Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Факультет інформатики

Кафедра інформатики

3-D ВІЗУАЛІЗАЦІЯ ОБ'ЄКТІВ НА ОСНОВІ МЕДИЧНИХ ЗОБРАЖЕНЬ

Текстова частина до курсової роботи

за спеціальністю 121 «Інженерія програмного забезпечення»

Керівник курсової роботи

Бучко О. А.
(*прізвище та ініціали*)

_____

(*підпис*)

"____"_____ 2022 р.

Виконав студент _____

Янченко Б. І.

(*прізвище та ініціали*)

_____

(*підпис*)

"____" _____ 2022 р.

Київ-2022

Тема: Застосування технології трасування променів для 3D зображень

Календарний план виконання роботи:

№	Назва етапу курсової роботи	Термін виконання	Примітка
1.	Отримання завдання на курсову роботу	вересень	
2.	Огляд літератури за темою роботи	жовтень-грудень	
3.	Створення практичної частини роботи	січень-квітень	
4.	Написання текстової частини роботи	березень-квітень	
6.	Надання роботи керівнику для перевірки	кінець квітня	
8.	Коригування роботи за результатами перевірки	кінець квітня-початок травня	
9	Подання роботи на кафедру для перевірки на плагіат	тиждень до захисту	
10.	Захист курсової роботи	кінець травня	

Студент Янченко Б. І.

Керівник Бучко О. А.

“____” _____

Зміст

Вступ 4
Теорія трасування променів..... 6
Основи 8
Камера 9
Трасування.....10
Відображення.....12
Висновок 14
Використані джерела: 15

Вступ

Застосування технології трасування променів для 3D зображень є однією з ключових областей комп'ютерної графіки, яка має значний вплив на візуалізацію реалістичних сцен. Ця технологія надає можливість створювати зображення з вражаючою деталізацією, реалістичним освітленням та тінями, що сприяє покращенню якості візуалізації у різних сферах, включаючи відеоігри, кіноіндустрію, архітектурне моделювання та медичну діагностику.

Мета цієї курсової роботи полягає в розгляді та дослідженні технології трасування променів для 3D зображень з урахуванням її застосування в різних контекстах. Використовуючи цю технологію, можна досягти високої якості реалістичного відображення об'єктів у тривимірному просторі, враховуючи їх матеріали, освітлення та взаємодію з навколишнім середовищем.

У цьому дослідженні будуть розглянуті основні алгоритми та методи трасування променів, такі як відстеження променів, взаємодія променів з поверхнями та обчислення освітлення. Крім того, будуть розглянуті сучасні розширення технології, включаючи трасування променів в реальному часі та використання апаратного прискорення для поліпшення продуктивності.

Окрім того, ця робота вивчить практичні застосування трасування променів для створення реалістичних 3D зображень. Будуть розглянуті приклади використання в індустріальному дизайні, відеоіграх, віртуальній реальності та архітектурному моделюванні. Дослідження реальних випадків застосування технології трасування променів дозволить нам зрозуміти, як ця технологія впливає на поліпшення якості зображень та взаємодії з віртуальними об'єктами.

Крім того, розглянемо потенційні виклики та обмеження, пов'язані з застосуванням трасування променів для 3D зображень. Одним з основних факторів є обчислювальна складність цього підходу, яка вимагає потужних обчислювальних ресурсів для швидкої та ефективної роботи. Також важливим є розгляд питань оптимізації алгоритмів трасування променів для досягнення оптимальної продуктивності.

Завершуючи вступ, можна сказати, що розуміння технології трасування променів для 3D зображень та її застосування в різних сферах візуалізації важливо для розвитку комп'ютерної графіки та створення більш реалістичних імітаційних середовищ. У цій курсовій роботі ми детально розглянемо цю технологію, її алгоритми та практичні застосування, щоб виявити її потенціал та межі.

Запропонована робота сприятиме поглибленню знань у галузі трасування променів для 3D зображень та допоможе розширити наше розуміння сучасних тенденцій у візуалізації тривимірних сцен.

Теорія трасування променів

Трасування променів (англ. ray tracing) є одним з основних алгоритмів рендерингу комп'ютерних зображень. В основі цього методу лежить моделювання поведінки світла, що падає на об'єкти сцени і відбивається від них.

Принцип трасування променів полягає у відстеженні шляху променів світла від джерела світла до камери або ока спостерігача. Алгоритм починається зі створення променя, який починається в точці камери і проникає через кожен піксель зображення. Цей промінь, називається промінь від камери, направлений уздовж видимої лінії зору.

Коли промінь від камери перетинає сцену, він перевіряється на перетин з об'єктами сцени. Це зазвичай виконується шляхом перетину променя з обмежувальною оболонкою (bounding volume) кожного об'єкта в сцені. Якщо промінь перетинає об'єкт, то він обчислює точку перетину та властивості світла, які впливають на цю точку, наприклад, кольор, відбиття та прозорість. В залежності від властивостей світла та матеріалу об'єкта, можуть бути враховані такі ефекти, як тіні, відбиття, прозорість, розсіяне освітлення та інші.

У деяких випадках промінь від камери може стикнутися з іншими об'єктами, відбиватися або проникати крізь них. В такому випадку відбувається відбиття або проникнення нового променя, який продовжує трасуватися до наступного перетину або до досягнення максимальної глибини трасування.

Процес трасування променів повторюється для кожного пікселя зображення, щоб визначити його кольор та яскравість. Коли всі промені прослідковані та обчислені, формується остаточне зображення сцени, в якому враховані всі світлові ефекти, такі як тіні, відбиття та прозорість.

Одним з найбільших переваг трасування променів є його здатність до досить реалістичного моделювання світла та тіней. Трасування променів дозволяє враховувати взаємодію світла з поверхнями об'єктів, що дає більш точні та реалістичні зображення порівняно з іншими методами рендерингу, такими як растровий рендеринг або векторний рендеринг.

Однак трасування променів є обчислювально витратним процесом, оскільки вимагає обчислення великої кількості променів та їх перетинів з об'єктами сцени. Це призводить до значного збільшення обчислювальних витрат, особливо при трасуванні складних сцен з багатьма об'єктами та джерелами світла.

З технологічним прогресом та використанням апаратного прискорення, такого як графічні процесори (GPU) та прискорювачі апаратного трасування променів, трасування променів стає більш доступним та широко використовується в комп'ютерній графіці, візуалізації та фільмовому виробництві.

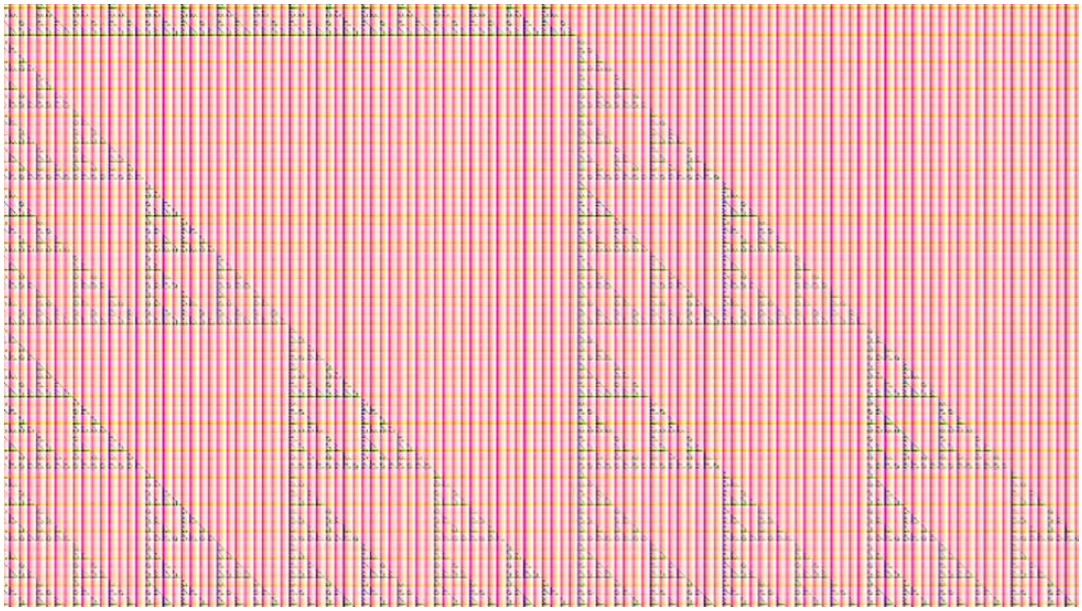
Основи

Функція OnRenderImage у Unity викликається автоматично після завершення рендерингу камери. Щоб здійснити рендеринг, ми спочатку створюємо цільовий рендер (render target) з необхідними розмірами та повідомляємо про це compute shader. У даному випадку використовується індекс 0 для функції ядра compute shader, оскільки ми маємо лише одне ядро.

Потім вказуємо шейдер, що запитуємо GPU про виконання груп потоків, які виконуватимуть код нашого шейдера. Кожна група потоків складається з певної кількості потоків, яку ми задаємо у самому шейдері. Розмір і кількість груп потоків можуть бути вказані у трьох вимірах, що дозволяє легко використовувати compute shaders для завдань будь-яких розмірів. У даному випадку створюємо групу потоків на кожні 8×8 пікселів цільового рендеру, що відповідає стандартному розміру групи потоків [numthreads(8,8,1)].

На завершення, використовуємо Graphics.Blit для запису результату на екран. Це дозволяє відобразити результат рендерингу.

Для перевірки програми, додаємо компонент RayTracingMaster до камери в сцені, призначаємо compute shader і запускаємо режим гри. Бачимо результуючий візуальний ефект, який відображається у вигляді красивого трикутного фракталу.



Камера

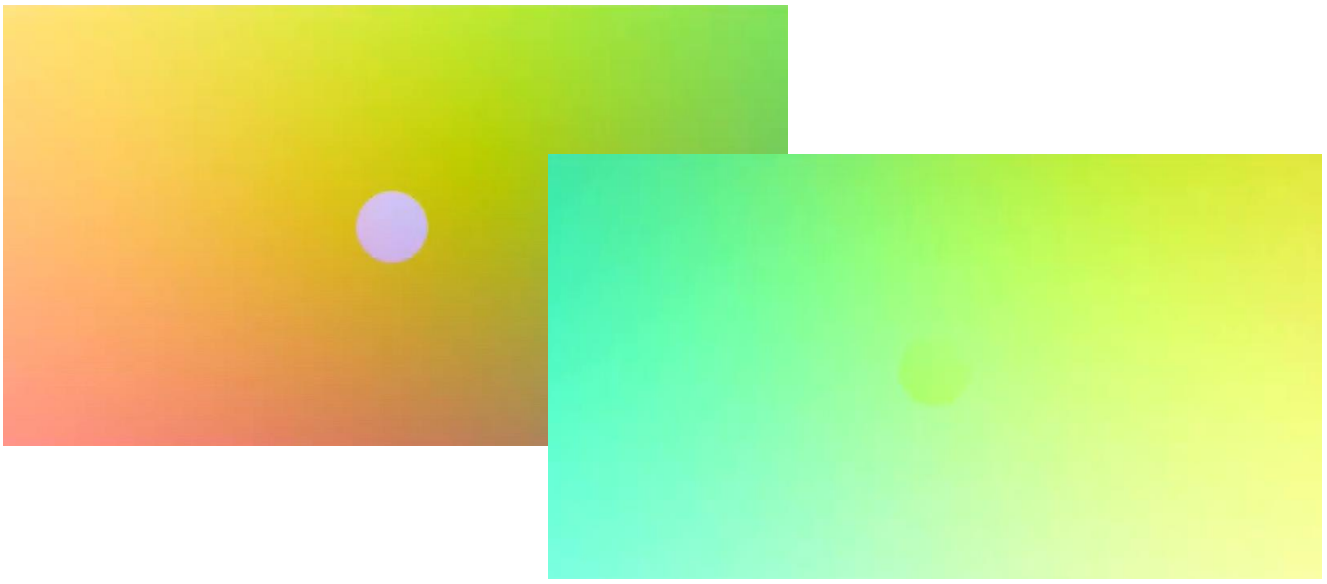
Тепер, коли відображені зображення на екрані, генеруємо промені від камери. Використовуючи вбудовану камеру Unity, ми можемо отримати необхідні матриці для цього. Давайте розглянемо, як ми будемо працювати з матрицями у шейдері.

```
private Camera _camera;

private void Awake()
{
    _camera = GetComponent<Camera>();
}
```

```
private void SetShaderParameters()
{
    _raytracingShader.SetMatrix("_CameraToWorld", _camera.cameraToWorldMatrix);
    _raytracingShader.SetMatrix("_CameraInverseProjection", _camera.projectionMatrix.inverse);

}
```

У шейдері ми визначаємо матриці, структуру Ray та функцію для конструювання. Потрібно врахувати, що HLSL, на відміну C#, оголошення функції чи змінної має виконуватися до використання. Для центру кожного екранного пікселя ми обчислюємо джерело та напрямок променя, і виводимо останнє як колір.

Спробуємо покрутити камерою, і бачимо, як змінюється небо:



Трасування

Математично ми можемо обчислити перетин між променем і геометрією сцени, і зберегти параметри зіткнення (позицію, нормаль і відстань вздовж променя). Якщо наш промінь стикається з кількома об'єктами, ми оберемо найближчий. Давайте визначимо в шейдері struct RayHit:

```
struct RayHit
{
    float3 position;
    float distance;
    float3 normal;
};

RayHit CreateRayHit()
{
    RayHit hit;
    hit.position = float3(0.0f, 0.0f, 0.0f);
    hit.distance = 1.#INF;
    hit.normal = float3(0.0f, 0.0f, 0.0f);
    return hit;
}
```

Завдання обчислення перетину прямої з нескінченною площиною при y=0 є простим. Однак, ми розглядаємо тільки зіткнення, які відбуваються в позитивному напрямку променя і відкидаємо всі зіткнення, які відбуваються на відстані, меншій, ніж потенційне попереднє зіткнення.

У мові програмування HLSL параметри передаються за значенням за замовчуванням, а не за посиланням. Тому ми працюємо лише з копією цих параметрів і не можемо передавати зміни функції, яка їх викликає. Ми передаємо структуру RayHit як параметр bestHit з кваліфікатором inout, щоб мати можливість змінювати вміст цієї структури під час виконання функції.

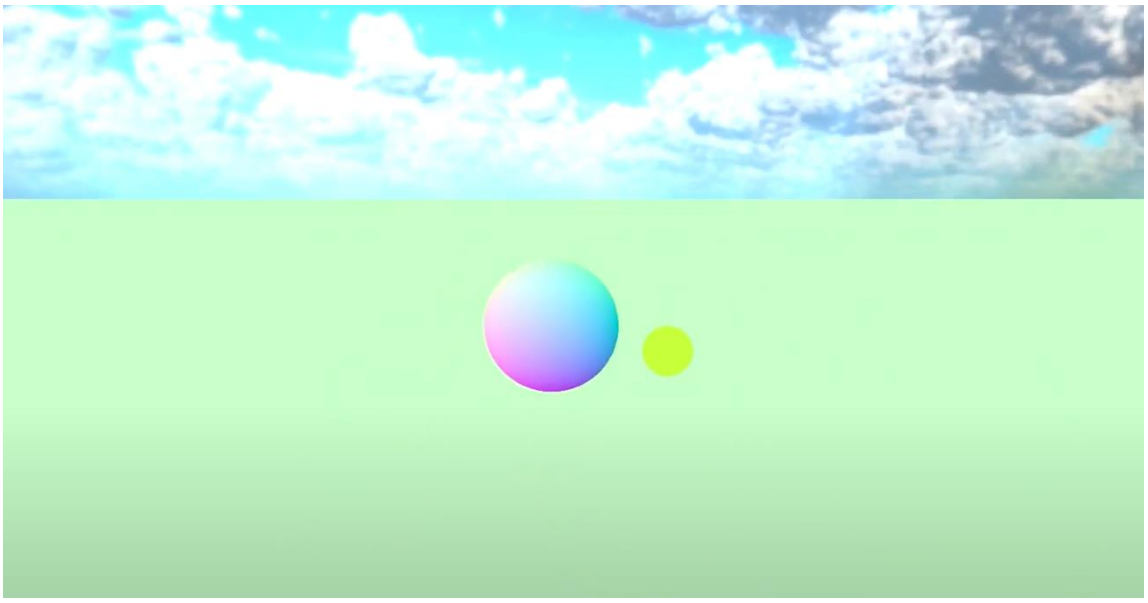
Додаємо функцію Trace

```
RayHit Trace(Ray ray)
{
    RayHit bestHit = CreateRayHit();
    IntersectGroundPlane(ray, bestHit);
    return bestHit;
}
```

Додамо сферу, та пропишемо для неї правила перетинання

```
void IntersectSphere(Ray ray, inout RayHit bestHit, float4 sphere)
{
    float3 d = ray.origin - sphere.xyz;
    float p1 = -dot(ray.direction, d);
    float p2sqr = p1 * p1 - dot(d, d) + sphere.w * sphere.w;
    if (p2sqr < 0)
        return;
    float p2 = sqrt(p2sqr);
    float t = p1 - p2 > 0 ? p1 - p2 : p1 + p2;
    if (t > 0 && t < bestHit.distance)
    {
        bestHit.distance = t;
        bestHit.position = ray.origin + t * ray.direction;
        bestHit.normal = normalize(bestHit.position - sphere.xyz);
    }
}
```

Ось, що маємо в результаті



Також, я завантажив з AssetStore Skybox Texture, яку буду потім використовувати

Відображення

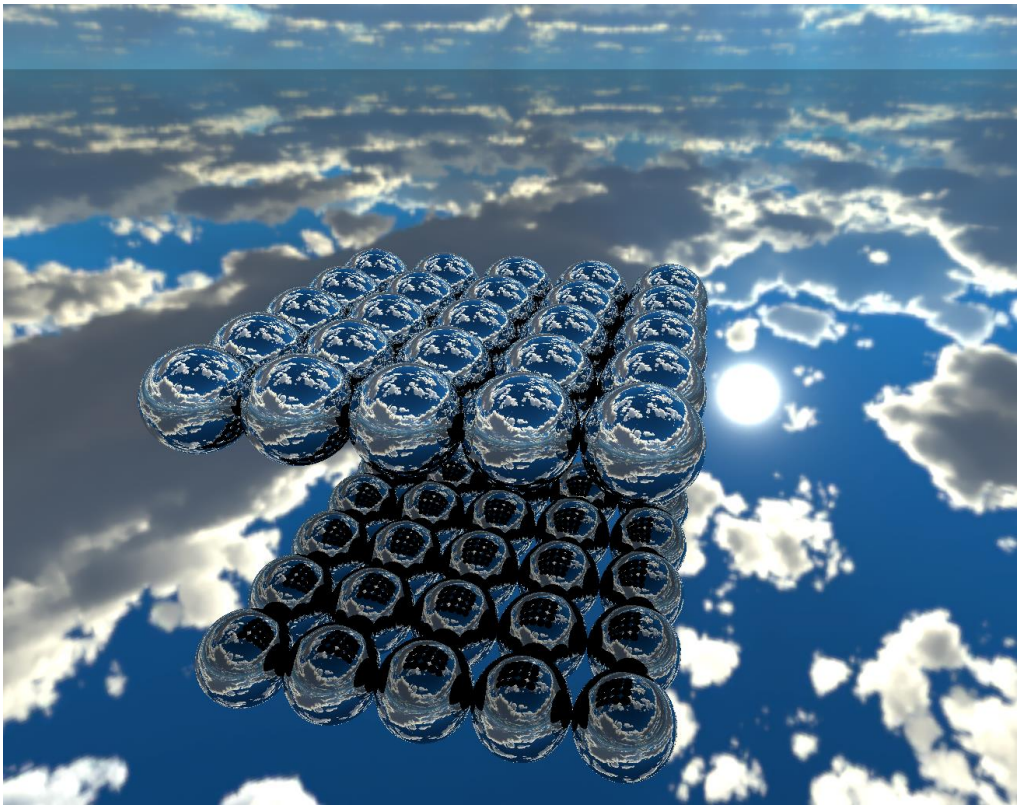
Фундамент для нашого трасувальника променів вже готовий, тому ми можемо розпочати складні речі, які насправді виділяють трасування променів на тлі інших технік рендерингу. Першими у цьому списку йдуть ідеальні відображення. Ідея проста: коли ми стикаємося з поверхнею, ми відображаємо промінь відповідно до закону відображення, який ви, можливо, пам'ятаєте зі школи (кут падіння = кут відображення), знижуємо його енергію і повторюємо процес, поки промінь або не зіткнеться з небом, або у нього не закінчиться енергія після заданої кількості відбитків.

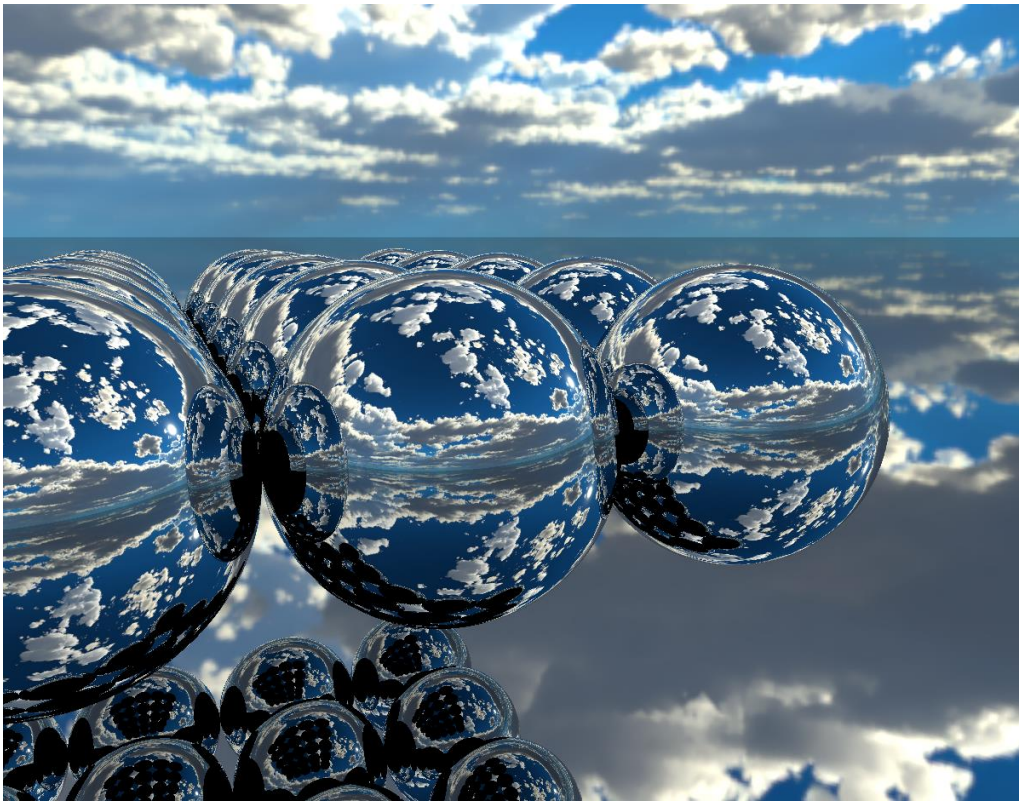
У шейдері додамо до променя змінну float3 energy та ініціалізуємо їх у функції CreateRay як ray.energy = float3(1.0f, 1.0f, 1.0f). Спочатку промінь матиме максимальні значення у всіх колірних каналах, які при кожному відображенні знижуватимуться.

Наша функція Shade тепер також виконує оновлення енергії та генерування відбитого променя, тому саме тут стає важливим inout. Для оновлення енергії ми виконуємо поелементне множення на відбитий колір поверхні. Наприклад, для золота коефіцієнт дзеркального відображення приблизно дорівнює float3(1.0f, 0.78f, 0.34f), тобто воно відображає 100% червоного кольору, 78% зеленого кольору та всього 34% синього кольору, надаючи відображенню характерного золотистого відтінку.

У HLSL є вбудована функція для відображення променя із заданою нормаллю, і це зручно. Через неточність чисел з плаваючою комою може статися так, що відбитий промінь блокується поверхнею, від якої він відбивається. Щоб уникнути цього, ми трохи змістимо позицію вздовж напрямку нормалі.

Додамо більше сфер, та побачимо результат:





Висновок

У даній курсовій роботі були розглянуті основи трасування променів, яке є одним з ключових алгоритмів рендерингу комп'ютерних зображень. Трасування променів дозволяє моделювати поведінку світла, що падає на об'єкти сцени та відбивається від них, що призводить до отримання реалістичних зображень з високою якістю світлових ефектів, таких як тіні, відбиття та прозорість.

Процес трасування променів включає створення променя від камери, трасування цього променя через сцену, визначення перетину з об'єктами, обчислення властивостей світла та повторення цього процесу для кожного пікселя зображення. Однак, трасування променів вимагає значних обчислювальних ресурсів, оскільки потребує обчислення великої кількості променів та їх взаємодії з об'єктами сцени.

Завдяки технологічному прогресу, такому як графічні процесори та прискорювачі апаратного трасування променів, трасування променів стає більш доступним та ефективним у реальному часі. Ця техніка широко використовується в комп'ютерній графіці, візуалізації та фільмовому виробництві для створення вражаючих візуальних ефектів та реалістичних зображень.

У майбутньому, з вдосконаленням алгоритмів та апаратних засобів, очікується подальше розвиток трасування променів, що дозволить отримати ще більш реалістичні та високоякісні зображення.

Використані джерела:

Онлайн-стаття: http://blog.three-eyed-games.com/2018/05/03/gpu-ray-tracing-in-unity-part-1/

YouTube відео: https://www.youtube.com/watch?v=Qz0KTGYJtUk&ab_channel=SebastianLague

Стаття Вікіпедії: https://uk.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D1%81%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%BF%D1%80%D0%BE%D0%BC%D0%B5%D0%BD%D1%96%D0%B2#:~:text=ray%20tracing)%20%D1%83%20%D0%BA%D0%BE%D0%BC%D0%BF'%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D1%96%D0%B9,'%D1%94%D0%BA%D1%82%D0%B0%D0%BC%D0%B8%2C%20%D1%89%D0%BE%20%D0%BF%D1%96%D0%B4%D0%BB%D1%8F%D0%B3%D0%B0%D1%8E%D1%82%D1%8C%20%D0%B2%D1%96%D0%B4%D0%BE%D0%B1%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%BD%D1%8E.

Онлайн-стаття: https://jak.koshachek.com/articles/metod-zvorotnogo-trasuvannja-promeniv.html

Наукова стаття: https://advances.realtimerendering.com/s2019/Benyoub-DXR%20Ray%20tracing-%20SIGGRAPH2019-final.pdf

Наукова стаття: https://dl.acm.org/doi/10.1145/358876.358882