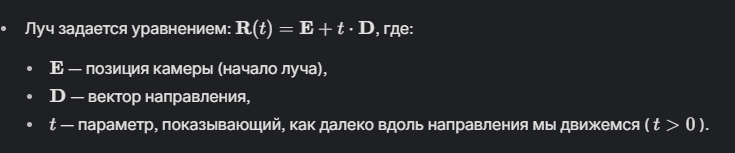
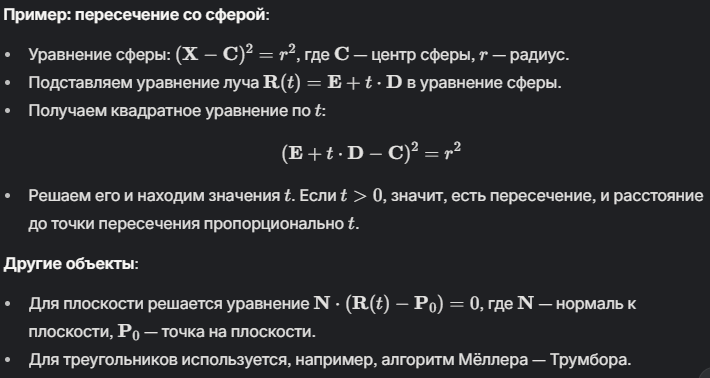
**Как луч реализуется программно**





А потом просто выбираем ближайшее пересечение

**Как трассировка лучей обрабатывает прозрачные объекты и материалы с преломлением, и какие вызовы это представляет для конвейера рендеринга?**  
Трассировка лучей обрабатывает прозрачные объекты и материалы с преломлением с помощью генерации **вторичных лучей** при пересечении исходного луча с такими поверхностями.

**ШЕЙДЕРЫ**

*в юнити шейдеры на hlsl, а на шейдертое glsl*

*GLSL ориентирован на кроссплатформенность и используется в OpenGL/Vulkan, тогда как HLSL привязан к экосистеме DirectX (Windows/Xbox).*

*(DirectX — это набор программных интерфейсов (API, Application Programming Interface), разработанных компанией Microsoft для упрощения создания мультимедийных приложений, в первую очередь компьютерных игр, на платформах Windows и Xbox. DirectX предоставляет разработчикам низкоуровневый доступ к аппаратным возможностям компьютера, включая графику, звук, ввод и сетевое взаимодействие.)*

Шейдеры — это небольшие программы, которые выполняются на GPU и определяют, как обрабатывать различные этапы рендеринга. В контексте трассировки лучей шейдеры используются для управления поведением лучей на разных этапах их пути через сцену. Они позволяют разработчикам настраивать, как лучи взаимодействуют с геометрией, как вычисляется освещение и что происходит, если луч не попал в объект.

**1. Any-Hit Shader**

* **Что делает?** Вызывается каждый раз, когда луч потенциально пересекает объект (примитив, например, треугольник), но до того, как будет определено ближайшее пересечение. Этот шейдер может решить, учитывать ли это пересечение или игнорировать его.
* **Пример использования:**
  + Для прозрачных объектов: Any-Hit Shader может проверить альфа-канал текстуры (прозрачность) и решить, пропустить ли луч через объект (игнорировать пересечение) или обработать его.
  + Для оптимизации: Если пересечение не влияет на результат (например, объект слишком далеко), шейдер может его отклонить.
* **На схеме:** После этапа "Acceleration Traversal" и проверки пересечения (Test object found), Any-Hit Shader решает, продолжить ли обработку данного пересечения.

**2. Intersection Shader**

* **Что делает?** Используется для определения пересечения луча с пользовательскими примитивами (например, не треугольниками, а какими-то сложными объектами, такими как сферы или процедурные поверхности). Этот шейдер заменяет стандартный алгоритм пересечения, предоставляемый RT Cores.
* **Пример использования:**
  + Если сцена содержит процедурную геометрию (например, облака или фрактальные поверхности), Intersection Shader вычисляет, пересекает ли луч эту геометрию.
* **На схеме:** Вызывается после обхода структуры ускорения (Traversal), если объект найден. Это этап "Test object found", где шейдер определяет точное пересечение для нестандартных примитивов.

**3. Closest-Hit Shader**

* **Что делает?** Вызывается после того, как цикл обхода (Traversal Loop) завершён и определено ближайшее пересечение луча с объектом. Этот шейдер отвечает за вычисление цвета, освещения и других свойств в точке пересечения.
* **Пример использования:**
  + Вычисляет освещение: учитывает материал объекта, нормали, текстуры, а также источники света.
  + Может генерировать вторичные лучи: например, для отражений или преломлений, вызывая новый TraceRay().
* **На схеме:** После завершения обхода ("Done with traversal?") и определения ближайшего пересечения, Closest-Hit Shader выполняется, чтобы обработать результат пересечения.

**4. Miss Shader**

* **Что делает?** Вызывается, если луч не пересёк ни один объект в сцене (то есть "промахнулся"). Этот шейдер определяет, что делать в таком случае, например, какой цвет присвоить пикселю.
* **Пример использования:**
  + Устанавливает цвет фона: например, если луч ушёл в небо, Miss Shader может вернуть цвет неба или текстуру окружающей среды.
  + Может учитывать глобальное освещение: например, если луч не попал в источник света, шейдер может вернуть значение по умолчанию.
* **На схеме:** Если после обхода ("Done with traversal?") пересечений не найдено, вызывается Miss Shader.

**Общий контекст шейдеров в схеме**

* **Traversal Loop (Цикл обхода):** Этот цикл отвечает за обход структуры ускорения (например, BVH), чтобы найти потенциальные пересечения луча с геометрией. Шейдеры вызываются на разных этапах этого процесса для обработки результатов.
* **Ray Shading (Затенение луча):** После определения пересечения (или его отсутствия) шейдеры (Closest-Hit или Miss) выполняют затенение, то есть вычисляют, как луч влияет на итоговый цвет пикселя.
* **Рекурсия:** Closest-Hit Shader может вызвать новый TraceRay() для вторичных лучей (например, отражений), что создаёт рекурсивный процесс.

**Какие современные движки или библиотеки позволяют реализовать трассировку лучей?**

**Игровые движки с поддержкой трассировки лучей**

Современные игровые движки интегрируют поддержку трассировки лучей через API, такие как DirectX Raytracing (DXR), Vulkan Ray Tracing и Metal (для macOS). Они также часто используют дополнительные технологии, такие как NVIDIA DLSS, для оптимизации производительности.

**1.1 Unreal Engine**

* **Версии с поддержкой:** Unreal Engine 4 (начиная с версии 4.22, анонсированной в 2019 году) и Unreal Engine 5.
* **Особенности:**
  + Unreal Engine 4.22 стал одним из первых движков, официально поддерживающих трассировку лучей в реальном времени, что было анонсировано NVIDIA в 2019 году.
  + В Unreal Engine 5 появилась система **Lumen**, которая предоставляет глобальное освещение и отражения с использованием трассировки лучей. Lumen поддерживает как аппаратную трассировку (через RT-ядра), так и программную (на основе mesh distance fields), что делает её доступной для широкого спектра устройств.
  + Поддерживает **DXR**, **Vulkan Ray Tracing** и **NVIDIA DLSS** (включая DLSS 3 и 3.5 с Ray Reconstruction для повышения качества изображения).

**1.2 Unity**

* **Версии с поддержкой:** Unity 2020 и выше, с полной поддержкой в Unity 2022 и 2023 через High Definition Render Pipeline (HDRP).
* **Особенности:**
  + Unity интегрировала поддержку трассировки лучей через **DXR** и **Vulkan Ray Tracing**.
  + Поддерживает **NVIDIA DLSS** (начиная с DLSS 2.0, а позже и DLSS 3), что помогает компенсировать высокую вычислительную нагрузку.
  + Unity использует **Ray Tracing Shader Graph** для создания кастомных эффектов освещения, отражений и теней.
  + Поддерживает **Path Tracing** для фотореалистичного рендеринга в оффлайн-режиме.

**1.4 Frostbite Engine**

* **Версии с поддержкой:** Frostbite 4 и выше (используется EA с 2019 года).

**2. Библиотеки для реализации трассировки лучей**

Если вы хотите реализовать трассировку лучей самостоятельно или интегрировать её в кастомный движок, существуют библиотеки, которые предоставляют необходимые инструменты.

**2.1 NVIDIA OptiX**

* **Описание:** OptiX — это API от NVIDIA, предназначенное для трассировки лучей в реальном времени. Впервые представлено в 2009 году.

**2.2 Vulkan Ray Tracing (Khronos Group)**

* **Описание:** Расширение Vulkan API, добавленное в 2020 году, для кроссплатформенной трассировки лучей.

**2.3 DirectX Raytracing (DXR)**

* **Описание:** Часть DirectX 12 Ultimate, представленная Microsoft в 2018 году.

**2.4 LuxCoreRender**

* **Описание:** Открытая библиотека для физически корректного рендеринга, основанная на трассировке лучей.

**2.5 Radeon ProRender (AMD)**

* **Описание:** Библиотека от AMD для трассировки лучей, представленная в 2016 году, с поддержкой аппаратного ускорения с 2020 года (Radeon RX 6000).

**Чем path tracing отличается от ray tracing и на каких этапах?**

Умеет в расчет освещенности сцены, вычищение всех возможных шумов с помощью алгоритма Монте-Карло (привет вычмат на вт).

По сути, тоже решает уравнение рендеринга численно, но учитывает общую освещенность сцены. Каустика

**Какие есть альтернативы BVH и в каких случаях они предпочтительнее?**

**1. Kd-Tree (K-Dimensional Tree)**

**Описание**

Kd-Tree — это структура данных, которая рекурсивно разбивает пространство на гиперпрямоугольники (в 3D — это параллелепипеды) вдоль осей. Каждый узел дерева представляет собой разделяющую плоскость, которая делит пространство на две части, а листья содержат примитивы (например, треугольники).

**Когда предпочтительнее BVH?**

* **Оффлайн-рендеринг:** Kd-Tree часто используется в оффлайн-рендеринге (например, в Path Tracing для кинематографа), где время построения не критично, а точность важна. Например, в рендерерах, таких как PBRT (Physically Based Rendering Toolkit) или Blender Cycles.
* **Сцены с высокой плотностью примитивов:** Если сцена содержит много мелких объектов (например, листва, волосы), Kd-Tree может быть эффективнее благодаря точному разбиению.
* **Статические сцены:** Kd-Tree идеален, если сцена не меняется (например, архитектурная визуализация).

**2. Uniform Grid (Равномерная сетка)**

**Описание**

Uniform Grid делит пространство сцены на равномерную сетку (обычно 3D-грид, где каждая ячейка — куб). Каждый примитив (например, треугольник) помещается в ячейки сетки, через которые он проходит. Лучи проходят через ячейки сетки, проверяя пересечения только с примитивами, находящимися в этих ячейках.

Uniform Grid часто используется в научных симуляциях (например, в вычислительной физике), где сцена состоит из множества одинаковых частиц, равномерно распределенных в пространстве.

**3. Octree (Октодерево)**

**Описание**

Octree — это иерархическая структура, которая делит пространство на восемь частей (октантов) на каждом уровне. Это похоже на Kd-Tree, но вместо разбиения по одной оси на каждом уровне,

**4. Bounding Interval Hierarchy (BIH)**

**Описание**

Bounding Interval Hierarchy (BIH) — это структура, похожая на Kd-Tree, но вместо разбиения пространства по одной оси она использует два интервала (bounding intervals) для каждой оси. BIH фокусируется на минимизации перекрытий между интервалами.

**5. Spatial Hashing (Пространственное хеширование)**

**Описание**

Spatial Hashing делит пространство на ячейки (как Uniform Grid), но вместо явного хранения сетки использует хеш-таблицу для быстрого доступа к ячейкам. Каждый примитив хешируется в одну или несколько ячеек на основе его позиции.

Spatial Hashing часто применяется в симуляциях частиц или жидкостей (например, в Houdini или в игровых движках для физики).

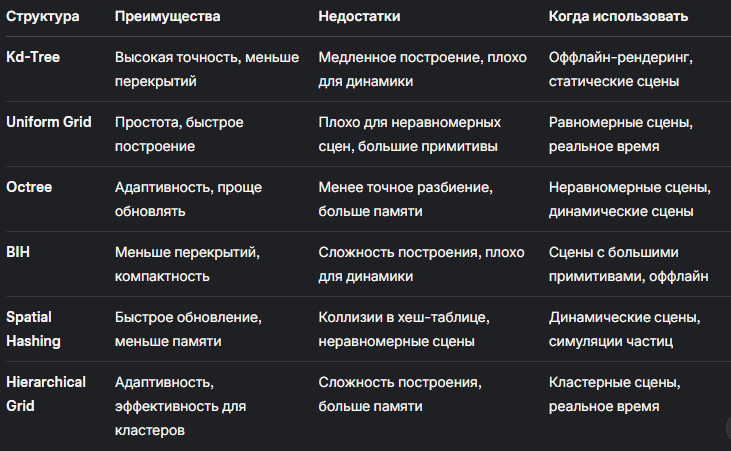
**6. Hierarchical Grid (Иерархическая сетка)**

**Описание**

Hierarchical Grid — это комбинация Uniform Grid и иерархических структур. Пространство сначала делится на крупную сетку, а затем ячейки с большим числом примитивов дополнительно разбиваются на меньшие подсетки.

**Когда предпочтительнее BVH?**

* **Кластерные сцены:** Хорошо подходит для сцен с неравномерным распределением (например, городская среда с зданиями и пустыми пространствами).
* **Реальное время:** Может быть использован в играх для управления видимостью или трассировки лучей в простых сценах.



**как реализуется неполное внутренне отражение**

Неполное внутреннее отражение, как в вашем примере с автобусным стеклом, реализуется в трассировке лучей через комбинацию отражения и преломления с использованием закона Френеля. Лучи случайным образом выбирают путь (отражение с вероятностью RRR или преломление с вероятностью TTT), что позволяет одновременно отобразить внутреннее освещение и внешнюю дорогу. В растеризации прозрачные и полупрозрачные объекты рендерятся отдельно с альфа-смешиванием, но в трассировке они обрабатываются в рамках единого конвейера с учетом их физических свойств. Таким образом, ваше предположение верно для растеризации, но в трассировке "отдельные законы" — это часть общего процесса, а не отдельный рендеринг. Современные технологии, такие как DXR и RT-ядра, делают этот эффект реалистичным даже в реальном времени.

Производительность: Полная трассировка преломленных лучей может быть ресурсоемкой, поэтому в играх часто применяются упрощения (например, ограничение глубины рекурсии до 2–3 отскоков).

**можно ли реализовать лучи с помощью волн**

Для реализации волнового подхода в трассировке лучей можно адаптировать существующие методы:

* **Гибридный метод:** Комбинировать геометрическую трассировку лучей с волновыми эффектами. Например:
  + Использовать трассировку лучей для определения прямых путей и первых отражений (как в акустической симуляции ранних отражений).
  + Добавить численное моделирование дифракции и интерференции для более сложных сценариев.
* **Метод конечных разностей во времени (FDTD):** Это численный метод решения волнового уравнения, который можно интегрировать с трассировкой лучей. FDTD моделирует распространение волны в пространстве и времени, учитывая границы и препятствия.
* **Beam Tracing с волновыми корректировками:** Beam Tracing (трассировка лучевых пучков) уже используется в акустике для моделирования распространения звука. Расширив его с учетом амплитуды и фазы, можно приблизить волновое поведение.

**5.1 Гибридные методы**

* **Photon Mapping с волнами:** Photon Mapping (трассировка фотонов) уже использует концепцию частиц света. Расширив его для акустики с учетом фазы, можно получить волновой эффект.
* **Hybrid Ray-FDTD:** Комбинация трассировки лучей для крупных отражений и FDTD для дифракции уже применяется в акустическом моделировании (например, в ПО вроде Odeon или CATT-Acoustic).

**5.2 Реальные реализации**

* **Odeon:** Коммерческое ПО для акустического моделирования, использующее гибрид трассировки лучей и волновых методов.
* **Blender с Cycles:** Хотя Cycles использует Path Tracing для света, его можно адаптировать для акустики с кастомными шейдерами и плагинами.
* **Unity/Unreal Engine:** С поддержкой трассировки лучей (DXR/Vulkan Ray Tracing) можно разработать плагин для волновой акустики, хотя это требует значительных усилий.

**TLAS и BLAS в динамике**  
**2.1 Разделение геометрии и трансформаций**

* **Геометрия в BLAS:** BLAS хранит геометрию объекта в локальной системе координат (local space). Например, треугольники автомобиля описываются относительно его собственной системы координат, где центр (0, 0, 0) может быть в середине машины.
* **Трансформация в TLAS:** TLAS хранит матрицу трансформации (4x4), которая переводит BLAS из локального пространства в мировое (world space). Например, если автомобиль движется, его BLAS остается неизменным, но матрица трансформации в TLAS обновляется, чтобы отразить новое положение.

**2.2 Обновление TLAS**

Когда объект движется:

* **Обновление трансформации:** Матрица трансформации для BLAS в TLAS изменяется, чтобы отразить новое положение, вращение или масштаб объекта.
* **Обновление AABB в TLAS:** Каждый узел TLAS, который ссылается на BLAS, обновляет свой ограничивающий объем (AABB). Это делается путем применения матрицы трансформации к AABB BLAS и пересчета нового AABB в мировом пространстве.
* **Частичная перестройка:** В зависимости от реализации, TLAS может либо полностью перестраиваться, либо обновляться частично:
  + **Полная перестройка:** Все узлы TLAS пересчитываются с нуля. Это дорогостоящий процесс, но может быть оправдан, если много объектов изменили положение.
  + **Частичное обновление (refit):** Обновляются только AABB узлов, которые ссылаются на изменившиеся BLAS’ы. Это быстрее, так как структура дерева TLAS не меняется, а пересчитываются только границы.

**3.1 Когда перестроение BLAS не требуется**

* **Движение/вращение/масштаб:** Если объект просто перемещается, вращается или масштабируется, BLAS не нужно перестраивать. Геометрия объекта (треугольники, вершины) не меняется, а все изменения обрабатываются на уровне TLAS через матрицу трансформации.
* **Причина:** BLAS описывает геометрию в локальном пространстве, и его структура (BVH) не зависит от положения объекта в мире. Трансформация выполняется на этапе обхода TLAS.

**3.2 Когда перестроение BLAS требуется**

Перестроение BLAS необходимо, если изменяется сама геометрия объекта:

* **Деформация:** Если объект деформируется (например, анимация персонажа с движением конечностей, сгибающийся лист дерева), локальная геометрия меняется, и BLAS нужно перестроить.
  + Пример: Персонаж бежит, и его руки двигаются. Треугольники, описывающие руки, изменяют свои позиции в локальном пространстве, что требует пересчета BVH в BLAS.
* **Добавление/удаление примитивов:** Если в объекте добавляются или удаляются треугольники (например, разрушение объекта), BLAS также нужно перестроить.
* **Изменение топологии:** Если топология объекта меняется (например, при генерации процедурных объектов), BLAS нужно обновить.

**Какие ограничения есть у аппаратных RT-ускорителей — в чём они пока уступают кастомным программным решениям?**

**Жёсткая архитектура и отсутствие гибкости (и узконаправленность)**

**Ограниченная поддержка сложных алгоритмов**

Описание: RT-ядра заточены под классическую трассировку лучей (Whitted Ray Tracing) и базовый Path Tracing с ограниченным числом отскоков (обычно 2–6). Они не предназначены для продвинутых алгоритмов, таких как Bidirectional Path Tracing, Metropolis Light Transport (MLT) или Photon Mapping, которые требуют сложной обработки случайных путей.

**Ограниченная** **глубина рекурсии**

RT-ядра поддерживают фиксированное максимальное количество отскоков лучей (например, 31 уровень рекурсии в DXR или Vulkan Ray Tracing). Это ограничивает моделирование сложных световых или звуковых путей с множественными отражениями и преломлениями.

**Какие существуют способы ускорения рендеринга без использования аппаратных RT Cores?**

**1.2 Упрощение геометрии и сцены**

* **LOD (Level of Detail):** Использование моделей с разным уровнем детализации в зависимости от расстояния до камеры. Например, удалённые объекты можно заменить низкополигональными версиями, чтобы уменьшить количество треугольников.
* **Impostors/Billboards:** Для дальних объектов (например, деревьев) можно использовать плоские текстуры вместо полной 3D-геометрии, что снижает число проверок пересечений.
* **Culling (отсечение):**
  + **Frustum Culling:** Исключение объектов, не попадающих в область видимости камеры.
  + **Occlusion Culling:** Пропуск объектов, скрытых за другими (например, за стенами).

**1.3 Оптимизация выборки лучей**

* **Адаптивная выборка (Adaptive Sampling):** Уменьшение числа лучей в областях с низкой вариативностью (например, равномерные поверхности) и увеличение выборок в сложных областях (например, края объектов или тени).
* **Важностная выборка (Importance Sampling):** Генерация лучей с учётом распределения света (например, выбор направлений, более вероятных для попадания в источники света). Это снижает шум в Path Tracing.

**1.4 Пространственная и временная когерентность**

* **Кэширование путей (Path Caching):** Ваш вопрос о кэше путей действительно относится к этому методу. Если лучи совпадают или имеют схожие траектории (например, в последовательных кадрах или в соседних пикселях), их пути можно кэшировать, чтобы избежать повторных вычислений. Подробно разберём это ниже.
* **Кластерная трассировка (Packet Tracing):** Группировка лучей в пакеты (например, 4x4 пикселя) для совместного обхода BVH. Это улучшает когерентность и снижает количество операций.