ідея полягає в тому, щоб здогадатися, коли слід використовувати цей метод правильно, для отримання задовільних результатів.

Суперсемпінг. Суперсемпінг – це протилежність методу субдискретизації, якщо ми віддаємо перевагу точності замість швидкості. Уявімо, що промені, які відповідають двом сусіднім пікселям, припадають на два різних об’єкта. Нам потрібно розмалювати кожен піксель у відповідний колір.

Кожен промінь повинен задавати колір для кожного квадрату сітки, через яку ми дивимося. Використовуючи по одному променю на піксель, ми умовно вирішуємо, що колір променя світла, який проходить через середину квадрату, визначає цілий квадрат, але це не завжди так.

Для вирішення цієї проблеми можна трасувати декілька променів на піксель – 5, 10, 17, і так далі, а потім брати середнє значення, для отримання кольору пікселя. Це призводить до того, що трасування променів становиться повільніше в 5, 10 або 17 разів, завдяки тій же самій причині, по якій субдискретизація робить його в N разів швидше. Але існує компроміс. Можна припустити, що властивості об’єкта на його поверхні змінюються плавно, тобто трасування 5 променів на піксель, які припадають на один об’єкт в сусідніх точках, не дуже сильно поліпшить вид сцени. Тому можна почати з одного променя на піксель та порівнювати сусідні промені: якщо вони припадають на інші об’єкти, або їх колір відрізняється більше ніж на порогове значення, то застосовуємо до них підрозділ пікселів. [6] У своєму методі на противагу точності, я вирішила обрати швидкість для більшої вибірки етапу тестування, адже рендер 3D-сцен в залежності від початкового розміру вхідного файлу може займати більше декількох днів.

4.5 Висновки до розділу

У цьому розділі я детально розібрала причини вибору конкретного середовища розробки та обраної мови програмування для розробки бібліотеки. Розробила програмний модуль, а також тестовий десктопний додаток для перевірки коректності роботи створеної бібліотеки. Зберегла отримані результати виконання для подальшого тестування, а також обґрунтувала методи оптимізації системи, такі як: кешування даних, розпаралелювання, оптимізація затінення, використання просторових фігур та суперсемпінг. Наостанок розробила певне удосконалення розробленого методу, щоб порівняти отримані результати під час тестування методів.

5 ТЕСТУВАННЯ ПРОГРАМНИХ ЗАСОБІВ

5.1 Покриття бібліотеки Unit-тестуванням для перевірки коректної роботи функцій опісля вдосконалення

В ході проходження практики, я отримала нові знання щодо написання чистого коду та тестування програмних засобів та вирішила застосувати набуті навички на прикладі написаної бібліотеки.

Додавши розпаралелювання процесів, також поставила лічильники для перевірки швидкості удосконаленого методу з початковим.

У результаті перевірки отримала зменшення часу рендерингу з 20 секунд до 4.8 при резолюції пікселя 4 та зі 120 до 20 при резолюції 25.

У аплікації в дужках до поля введення встановлена примітка з метою зменшення часу очікування при тестуванні. З використаними технічними характеристиками якість вихідного зображення не є помітна людському оку, але дозволяє зменшити час очікування.

5.2 Методи тестування

Для базового тестування системи побудови зображень 3D моделей з використанням технології відстеження зворотної траєкторії променя було обрано 3 початкових сцени. Всі тестування розробленої системи відбувались на комп’ютері за наступними технічними характеристиками:

- Процесор Intel(R) Core(TM) i5-6198DU CPU @ 2.30GHz 2.40 GHz;
- Оперативна пам’ять DDR 2, 8 gb;
- Відеокарта GeForce 960;

5.3 Використання результатів бібліотеки з тестовими існуючими прикладами

Щоб дослідити декілька існуючих прикладів, написаних з використанням

цієї бібліотеки, один з яких – це Поле Корнелла, більше тест, аніж демонстрація, спрямований на визначення точності візуалізації програмного забезпечення шляхом порівняння відображеної сцени з фактичною фотографією тієї ж сцени і який вважається найбільш вживаною 3D-тестовою моделлю, я знайшла декілька прикладів у відкритому доступі для незалежного тестування за допомогою тестової аплікації у форматі JSON [26]. Для отримання різниці між отриманими методами, додала базовий метод трасування променів без використання жодних проаналізованих удосконалень, щоб побачити справжній час візуалізації 3D-сцен. Для дослідження вирішила провести 4 тестові візуалізації для кожної з 4 сцен з урахуванням різної резолюції (або ж щільності) пікселя, щоб побачити, чи отримаємо залежність часу від точності вхідного файлу.

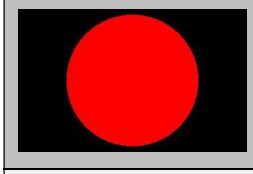
5.4 Результати тестування

Для тестування результатів роботи розробленої системи побудови зображення 3D моделей з відстеженням зворотної траєкторії променя були побудовані зображення з наступними властивостями:

- Базова растеризація зображення;
- Затінення;
- Білінійна інтерполяція (удосконалення методів існуючого рендерингу);
- Відображення променів;
- Навколишнє освітлення сцени;
- Відображення променя;
- Заломлення світла;

Отже, наступні приклади демонструють функціональну роботу системи:

Таблиця 5.1 – час рендерингу для різних методів оптимізації, отриманий з використанням таймеру при візуалізації 3D-сцен у тестовій аплікації

Резолюція пікселя: ***Тест 1: 1*** ***Тест 2: 4*** ***Тест 3: 16*** ***Тест 4: 25***	***Час роботи з використанням методу трасування променів (мс)***	***Час роботи з використанням методу трасування променів, під'єднанням просторових фігур(мс)***	***Час роботи з використанням методу трасування променів, під'єднанням просторових фігур, розпаралеленням та з оптимізацією затінення (мс)***
Базова сцена з однією сферою (0) 	Тест 1: 87 Тест 2: 104 Тест 3: 389 Тест 4: 526	Тест 1: 43 Тест 2: 80.7 Тест 3: 211 Тест 4: 267	Тест 1: 17 Тест 2: 29 Тест 3: 100.3 Тест 4: 153
Сцена 1	Тест 1: 10967 Тест 2: 12065 Тест 3: 58269 Тест 4: 183561	Тест 1: 4265 Тест 2: 6928 Тест 3: 17301 Тест 4: 90879	Тест 1: 763 Тест 2: 4801 Тест 3: 11448 Тест 4: 20444
Сцена 2	Тест 1: 7594 Тест 2: 12065 Тест 3: 49007 Тест 4: 92087	Тест 1: 2042 Тест 2: 5803 Тест 3: 22801 Тест 4: 92087	Тест 1: 561 Тест 2: 2007 Тест 3: 13684 Тест 4: 92087
Сцена 3	Тест 1: 6985 Тест 2: 19664 Тест 3: 84051 Тест 4:119034	Тест 1: 2863 Тест 2: 9094 Тест 3: 33754 Тест 4: 42806	Тест 1: 896 Тест 2: 6859 Тест 3: 10089 Тест 4: 18074

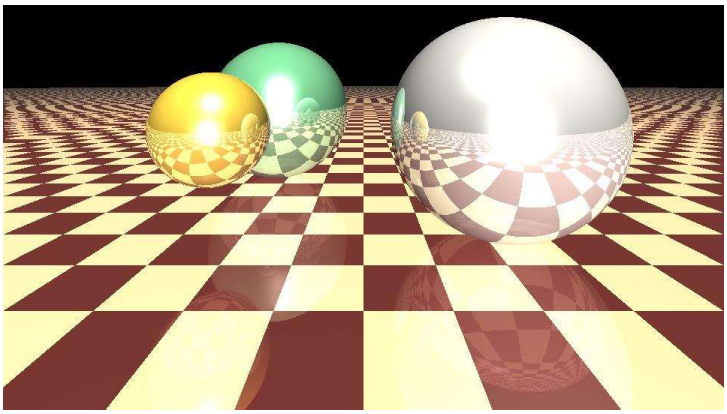


Рис. 5.2 – сцена 1 з застосуванням технології зворотного відстеження променя

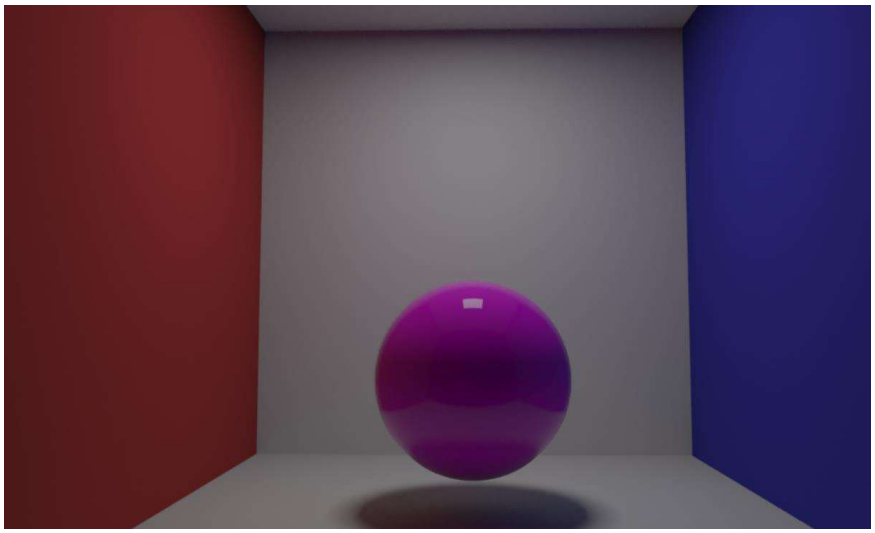


Рис. 5.3 – сцена 2 з застосуванням технології зворотного відстеження променя

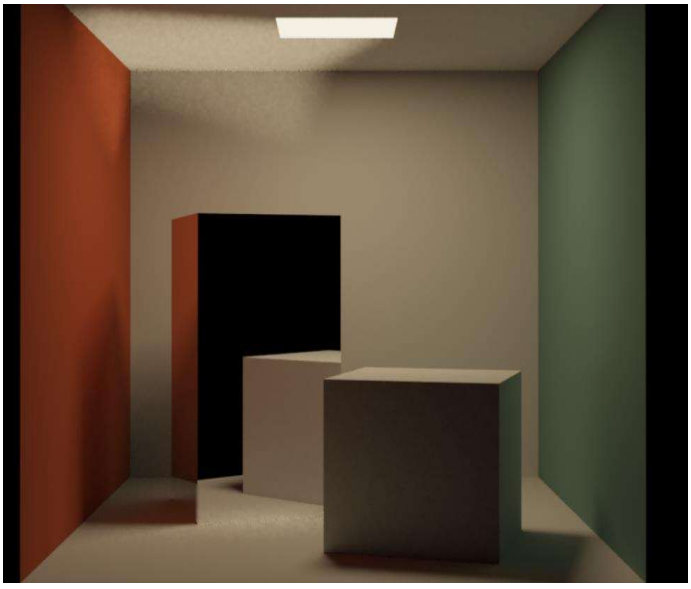


Рис. 5.4 - сцена 3 з застосуванням технології зворотного відстеження променя

У результаті виконання тестування можна побачити закономірності щодо використання удосконаленого методу трасування променів на противагу його базовій основі.

5.5 Висновки до розділу

В даному розділі було продемонстровано функціонування методів оптимізації та доведена їх коректність. Крім цього, візуально показана робото-

спроможність та фото-реалістичність системи побудови зображення 3D моделей з відстеженням зворотної траєкторії променя на багатьох прикладах. Також була протестована система на слабших комп'ютерах, і виявлено, що вона може не працювати коректо, у зв'язку з тим, що система потребує доволі потужну обчислювальну спроможність.

ВИСНОВКИ

В ході першого етапу досліджень я зробила висновок, що розрахунок освітлення є по своїй суті складною задачею. Навіть з урахуванням усіх спрощень та удосконалень, що допускаються у ряді методів моделювання освітлення, результуюча складність усе одно може бути завеликою і затрати часу на візуалізацію окремого кадру можуть перевищувати визначений максимальний ліміт.

Відповідно до цього було досліджено існуючі аналоги математичних моделей для побудови різних видів сцен та розроблено певні додаткові оптимізації щодо них для можливості зменшення використання часу на процес візуалізації, а також спроби використання більш складних та якісних методів моделювання освітлення, щоб, за рахунок цього, за загальним часом візуалізації залишитись у встановлених межах, не перевищуючи час виконання початкового варіанту алгоритму, та зі збереженням якості сцен.

В ході дослідження та опрацювання даної теми мною було створено програмний модуль з використанням алгоритму на основі методу трасування променів для збереження якості сцен з урахуванням заломлення променів світла відповідно до заданих матеріалів при зменшенні часу на обробку та формування даної сцени, а також ваги вихідного файлу та мови програмування Python завдяки гнучкості та простоті синтаксису даної мови.

Саме метод трасування променів увійшов в основу за рахунок можливості рендерингу гладеньких об’єктів без апроксимації їх полігональними поверхнями, відсічення невидимих поверхонь, перспективи і коректних змін поля зору, які є логічним наслідком алгоритму та низки переваг над іншими методами, які були розглянуті в ході досліджень, за рахунок яких було отримано зменшення часу на генерацію одного кадру результуючої сцени з мінімальними втратами по якості для користувача.

На даний момент основним призначенням даного модулю є його використання, як бібліотеки для використання різними програмами, які створені як системи віртуальної реальності, так і для професійних цілей, зокрема перетворювачів для зменшення розмірів вхідних файлів для подальшого опрацювання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Adobe Електронний ресурс: Файли PPM. – 2022 – 15 листопада 2022 – Режим доступу: https://www.adobe.com/ua/creativecloud/file-types/image/raster/ppm-file.html

2. Черняхівський В.В. Тема 5. Динамічні бібліотеки. Загальні принципи будови. / В.В. Черняхівський – ЛНУ ім. І. Франка, 2021 – 15 с.

3. NVIDIA Electronic source: Accelerating The Real-Time Ray Tracing Ecosystem: DXR For GeForce RTX and GeForce GTX / Andrew Burnes. – 2019 – [Cited March 18, 2019]. – Available from: https://www.nvidia.com/en-us/geforce/news/geforce-gtx-ray-tracing-coming-soon/

4. Ковальчук С. А., Комп'ютерне моделювання сцен з урахуванням заломлення променів світла / С. А. Ковальчук // Міжнародна Студентська Наукова Конференція з Прикладної Математики та Комп'ютерних Наук AMICon : Тези доповідей 2022 року – Львів, 2022 р. – С. 132-134

5. Інформатика – баз: 6.10. Колірна модель [Електронний ресурс] – 2013 – [Цит. 23 травня 2013] – Режим доступу: http://www.zhu.edu.ua/mk_school/mod/page/view.php

6. Волошина М., Полігональне моделювання: значення, особливості, рекомендації в роботі [Електронний ресурс] / М. Волошина – 2016 – [Цит. 22 лютого 2016] – Режим доступу: https://klona.ua/uk/blog/3d-modeling-and-visualization-uk/poligonalne-modelyuvannya-znachennya-o

7. Archicgi Electronic source: 3D Modeling: 4 Types Used in Architectural Projects / Ana Wayne – 2022 – [Cited February 1, 2022]. – Available from: https://archicgi.com/architecture/3d-modeling-types-in-architecture/#:%7E:text=The%20first%203D%20models%20were,Sutherland%2C%20the%20creator%20of%20Sketchpad

8. Тема 16. Моделювання. Основні поняття. Види моделей, їх класифікація. Вимоги до моделей [Електронний ресурс] – Харківський національний економічний університет імені Семена Кузнеця, Відділ електронних

засобів навчання, 2016 – Режим доступу: https://pns.hneu.edu.ua/pluginfile.php/293321/mod_resource/content/2/%D0%A2%D0%B5%D0%BC%D0%B0%2016.pdf

9. Concept Art Empire Electronic source: What is 3D Modeling & What's It Used For? / Josh Petty – 2018 – [Cited April 27, 2018] – Available from: https://conceptartempire.com/what-is-3d-modeling/

10. ArtStation Electronic source: This is normal 1: What normal maps are and how they work. / Carlos Lemos – 2019 – [Cited June 16, 2019] – Available from: https://typhen.artstation.com/blog/GMyG/this-is-normal-1-what-normal-maps-are-and-how-they-work

11. Colin Smith, On Vertex-Vertex Systems and Their Use in Geometric and Biological Modelling. Дис канд. філософ. Наук – April, 2006 / The university of Calgary, Alberta – 2006 – С. 13-18

12. Pinterest допис : Приклад оптимізації поліноміальної моделі [Електронний ресурс] / Kenny – 2016 – Режим доступу: https://www.pinterest.com/pin/395753885989194401/

13. Schöning J., Heidemann G. Interactive 3D modeling / Julius Schoning and Gunther Heidemann. – Osnabruck, Germany: Institute of Cognitive Science, University of Osnabruck, 2015 – P. 6

14. Adobe Електронний ресурс: Розширене кадрування, зміна розміру, зміна роздільної здатності – 2022 – 15 листопада 2022 – Режим доступу: https://helpx.adobe.com/ua/photoshop/kb/advanced-cropping-resizing-resampling-photoshop.html

15. Buss, S.R. V.2 Bump mapping // 3D Computer Graphics: A Mathematical Introduction with OpenGL. — Cambridge University Press, 2003. — С. 371 — ISBN 9780521821032

16. Brawley, Z., Parallax Occlusion Mapping: Self-Shadowing, Perspective-Correct Bump Mapping Using Reverse Height Map Tracing. In ShaderX3: Advanced

Rendering with DirectX and OpenGL // Brawley, Z., Tatarchuk, N., Engel, W., Ed., Charles River Media, – 2004 – C. 135–154.

17. Foley J.D., van Dam A. Feiner S.K., Hughes J.F. Computer Graphics: Principles and Practice // Addison-Wesley Publishing Company. – 1996. – P. 718–739

18. Зацерковний В.І., Алгоритмізація та програмування. Конспект лекцій для студентів спеціальності «Економічна кібернетика» // Зацерковний В.І., Гур'єв В.І., Сімакін Ю.С., Фірсова І.В – Чернігівський державний інститут економіки та управління, 2012 – 184 с.

19. Wylie. C., Halftone Perspective Drawings by Computer," // Wylie. C., Romney. G. W. Evans., Erdahl A. – Proc. AFIPS FJCC 1967 – Vol. 31, 49

20. Scott D, Roth "Ray Casting for Modeling Solids", Computer Graphics and Image Processing – February, 1982 – с. 109–144

21. J. Avro, A survey of ray tracing acceleration techniques. In An Introduction to Ray Tracing // J. Avro, D. Kirk. – A. Glassner, Ed., – Academic Press, San Diego, CA, 1989 – с. 201–262.

22. Артур Аппель. «Деякі методики відтінку твердих тіл» // Аппель А., – 1969.

23. Scratchapixel Electronic source: An Overview of the Ray-Tracing Rendering Technique // Scratchapixel Authors – 2022 – [Cited August 31, 2022] – Available from: https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-overview

24. Мізюк О., Путівник мовою програмування Python [Електронний ресурс] // О. Мізюк – 2020 – [Цит. 3 грудня 2020] – Режим доступу: https://pythonguide.rozh2sch.org.ua/

25. Юніс Мохаммад. Залучення блочної міжпіксельної інтерполяціх для прискорення алгоритму трасування променів / Мохаммад Юніс, Р.В. Мальчева, С.О. Ковалев // Машинобудування та техносфера XXI століття. / Збірник XVIII міжнародної научної конференції. - Донецьк: ДонНТУ, 2011. - Т.4. - С.189-191

26. Бібліотека 3D-сцен sketchfab. Поле Корнелла [Електронний ресурс] – 2017 – [Режим доступу]: https://sketchfab.com/3d-models/cornell-box-original-0d18de8d108c4c9cab1a4405698cc6b6

27. Кузьменко А., Що таке трасування променів [Електронний ресурс] / А. Кузьменко – 2019 – [Цит. 09 липня 2019] – Режим доступу: http://surl.li/dzdao

Додаток А. Вихідний код DLL

```
##########################################################################
##########################################################################
##########   Розкодування бібліотеки для рендеру зображень      ##########
##########              в режимі реального часу                  ##########
##########                                                       ##########
##########       Виконала: Ковальчук Софія, студентка            ##########
##########           кафедри Інформаційних систем                ##########
##########   факультету Прикладної математики та інформатики     ##########
##########       Львівського національного університету          ##########
##########                 імені Івана Франка                    ##########
##########                     Львів, 2022                       ##########
##########################################################################
##########################################################################

import math
import shutil
import tempfile
from multiprocessing import Process, Value
from pathlib import Path


class Vector:
    """Означення вектора"""

    def __init__(self, x=0.0, y=0.0, z=0.0):
        self.x = x
        self.y = y
        self.z = z

    def __str__(self):
        return "({}, {}, {})".format(self.x, self.y, self.z)

    def dot_product(self, other):
        return self.x * other.x + self.y * other.y + self.z * other.z

    def magnitude(self):
        return math.sqrt(self.dot_product(self))

    def normalize(self):
        return self / self.magnitude()

    def __add__(self, other):
        return Vector(self.x + other.x, self.y + other.y, self.z + other.z)

    def __sub__(self, other):
        return Vector(self.x - other.x, self.y - other.y, self.z - other.z)

    def __mul__(self, other):
        assert not isinstance(other, Vector)
        return Vector(self.x * other, self.y * other, self.z * other)

    def __rmul__(self, other):
        return self.__mul__(other)

    def __truediv__(self, other):
        assert not isinstance(other, Vector)
        return Vector(self.x / other, self.y / other, self.z / other)


class Image:
    """ Означення зображення на противагу використання Pillow """
```

```
    def __init__(self, width, height):
        self.width = width
        self.height = height
        self.pixels = [[None for _ in range(width)] for _ in range(height)]

    def set_pixel(self, x, y, col):
        self.pixels[y][x] = col

    def write_ppm(self, img_fileobj):
        Image.write_ppm_header(img_fileobj, height=self.height, width=self.width)
        self.write_ppm_raw(img_fileobj)

    @staticmethod
    def write_ppm_header(img_fileobj, height=None, width=None):
        """Ініціалізація заголовку PPM файлу"""
        img_fileobj.write("P3 {} {}\n255\n".format(width, height))

    def write_ppm_raw(self, img_fileobj):
        """
        PPM - скорочено від portable pixel map.
        Це розширення файлу, в якому визначаємо P3 як формат зображення,
прописуємо кількість стовпців та рядків нашої колірної матриці.
        В наступному рядку задаємо максимальне значення кольору - 255 для RGB
(колірної системи, під яку прописуватимемо матрицю).
        В третьому рядку додаємо колірне значення кожного пікселя у форматі (R G
B)
        Наприклад:
        P3 3 2
        255
        255  0  0     0 255   0    0   0 255
        255 255 0   255 255 255    0   0   0
        У прикладі отримала палітру з шести кольорів, які відображаю у 3 стовпцях
та 2 рядках:
        червоний зелений синій
        жовтий   білий   чорний
        """

        def to_byte(c):
            return round(max(min(c * 255, 255), 0))

        for row in self.pixels:
            for color in row:
                img_fileobj.write(
                    "{} {} {} ".format(
                        to_byte(color.x), to_byte(color.y), to_byte(color.z)
                    )
                )
            img_fileobj.write("\n")


class Color(Vector):
    """Тут колір тривимірної моделі передаватимемо, як значення вектора у форматі
RGB.
    Сутність вектора."""

    @classmethod
    def from_hex(cls, hexcolor="#000000"):
        x = int(hexcolor[1:3], 16) / 255.0
        y = int(hexcolor[3:5], 16) / 255.0
        z = int(hexcolor[5:7], 16) / 255.0
        return cls(x, y, z)
```

```
class Point(Vector):
    """Точка відображається у тривимірній множині координат. Сутність вектора"""
    pass


class Ray:
    """Модель променю у тривимірному декартовому просторі.
       Промінь - це шлях з оригінальним та нормалізованим напрямом (промінь від
глядача до предмету) """

    def __init__(self, origin, direction):
        self.origin = origin
        self.direction = direction.normalize()


class Intersection:
    """ Означення точки або прямої перетину променя крізь об'єкт """
    def __init__(self, point, distance, normal, obj):
        self.point = point
        self.distance = distance
        self.normal = normal
        self.obj = obj


class Scene:
    """Сцена міститиме інформацію, яка необхідна для опрацювання методу трасування
променів"""

    def __init__(self, camera, objects, lights, width, height):
        # Модель променевого емітера - камера
        self.camera = camera
        self.objects = objects
        self.lights = lights
        self.width = width
        self.height = height


class Triangle:
    """Полігон-трикутник - найпростіше відображення низькополігонального
зображення"""

    def __init__(self, a=Vector(), b=Vector(), c=Vector()):
        self.a = a
        self.b = b
        self.c = c

    @classmethod
    def calculate_area(cls):
        s = (cls.a + cls.b + cls.c) / 2
        return float((s * (s - cls.a) * (s - cls.b) * (s - cls.c))) ** 0.5

    def get_area(self):
        return self.area


class Sphere:
    """Сфера містить необхідну інформацію про центр об'єкту, радіус та матеріал, з
якого виготовлена"""

    def __init__(self, center, radius, material):
        self.center = center
        self.radius = radius
        self.material = material
```

```
    def intersects(self, ray):
        """ Тут відбувається перевірка перетину сфери променем.
        Якщо немає перетину - повертаємо None, інакше - відстань перетину
        (дистанцію або точну в залежності від випадку) """
        sphere_to_ray = ray.origin - self.center
        # a = 1
        b = 2 * ray.direction.dot_product(sphere_to_ray)
        c = sphere_to_ray.dot_product(sphere_to_ray) - self.radius * self.radius
        discriminant = b * b - 4 * c

        if discriminant >= 0:
            dist = (-b - math.sqrt(discriminant)) / 2
            if dist > 0:
                return dist
        return None

    def normal(self, surface_point):
        """Тут повертаємо нормаль поверхні до точки на поверхні сфери"""
        return (surface_point - self.center).normalize()


class Material:
    """Матеріал надає інформацію про те, як об'єкт реагуватиме на світло"""

    def __init__(self, color=Color.from_hex("#FFFFFF"), ambient=0.05, diffuse=1.0,
specular=1.0, reflection=0.5):
        self.color = color  # інформація про базовий колір
        self.ambient = ambient  # інформація про наявність відбиття від
навколишнього освітлення, якщо немає джерела
        self.diffuse = diffuse  # інформація про відбиття розсіяного джерела
світла
        self.specular = specular  # інформація про наявне дзеркальне підсвічування
        self.reflection = reflection

    def color_at(self, position):
        return self.color


class ChequeredMaterial:
    """Матеріал для прикладу шахової дошки - мітсить інформацію про два базові
кольори поділу"""

    def __init__(
            self,
            color1=Color.from_hex("#FFFFFF"),
            color2=Color.from_hex("#000000"),
            ambient=0.05,
            diffuse=1.0,
            specular=1.0,
            reflection=0.5,
    ):
        self.color1 = color1
        self.color2 = color2
        self.ambient = ambient
        self.diffuse = diffuse
        self.specular = specular
        self.reflection = reflection

    def color_at(self, position):
        if int((position.x + 5.0) * 3.0) % 2 == int(position.z * 3.0) % 2:
            return self.color1
        else:
```

```
            return self.color2


class Cylinder(object):
    """Циліндр містить необхідну інформацію про висоту, радіус та матеріал, з
якого виготовлений"""
    def __init__(self, startpoint=Vector(), endpoint=Vector(), radius=0.1,
material=Material()):
        self.startpoint = startpoint
        self.endpoint = endpoint
        self.radius = radius
        self.material = material

    def intersection(self, l):
        q = l.d.dot(l.o - self.c) ** 2 - (l.o - self.c).dot(l.o - self.c) +
self.radius ** 2
        if q < 0:
            return Intersection(Vector(0, 0, 0), -1, Vector(0, 0, 0), self)
        else:
            d = -l.d.dot(l.o - self.c)
            d1 = d - math.sqrt(q)
            d2 = d + math.sqrt(q)
            if 0 < d1 and (d1 < d2 or d2 < 0):
                return Intersection(l.o + l.d * d1, d1, self.normal(l.o + l.d *
d1), self)
            elif 0 < d2 and (d2 < d1 or d1 < 0):
                return Intersection(l.o + l.d * d2, d2, self.normal(l.o + l.d *
d2), self)
            else:
                return Intersection(Vector(0, 0, 0), -1, Vector(0, 0, 0), self)

    def normal(self, b):
        return (b - self.c).normal()


class Plane:
    """Площина містить необхідну інформацію про напрям, центр об'єкту та матеріал,
з якого виготовлена"""
    def __init__(self, point: object = Vector(), normal: object = Vector(),
material: object = Material()) -> object:
        self.normal = normal
        self.point = point
        self.material = material

    def intersection(self, l):
        d = l.d.dot(self.normal)
        if d == 0:
            return Intersection(Vector(0, 0, 0), -1, Vector(0, 0, 0), self)
        else:
            d = (self.point - l.o).dot(self.normal) / d
            return Intersection(l.o + l.d * d, d, self.normal, self)


class Rectangle(Plane):
    """Нащадок площини, в якого можемо вирахувати площу"""

    def __init__(self, point, normal, material):
        Plane.__init__(self, point, normal, material)

    def intersection(self, ray):
        destination = ray.dest.dot(self.normal)
        if destination == 0:
            return Intersection(Vector(0, 0, 0), -1, Vector(0, 0, 0), self)
```

```
        else:
            destination = (self.point - ray.orig).dot(self.normal) / destination
            return Intersection(ray.orig + ray.desti * destination, destination,
self.normal, self)


class Light:
    """Світло надає перетворення кольору при взаємодії з об'єктом"""

    def __init__(self, position, color=Color.from_hex("#FFFFFF")):
        self.position = position  # інформація про позицію джерела світла
        self.color = color  # інформація про колір освітлення (тепле, холодне
тощо)


class RenderEngine:
    """Рендер 3D сцени в 2D зображнення з використанням методу трасування
променів"""

    MAX_DEPTH = 5
    MIN_DISPLACE = 0.0001

    def render_multiprocess(self, scene, process_count, img_fileobj):
        def split_range(count, parts):
            d, r = divmod(count, parts)
            return [
                (i * d + min(i, r), (i + 1) * d + min(i + 1, r)) for i in
range(parts)
            ]

        width = scene.width
        height = scene.height
        ranges = split_range(height, process_count)
        temp_dir = Path(tempfile.mkdtemp())
        temp_file_tmpl = "kovalchuk-part-{}.temp"
        processes = []
        try:
            rows_done = Value("i", 0)
            for hmin, hmax in ranges:
                part_file = temp_dir / temp_file_tmpl.format(hmin)
                processes.append(
                    Process(
                        target=self.render,
                        args=(scene, hmin, hmax, part_file, rows_done),
                    )
                )
            # Стартуємо всі процеси
            for process in processes:
                process.start()
            # Чекаємо на закінчення усіх процесів
            for process in processes:
                process.join()
            # Будуємо зображення на основі додання отриманих частин
            Image.write_ppm_header(img_fileobj, height=height, width=width)
            for hmin, _ in ranges:
                part_file = temp_dir / temp_file_tmpl.format(hmin)
                img_fileobj.write(open(part_file, "r").read())
        finally:
            shutil.rmtree(temp_dir)

    def render(self, scene, hmin, hmax, part_file, rows_done):
        width = scene.width
        height = scene.height
```

```
        aspect_ratio = float(width) / height
        x0 = -1.0
        x1 = +1.0
        xstep = (x1 - x0) / (width - 1)
        y0 = -1.0 / aspect_ratio
        y1 = +1.0 / aspect_ratio
        ystep = (y1 - y0) / (height - 1)

        camera = scene.camera
        pixels = Image(width, hmax - hmin)

        for j in range(hmin, hmax):
            y = y0 + j * ystep
            for i in range(width):
                x = x0 + i * xstep
                ray = Ray(camera, Point(x, y) - camera)
                pixels.set_pixel(i, j - hmin, self.ray_trace(ray, scene))
            # Update progress bar
            if rows_done:
                with rows_done.get_lock():
                    rows_done.value += 1
                    # відсоткове відображення процесу рендеру
                    print(
                        "{:3.0f}%".format(float(rows_done.value) / float(height) *
100),
                        end="\r",
                    )
        with open(part_file, "w") as part_fileobj:
            pixels.write_ppm_raw(part_fileobj)

    def ray_trace(self, ray, scene, depth=0):
        color = Color(0, 0, 0)
        # шукаємо найближчий об'єкт, якого досягає промінь в даній сцені
        dist_hit, obj_hit = self.find_nearest(ray, scene)
        if obj_hit is None:
            return color
        hit_pos = ray.origin + ray.direction * dist_hit
        hit_normal = obj_hit.normal(hit_pos)
        color += self.color_at(obj_hit, hit_pos, hit_normal, scene)
        if depth < self.MAX_DEPTH:
            new_ray_pos = hit_pos + hit_normal * self.MIN_DISPLACE
            new_ray_dir = (
                    ray.direction - 2 * ray.direction.dot_product(hit_normal) *
hit_normal
            )
            new_ray = Ray(new_ray_pos, new_ray_dir)
            # Послабляємо відбитий промінь на коефіцієнт відбиття
            color += (
                    self.ray_trace(new_ray, scene, depth + 1) *
obj_hit.material.reflection
            )
        return color

    def find_nearest(self, ray, scene):
        dist_min = None
        obj_hit = None
        for obj in scene.objects:
            dist = obj.intersects(ray)
            if dist is not None and (obj_hit is None or dist < dist_min):
                dist_min = dist
                obj_hit = obj
        return (dist_min, obj_hit)
```

```
    def color_at(self, obj_hit, hit_pos, normal, scene):
        material = obj_hit.material
        obj_color = material.color_at(hit_pos)
        to_cam = scene.camera - hit_pos
        specular_k = 50
        color = material.ambient * Color.from_hex("#FFFFFF")
        # Обрахунок світла
        for light in scene.lights:
            to_light = Ray(hit_pos, light.position - hit_pos)
            # інформація про відбиття розсіяного джерела світла та тінь (за
Ламбертом)
            color += (
                    obj_color
                    * material.diffuse
                    * max(normal.dot_product(to_light.direction), 0)
            )
            # інформація про наявне дзеркальне підсвічування ( за Blinn-Phong)
            half_vector = (to_light.direction + to_cam).normalize()
            color += (
                    light.color
                    * material.specular
                    * max(normal.dot_product(half_vector), 0) ** specular_k
            )
        return color
```