УДК 004.925.3

ОПТИМИЗАЦИЯ ТРАССИРОВКИ ПУТИ В VULKAN ЧЕРЕЗ КЛАСТЕРИЗАЦИЮ ГЕОМЕТРИИ И РАЗДЕЛЕНИЕ СТРУКТУР УСКОРЕНИЯ ВЕРХНЕГО УРОВНЯ

Акименко А.В., студент гр. 151004

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Ярмолик В.Н. – доктор техн. наук, профессор

Аннотация. В статье исследуется метод оптимизации производительности трассировки пути (path tracing) для рендеринга крупномасштабных и динамичных трёхмерных сцен с использованием графического API Vulkan и его расширений для трассировки лучей (VK_KHR_ray_tracing_pipeline, VK_KHR_acceleration_structure). Предлагается и детально анализируется подход, основанный на пространственной кластеризации геометрии сцены и построении множественных, независимых структур ускорения верхнего уровня (TLAS) вместо традиционной единой монолитной структуры. Рассматриваются стратегии кластеризации, аспекты реализации в Vulkan, включая управление ресурсами и адаптацию шейдерного конвейера. Анализируются преимущества данного метода в контексте снижения вычислительных затрат на построение и обновление TLAS, улучшения локальности данных при обходе структур ускорения графическим процессором, а также возможностей для реализации эффективных стратегий отсечения (culling) невидимых частей сцены.

Ключевые слова. Трассировка пути, Vulkan, трассировка лучей, структуры ускорения, TLAS, BLAS, рендеринг, 3D графика, оптимизация GPU, крупномасштабные сцены, кластеризация геометрии, отсечение геометрии, VK_KHR_ray_tracing_pipeline, VK_KHR_acceleration_structure, управление памятью GPU.

Трассировка пути (path tracing) утвердилась как основной метод для достижения фотореалистичного качества визуализации в компьютерной графике благодаря своей способности точно моделировать сложные явления глобального освещения, такие как мягкие тени, каустики, непрямое освещение и реалистичные отражения/преломления [3]. Однако, фундаментальная стохастическая природа метода Монте-Карло, лежащего в его основе, требует генерации и обработки огромного количества лучей для получения изображения без шума, что накладывает экстремальные вычислительные требования. Особенно остро эта проблема проявляется при рендеринге крупномасштабных сцен, содержащих миллиарды геометрических примитивов, характерных для современных игр с открытым миром или сложных CAD/CAM моделей [1].

Появление низкоуровневых графических API, таких как Vulkan, и стандартизированных расширений для аппаратной трассировки лучей (VK_KHR_acceleration_structure, VK_KHR_ray_tracing_pipeline, VK_KHR_ray_query и др.) [2] предоставило разработчикам беспрецедентный контроль над графическим процессором (GPU) и возможность использовать специализированные аппаратные блоки для ускорения операций поиска пересечений лучей с геометрией. Тем не менее, даже с аппаратным ускорением, эффективность построения, обновления и обхода структур ускорения (Acceleration Structures – AS) остается критически важным фактором производительности, особенно для динамичных сцен большого масштаба.

Стандартный подход к организации геометрии для трассировки лучей в Vulkan предполагает использование двухуровневой иерархии структур ускорения: множество структур ускорения нижнего уровня (Bottom-Level Acceleration Structure – BLAS), каждая из которых строится для уникальной геометрии (например, меша объекта), и единственная структура ускорения верхнего уровня (Top-Level Acceleration Structure – TLAS), которая содержит экземпляры (instances) всех BLAS в сцене с их трансформациями.

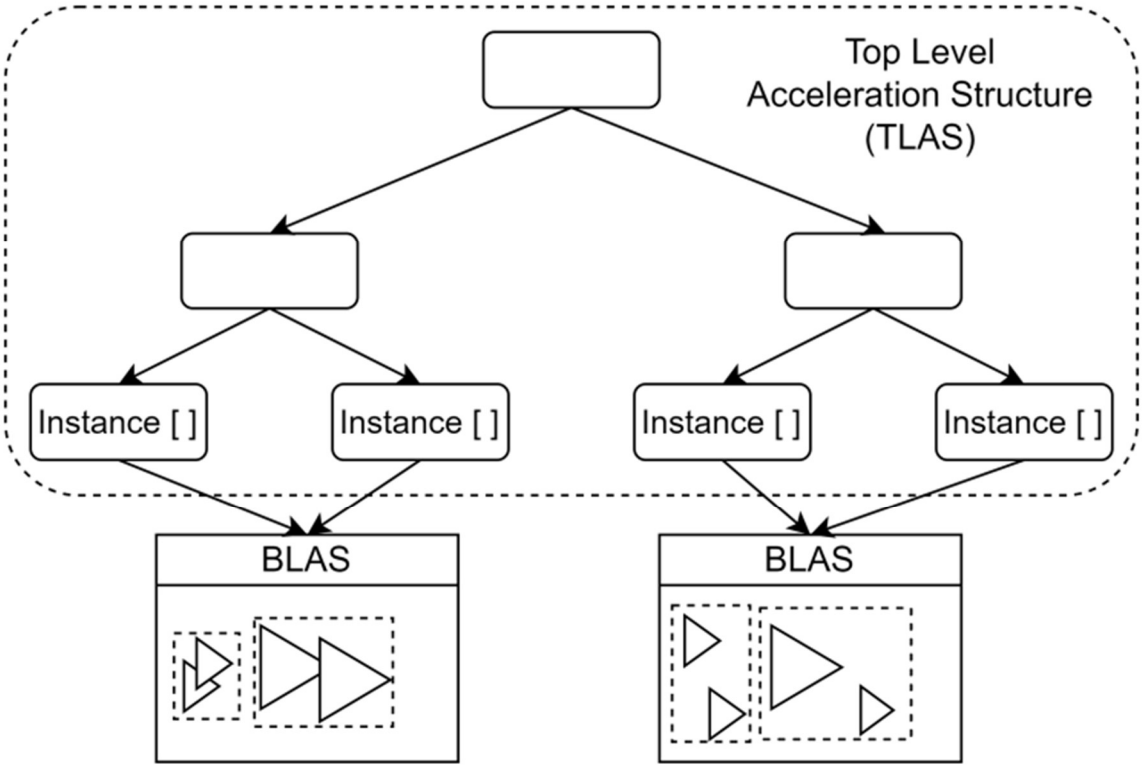

Рисунок 1 – Обзор структуры Top-Level Acceleration Structure

Хотя этот подход хорошо работает для статичных или умеренно сложных сцен, он сталкивается с серьезными проблемами масштабируемости при работе с «мега-геометрией»:

1. Высокая стоимость построения/обновления TLAS. Построение TLAS, содержащей миллионы или миллиарды экземпляров, является вычислительно дорогой операцией. В динамичных сценах, где объекты постоянно движутся, добавляются или удаляются, требуется частое обновление (refitting) или полное перестроение (rebuilding) TLAS. Обновление даже одного экземпляра в монолитной TLAS может потребовать обновления значительной части её внутренних узлов, а полное перестроение – это крайне затратная операция, потенциально занимающая несколько миллисекунд и становящаяся «узким местом» всего кадра.

2. Неэффективность обхода. Обход очень большой TLAS для каждого луча может страдать от плохой локальности данных. Узлы TLAS, соответствующие геометрически удаленным частям сцены, могут попадать в кэш GPU, вытесняя более релевантные данные и приводя к увеличению задержек доступа к памяти (cache thrashing). Это снижает эффективность использования аппаратных блоков трассировки лучей.

3. Ограниченные возможности отсечения. Единая TLAS не предоставляет удобных механизмов для гранулярного отсечения (culling) больших частей сцены, которые заведомо невидимы (например, находятся за пределами пирамиды видимости камеры или полностью перекрыты другими объектами). Хотя аппаратный обход TLAS сам по себе выполняет пространственное отсечение, он все равно требует обработки корневых узлов структуры, что может быть излишним, если известно, что целые регионы сцены невидимы.

Для преодоления ограничений монолитной TLAS, предлагается метод, основанный на разделении геометрии сцены на пространственные или логические кластеры и построении отдельной, независимой TLAS для каждого кластера.

Первым шагом является разбиение всего множества экземпляров геометрии сцены на непересекающиеся или частично пересекающиеся подмножества – кластеры. Эффективность всего подхода во многом зависит от выбранной стратегии кластеризации:

– Пространственное разбиение. Наиболее распространенный подход, использующий пространственные структуры данных для декомпозиции сцены. Вариациями пространственного разбиения являются:

1. Регулярная сетка (Grid). Пространство сцены делится на ячейки одинакового размера. Просто в реализации, но может приводить к несбалансированным кластерам (очень много геометрии в одних ячейках и пусто в других).

2. Иерархические структуры (Octree, KD-Tree, BVH). Адаптивно делят пространство, создавая более сбалансированные кластеры, лучше соответствующие распределению геометрии. Требуют более сложной реализации и управления.

3. Семантическая группировка. Объекты объединяются по логическому признаку, не обязательно связанному с их пространственным положением (например, все статические объекты, все динамические объекты, объекты одного типа, компоненты одного сложного механизма). Может быть полезна для специфических сценариев обновления или шейдинга.

– Комбинированный подход. Сочетание пространственного и семантического разбиения для достижения наилучшего баланса. Например, пространственное разбиение для статической геометрии и отдельные кластеры для крупных динамических объектов.

Выбор стратегии должен учитывать характеристики сцены (статичная/динамичная, плотность геометрии) и целевые задачи оптимизации (скорость обновления, эффективность обхода, гранулярность отсечения).

Вместо одной команды vkCmdBuildAccelerationStructuresKHR для построения глобальной TLAS, выполняется серия вызовов для создания или обновления отдельных TLAS для каждого кластера. Каждая TLAS содержит только те экземпляры геометрии (и ссылки на их BLAS), которые принадлежат данному кластеру.

Это приводит к ряду преимуществ:

– Снижение затрат на обновление. При изменении трансформации или видимости объекта требуется обновить или перестроить только ту TLAS (или те TLAS, если объект на границе), к которой он принадлежит. Поскольку каждая TLAS значительно меньше монолитной, эти операции выполняются гораздо быстрее. Обновления разных TLAS могут выполняться параллельно на GPU.

– Улучшение локальности данных. При трассировке луча, пересекающего объекты одного кластера, GPU будет с большей вероятностью работать с данными (узлами TLAS, экземплярами, матрицами трансформаций), находящимися близко друг к другу в памяти и потенциально кэшированными. Это повышает эффективность обхода AS.

– Эффективное отсечение (Culling). Появляется возможность выполнять отсечение на уровне кластеров. На CPU можно определить, какие кластеры (и их TLAS) пересекаются с пирамидой видимости камеры (frustum culling) или не перекрыты другими объектами (occlusion culling). В шейдер генерации лучей (Ray Generation shader) передается только список «активных» TLAS, и трассировка выполняется только по ним, полностью игнорируя геометрию в отсеченных кластерах.

Управление множеством TLAS требует более сложной логики со стороны приложения. Необходимо хранить и управлять массивом дескрипторов VkAccelerationStructureKHR и соответствующими буферами памяти. На Рисунках 2 и 3 схематично показаны различия между монолитным и мульти-TLAS подходами.

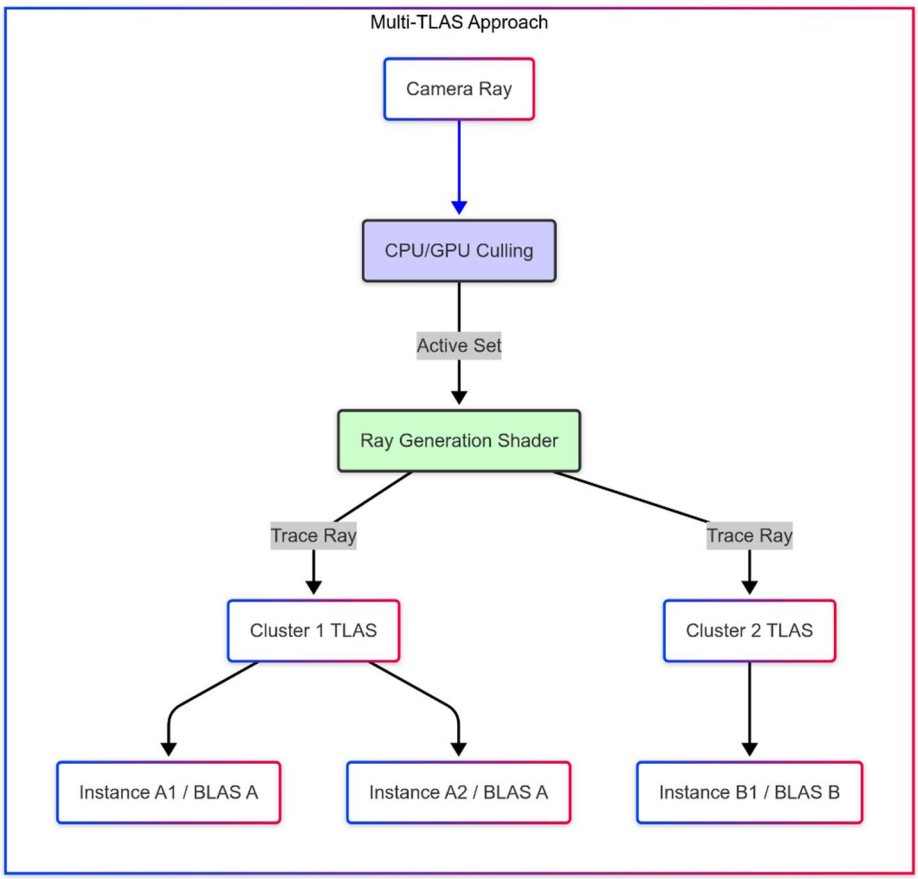

Рисунок 2 – Общее представление мульти-TLAS подхода

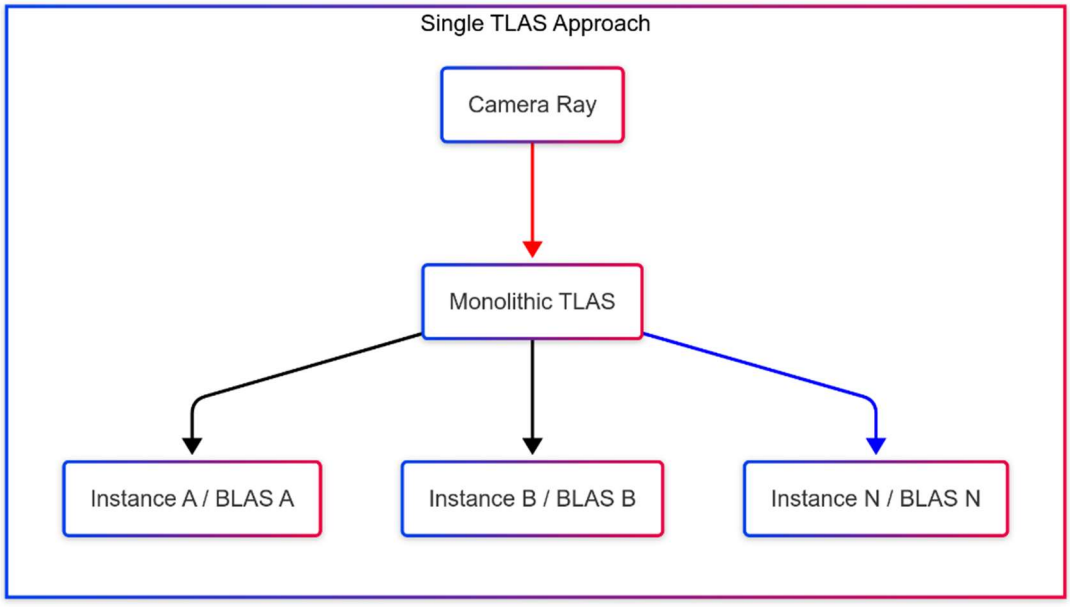

Рисунок 3 – Общее представление монолитного подхода

Использование множественных TLAS требует модификации конвейера трассировки лучей:

– Выбор TLAS для трассировки. Как упомянуто, основной механизм – передача списка активных TLAS (например, их индексов или дескрипторов через буфер хранения – storage buffer) в шейдер генерации лучей (RGen). Шейдер RGen затем инициирует трассировку (vkCmdTraceRaysKHR или traceRayEXT в GLSL) либо итеративно по списку активных TLAS, либо используя более сложную логику, основанную на направлении луча и пространственном расположении кластеров.

– Таблица Привязки Шейдеров (Shader Binding Table – SBT). Структура SBT может оставаться

относительно неизменной, но необходимо правильно вычислять смещения для доступа к записям шейдеров (Hit Group, Miss). Поскольку экземпляры в разных TLAS могут ссылаться на одни и те же BLAS и использовать одни и те же шейдеры, важно, чтобы индексы экземпляров (InstanceID или InstanceCustomIndexKHR) корректно отображались на глобальные идентификаторы материалов или объектов для выборки нужных данных и вызова правильных шейдеров из SBT. Поле InstanceCustomIndexKHR в структуре VkAccelerationStructureInstanceKHR может быть использовано для хранения уникального идентификатора объекта или материала независимо от того, в какой TLAS он находится.

Несмотря на очевидные преимущества, подход с множественными TLAS не лишен недостатков и требует тщательного рассмотрения:

– Увеличение сложности управления. Управление жизненным циклом, памятью и обновлениями множества TLAS сложнее, чем для одной структуры. Требуется надежная система управления кластерами и их синхронизации.

– Накладные расходы на выбор TLAS. Логика определения релевантных TLAS (как на CPU, так и потенциально в RGen шейдере) вносит дополнительные вычислительные расходы. Необходимо найти баланс между гранулярностью разбиения и этими расходами. Слишком мелкие кластеры могут привести к тому, что накладные расходы на управление и выбор превысят выгоду от уменьшения размера TLAS.

– Обработка границ кластеров. Объекты, находящиеся на границах кластеров, могут потребовать специальной обработки: либо включения в несколько TLAS (что приводит к дублированию данных экземпляров), либо реализации механизма передачи лучей между соседними TLAS.

– Потребление памяти. Хотя каждая отдельная TLAS меньше, суммарный объем памяти, занимаемый всеми TLAS и их внутренними узлами, может быть несколько больше, чем у одной оптимизированной монолитной TLAS, из-за некоторой избыточности в узлах верхнего уровня и потенциального дублирования экземпляров на границах.

Разделение единой структуры ускорения верхнего уровня на множество кластеризованных TLAS в рамках Vulkan Ray Tracing представляет собой мощный и перспективный метод оптимизации рендеринга методом трассировки пути, особенно для крупномасштабных, динамичных сцен с «мега-геометрией». Потенциальные выгоды от снижения затрат на построение и обновление AS, улучшения локальности данных при обходе и возможности эффективного гранулярного отсечения геометрии могут значительно повысить общую производительность и приблизить возможность использования полноценной трассировки пути в реальном времени для визуализации сцен нового поколения.

Данный подход хорошо синергирует с другими техниками оптимизации, такими как использование уровней детализации (LOD), карт непрозрачности (Opacity Micromaps – OMM) и карт смещения (Displacement Micromeshes – DMM) [1], позволяя более гибко управлять структурами ускорения для различных представлений геометрии внутри кластеров. Улучшение базовой производительности трассировки также положительно сказывается на техниках глобального освещения, использующих эти лучи, например, Neural Radiance Cache (NRC) из RTXGI [4].

Несмотря на возрастающую сложность реализации и управления ресурсами, ожидаемый выигрыш в производительности оправдывает применение данного метода в требовательных графических приложениях. Дальнейшие исследования могут быть направлены на:

– Разработку адаптивных и динамических стратегий кластеризации, изменяющих структуру кластеров во время выполнения в зависимости от положения камеры и распределения динамических объектов.

– Исследование гибридных подходов, сочетающих множественные TLAS для удаленных или статичных частей сцены с единой TLAS для близлежащих или высокодинамичных областей.

– Анализ влияния различных стратегий выбора и обхода активных TLAS в шейдерах на итоговую производительность и разработку специализированных алгоритмов для минимизации накладных расходов.

– Оптимизацию управления памятью для минимизации избыточности при использовании множественных TLAS.

Другим перспективным направлением развития подхода с мульти-TLAS является интеграция с системами управления видимостью на уровне движка сцены. Путём использования информации из графа сцены и систем пространственного индексирования (например, BVH, Octree, зон видимости порталов и комнат), можно заранее определить минимально необходимое множество TLAS для обработки текущего кадра. Это позволяет дополнительно сократить объём данных, передаваемых на GPU, и уменьшить нагрузку на трассировку. Интеграция мульти-TLAS в движки рендеринга требует координации между CPU и GPU, особенно в аспектах синхронизации буферов, управления состоянием объектов и обновления структур ускорения в реальном времени.

В будущем интерес может представлять реализация автоматических систем кластеризации с применением алгоритмов машинного обучения. Такие системы смогут адаптивно формировать кластеры на основе анализа предыдущих кадров, предсказывать области интереса камеры или игрока

и, соответственно, заранее подготавливать необходимые TLAS. Это особенно актуально для игр с открытым миром, где высокая степень свободы перемещения пользователя делает традиционные методы предвыборки данных неэффективными. Исследование таких интеллектуальных систем управления сценами может привести к значительным улучшениям как в производительности рендеринга, так и в экономии ресурсов GPU.

Список использованных источников:

1. Handling Billions of Triangles: NVIDIA RTX Mega Geometry Now Available with New Vulkan Samples // NVIDIA Developer Blog. – 2024 [Электронный ресурс]. – Режим доступа: https://developer.nvidia.com/blog/nvidia-rtx-mega-geometry-now-available-with-new-vulkan-samples/ – Дата доступа: 02.04.2025.

2. Vulkan Ray Tracing Specification. Khronos Group. [Электронный ресурс]. – Режим доступа: https://registry.khronos.org/vulkan/specs/1.3-extensions/man/html/VK_KHR_ray_tracing_pipeline.html – Дата доступа: 30.03.2025.

3. Kajiya, J. T. The rendering equation // ACM SIGGRAPH Computer Graphics. – 1986. – Vol. 20. – №. 4. – Pp. 143-150.

4. Majercik, A., Crassin, C., Shirley, P., McGuire, M., & Luebke, D. Neural Radiance Cache for Real-Time Global Illumination // NVIDIA Technical Brief. – 2021 [Электронный ресурс]. – Режим доступа: https://github.com/NVIDIA-RTX/RTXGI/blob/main/Docs/NrcGuide.md – Дата доступа: 03.04.2025.

5. Wald, I., Ize, T., Kensler, A., Knoll, A., & Parker, S. Ray tracing deformable scenes using dynamic bounding volume hierarchies // ACM Transactions on Graphics (TOG). – 2007. – Vol. 26. – №. 1. – P. 6.

UDC 004.925.3

OPTIMIZATION OF PATH TRACING IN VULKAN VIA SPATIAL GEOMETRY CLUSTERING AND TLAS SPLITTING

Akimenko A.V.

Belarusian State University of Informatics and Radioelectronics, Minsk, Republic of Belarus

Yarmolik V.N. – Doctor of Technical Sciences, professor

Annotation. This arcticle investigates a performance optimization method for path tracing of large-scale 3D scenes using the Vulkan graphics API and its ray tracing extensions (VK_KHR_ray_tracing_pipeline). An approach based on spatial clustering of scene geometry and building multiple top-level acceleration structures (TLAS) instead of a single monolithic one is proposed. The advantages of this method are analyzed in terms of reducing TLAS build/update costs, improving data locality during traversal, and enabling efficient culling of invisible scene parts. Implementation aspects in Vulkan and the potential impact on overall rendering performance are discussed.

Keywords. Path tracing, Vulkan, ray tracing, acceleration structures, TLAS, BLAS, rendering, 3D graphics, GPU optimization, large-scale scenes, geometry clustering, culling, VK_KHR_ray_tracing_pipeline, VK_KHR_acceleration_structure, GPU memory management.