Симуляция и оптимизация physics-based освещения на примере трассировки лучей

Вавилина Екатерина P3230

Трассировка лучей позволяет моделировать свет так, как он ведёт себя в реальности — с учётом законов отражения, преломления и рассеивания.

Мы получаем:

Рефлексы

Отсвет цвета/света от окружающих объектов

Отражения

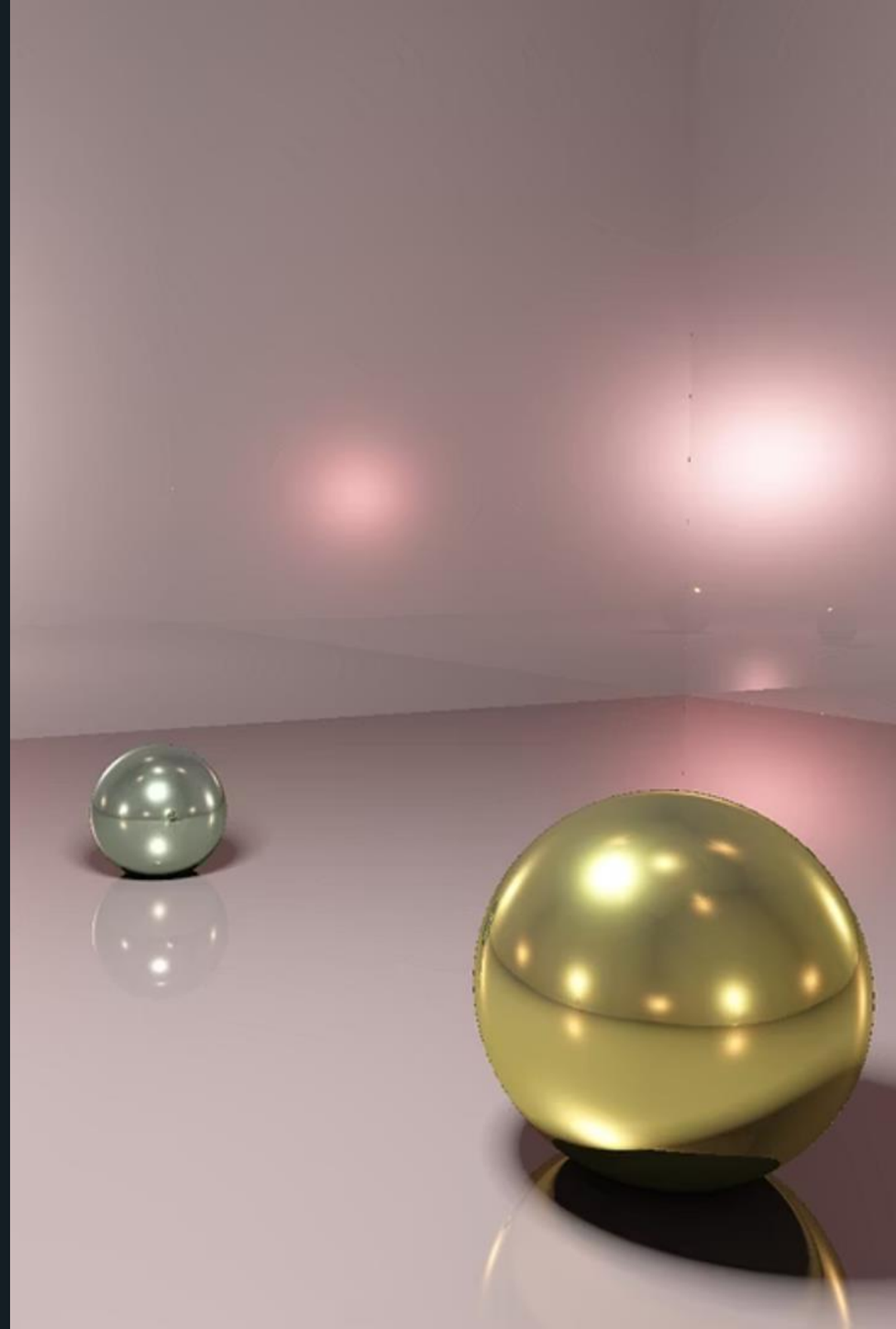
По настоящему честные отражения

Мягкие тени

Полутени и плавный переход между тенью и светом

Glossiness

Отражение зависит от цвета материала



Конвейер трассировки лучей

Генерация лучей

Отправка лучей из камеры через каждый пиксель Viewport (проекции области сцены, которую видит пользователь)

1

Поиск пересечений с геометрией

Определение объектов, с которым лучи сталкиваются в течении своего "жизненного цикла".

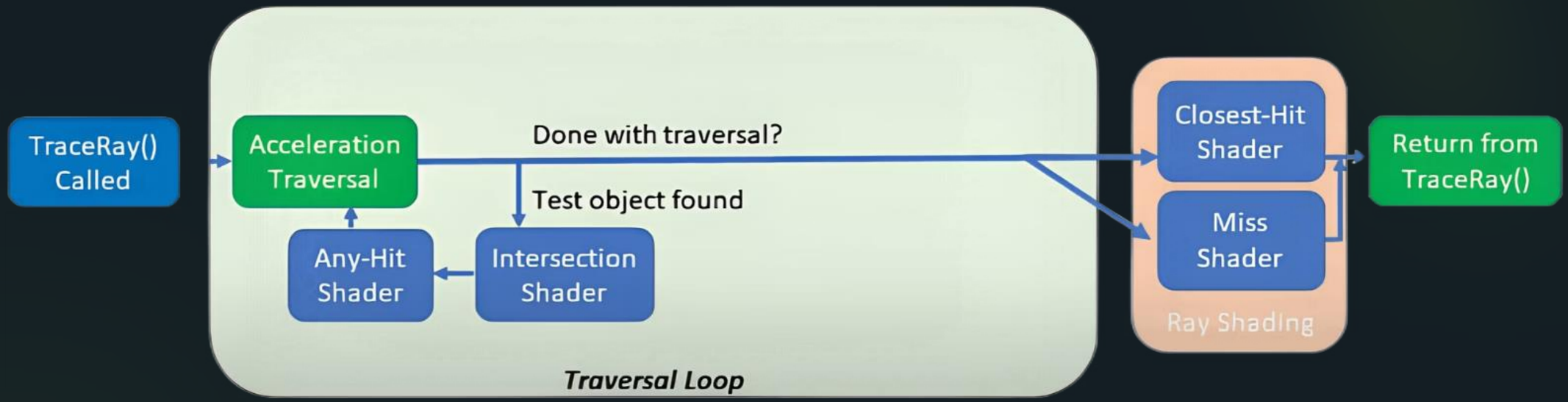
2

Расчет освещенности

Вычисление цвета и интенсивности света в точках пересечения и цвета пикселя Viewport

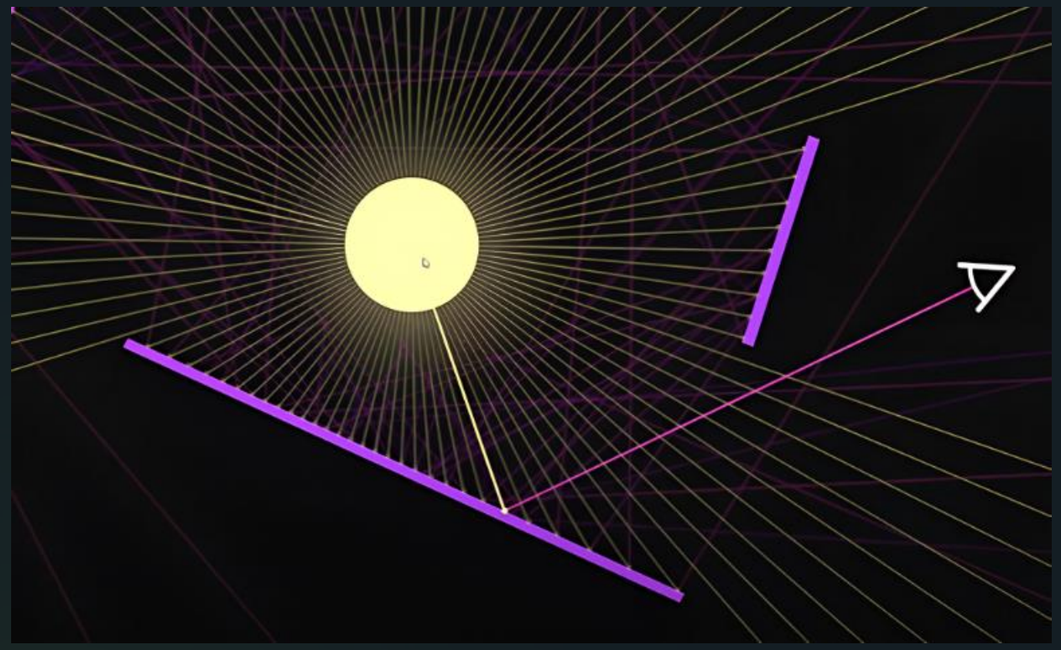
3

Конвейер трассировки лучей

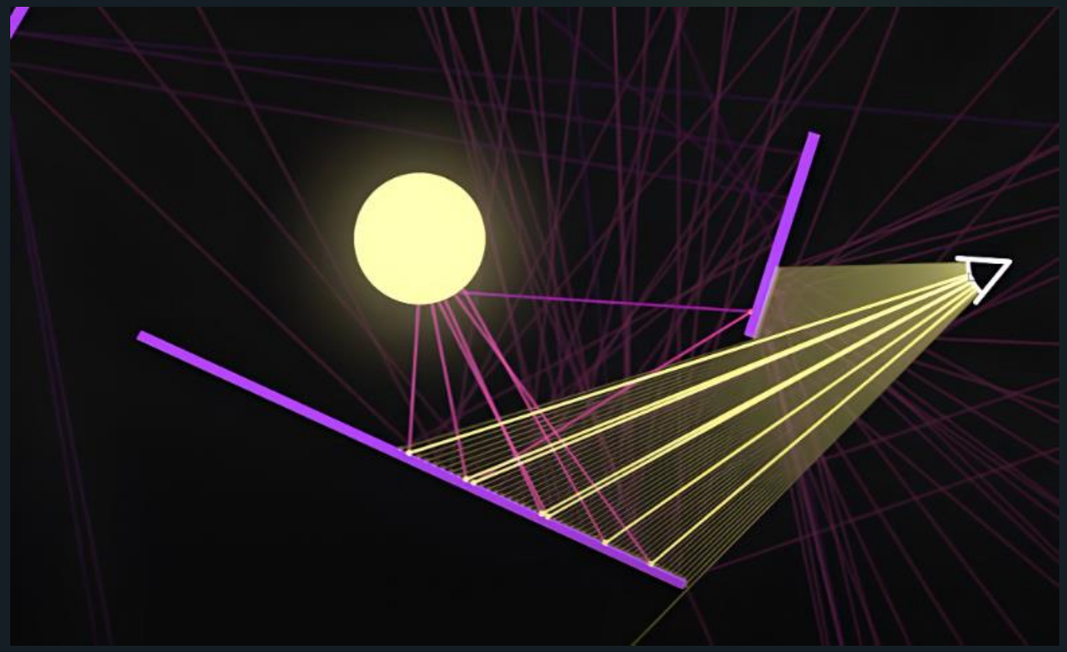


Генерация лучей

Лучи из источника света в камеру



Лучи из камеры в источник света

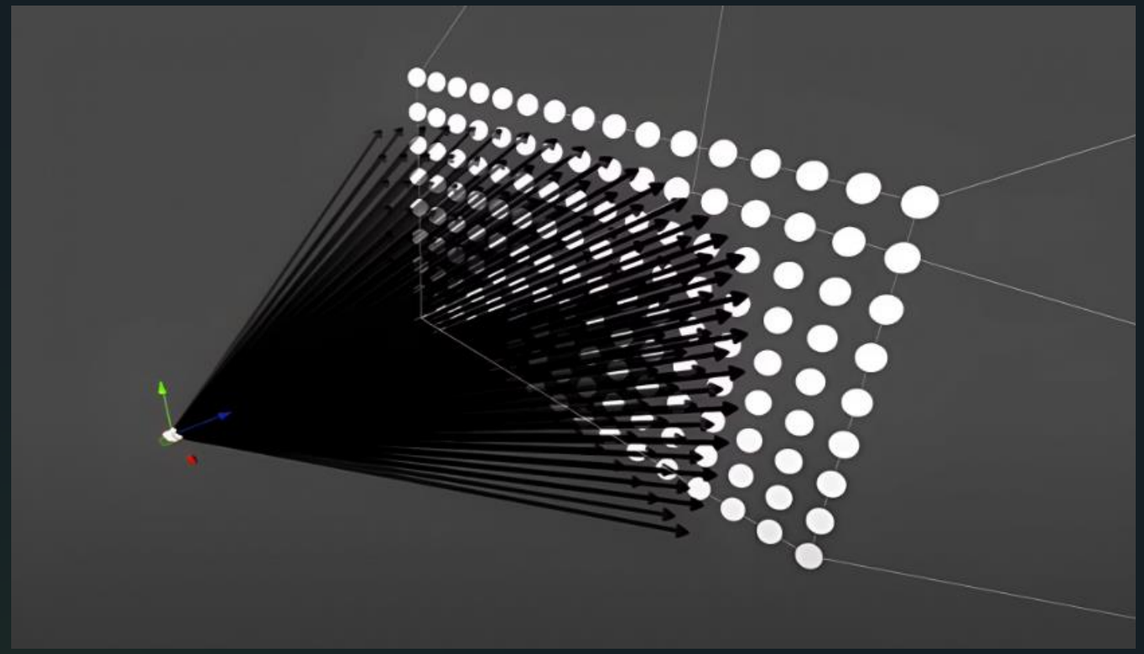


В первом случае мы генерируем слишком много лучей заря, да и источников света может быть много.

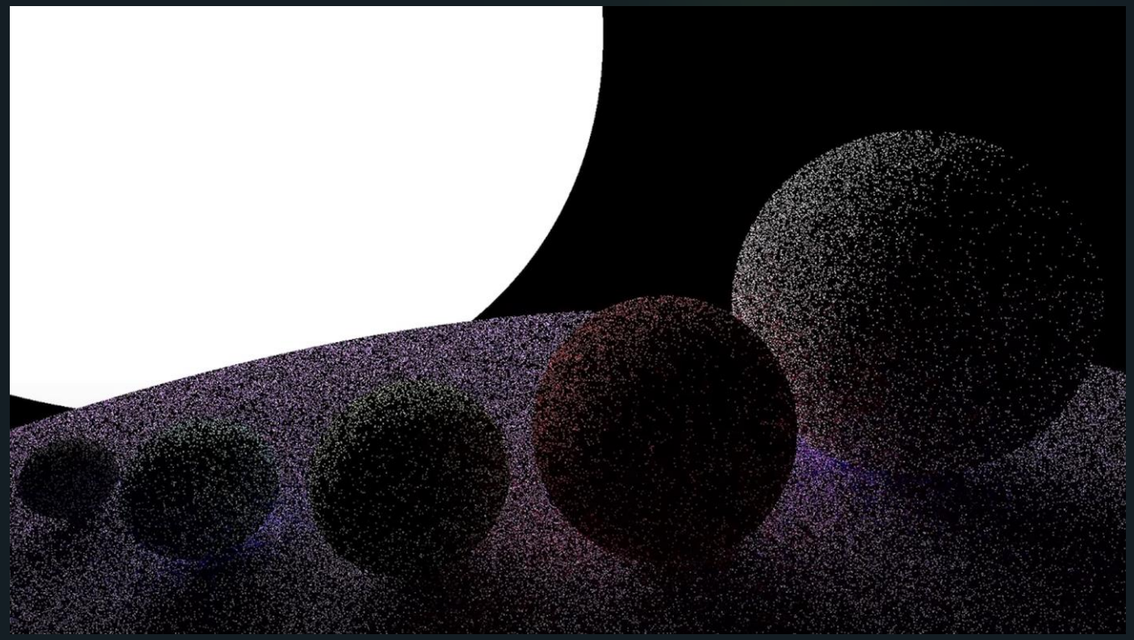
Поэтому выбираем второй вариант реализации.

Генерация лучей

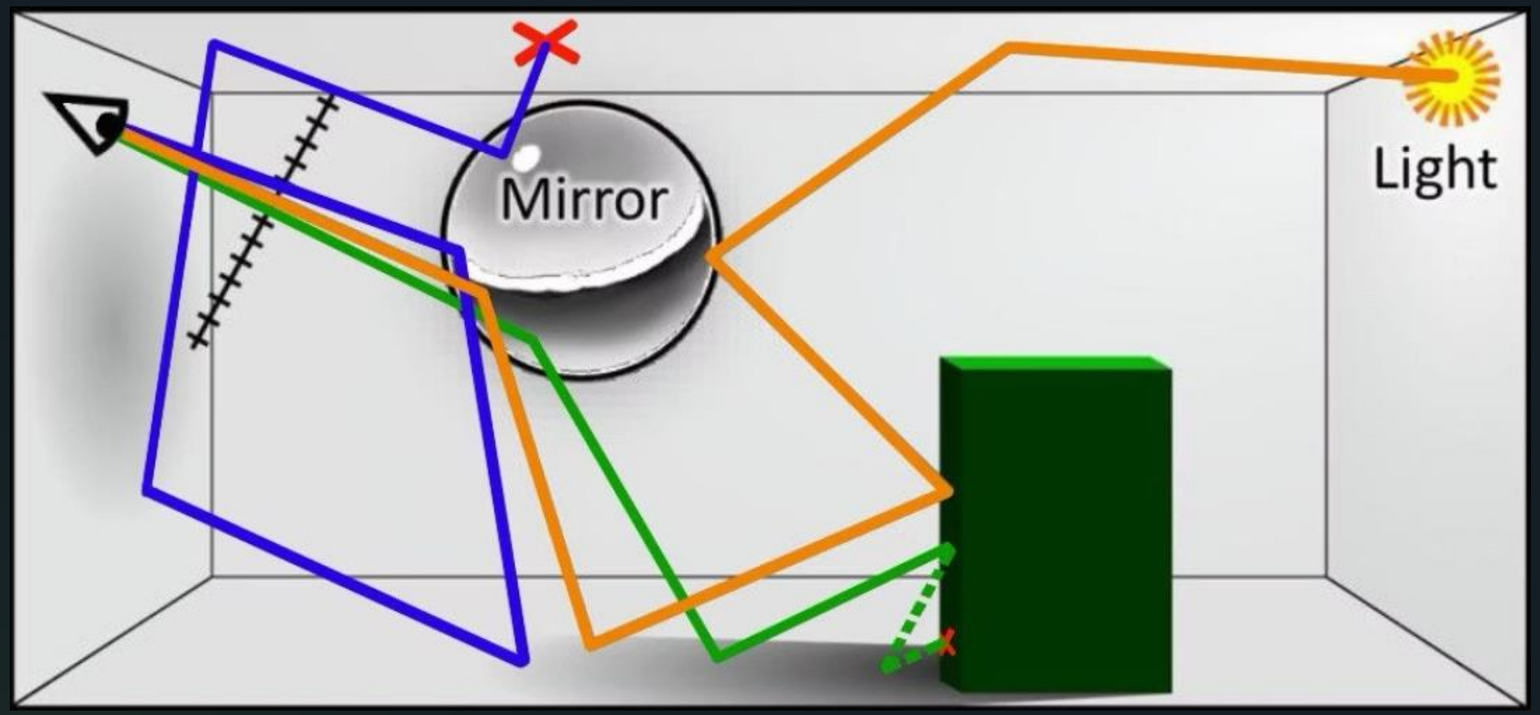
Viewport и идущие к нему лучи



Анимация увеличения количества лучей на пиксель



Путь луча по сцене



Останавливаемся когда:

1. Луч попал в источник света (оранжевый)
2. Достигнут максимум отскоков (синий)
3. Луч окончательно затух (зеленый)

Расчет столкновений. Рассеивание света.

Луч отражается в случайном направлении.

Для расчета освещенности может использоваться формула:

$$L = \sum_{i=0}^{N} E_i \cdot \prod_{j=0}^{i-1} C_j$$

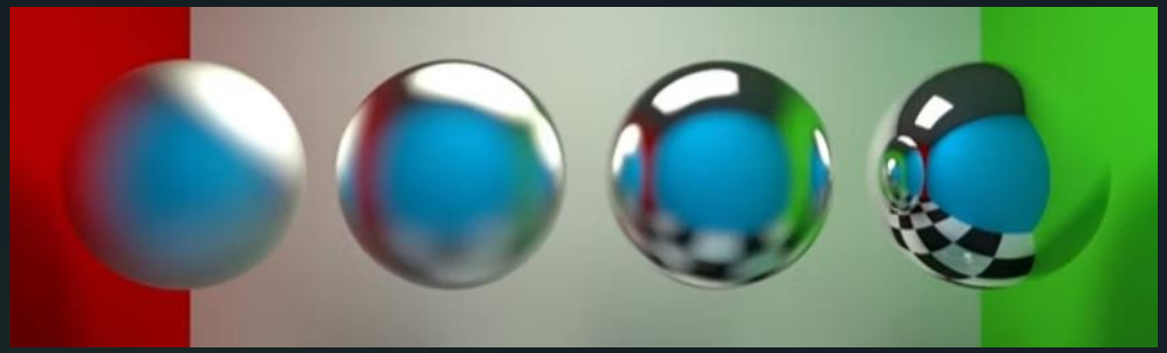
L - итоговая освещенность пикселя

E - яркость луча на i уровне

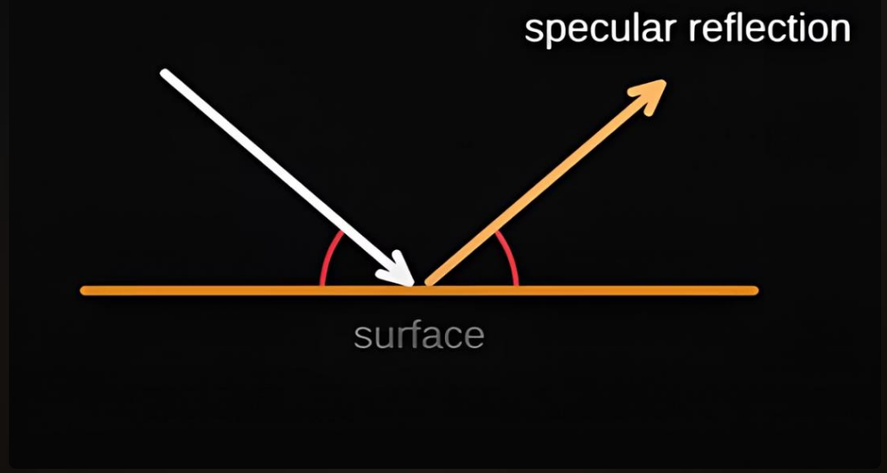
C - цвет материала на i уровне учетом затухания

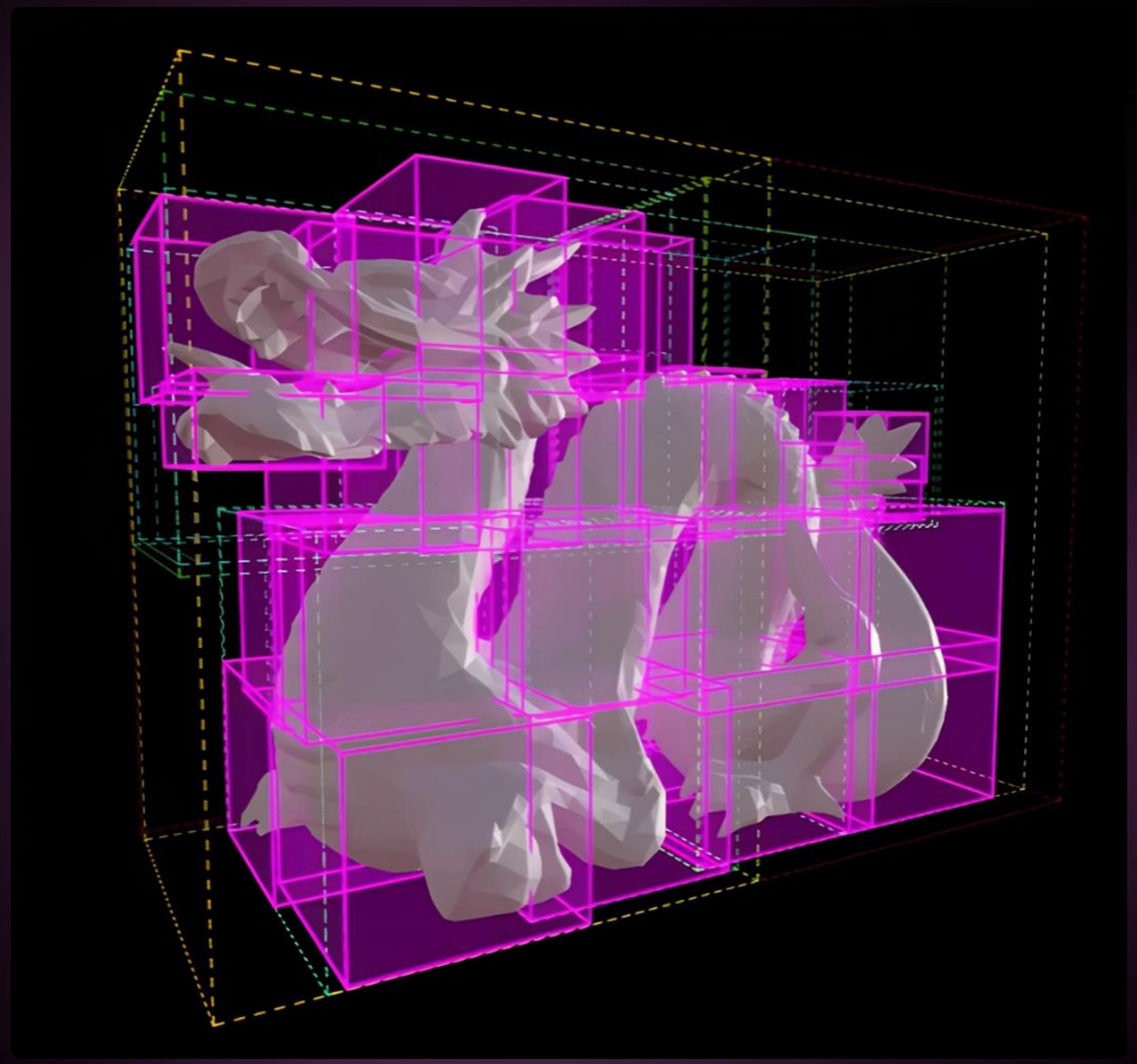
Расчет столкновения. Зеркальные отражений.

Если зеркало идеальное - всегда выбираем отражение. Иначе мы выбираем между отражением и рассеиванием на основании коэффициента отражения



! Процесс построения отражений ресурсоемкий, так как нам надо знать полный путь луча до источник света





Ускорение трассировки. Bounding Volume Hierarchy

1. Строим группы примитивов
2. Получаем иерархическое дерево групп
3. Ищем попадание в группу от корня к его наследникам до пересечения с примитивом

Ускорение трассировки. Surface Area Heuristic

SAH - эвристика, используемая для оптимального построения BVH. Она минимизирует ожидаемую стоимость проверок пересечений луча с геометрией .

$$\text{Cost} = C_{\text{traversal}} + \frac{A_1}{A_p} \cdot N_1 \cdot C_{\text{intersect}} + \frac{A_2}{A_p} \cdot N_2 \cdot C_{\text{intersect}}$$

Cost - стоимость проверки

A1 и A2 - площади поверхности левого и правого ящиков

Ap - площадь поверхности ящика-родителя

$C_{traversal}$, $C_{intersect}$ - оценочные стоимости операций (обход и пересечение)

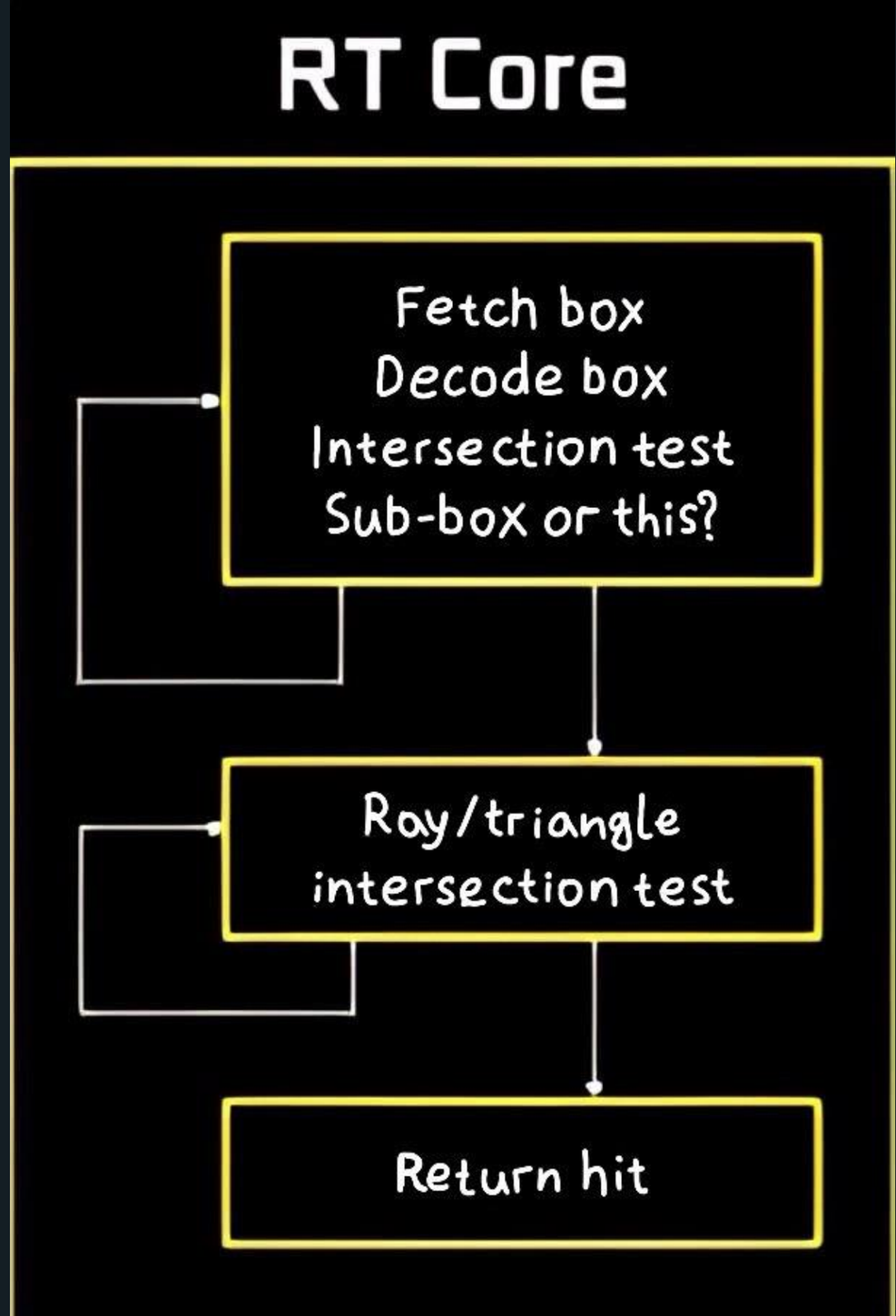
Аппаратная поддержка трассировки

RT Cores (Ray Tracing Cores) в GPU

Ускоряют поиск пересечений лучей в BVH

Acceleration Structures

Особые иерархические структуры, которые помогают оптимально хранить и работать с BVH.



Источники

1. Документация NVIDIA о RTX и архитектуре аппаратной трассировки лучей: https://developer.nvidia.com/rtx/raytracing

2. Конвейер трассировки лучей: https://developer.nvidia.com/blog/ray-tracing-essentials-part-4-the-ray-tracing-pipeline/

3. Видео "Coding Adventure: Ray Tracing" — Sebastian Lague, YouTube: https://www.youtube.com/watch?v=QzoKTGYJtUk

4. Видео "Coding Adventure: Ray Tracing Optimization" — Sebastian Lague, YouTube: https://www.youtube.com/watch?v=C1H4zIiCOaI

книги от Nvidia по трассировке лучей