DOI: 10.15514/ISPRAS–2020–32(4)–4

Модель производительности графического конвейера для однопроходной схемы рендеринга динамических трехмерных сцен

В.И. Гонахчян, ORCID: 0000-0002-4348-8443 <pusheax@ispras.ru>
Институт системного программирования им. В.П. Иванникова РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. В работе рассматривается однопроходная схема рендеринга динамических трехмерных сцен с использованием современных видеокарт (GPU) и графических интерфейсов. В рамках этой схемы используются следующие методы и техники: отсечение объектов с использованием методов пространственной декомпозиции и индексирования, аппаратные проверки видимости, фрагментация и кэширование командных буферов. Для выполнения этих методов требуются значительные вычислительные ресурсы, а объем работы на этапах графического конвейера зависит от их результатов. Поэтому важно сбалансированное использование ресурсов при конвейерной обработке и передаче графических данных. Предлагается модель производительности графического конвейера применительно к задачам рендеринга динамических трехмерных сцен, позволяющая оценивать требуемые ресурсы в зависимости от применяемых базовых методов и характеристик отображаемой сцены. В отличие от существующих методов и моделей, предлагаемая модель позволяет рассчитать затраты на составление буферов команд с использованием различных техник записи, затраты на отправку, выполнение, получение результатов аппаратных проверок видимости. Выводятся формулы для расчета временных затрат в зависимости от количества проверок видимости. Предлагается метод оценки количества аппаратных проверок видимости для эффективного выполнения рендеринга динамических сцен. Проводятся вычислительные эксперименты, показывающие релевантность предложенной модели и эффективность разработанного метода при отображении больших динамических сцен.

Ключевые слова: рендеринг; составление командных буферов; удаление невидимых поверхностей; аппаратные проверки видимости; модель производительности графического конвейера

Для цитирования: Гонахчян В.И. Модель производительности графического конвейера для однопроходной схемы рендеринга динамических трехмерных сцен. Труды ИСП РАН, том 32, вып. 4, 2020 г., стр. 53–72. DOI: 10.15514/ISPRAS–2020–32(4)–4.

Performance model of graphics pipeline for single-pass dynamic 3d scene rendering scheme

V.I. Gonakhchyan, ORCID: 0000-0002-4348-8443 <pusheax@ispras.ru>
Ivannikov Institute for System Programming of the RAS,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.

Abstract. The paper considers a single-pass scheme for rendering dynamic 3D scenes using modern GPUs and graphics interfaces. The following methods and techniques are used with this scheme: clipping of objects using spatial decomposition and indexing methods, hardware occlusion queries, fragmentation and caching of command buffers. These methods require significant computational resources to execute, and the amount of work in the stages of the graphics pipeline depends on their results. Therefore, a balanced use of resources when transferring graphics data and executing commands is important. A performance model of the graphics pipeline

is proposed, which makes it possible to estimate the required resources depending on the applied base methods and characteristics of the displayed scene. Unlike existing methods and models, the proposed model allows to calculate the costs of composing command buffers using various recording techniques, the costs of sending, executing, and receiving the results of hardware occlusion queries. Formulas are derived to calculate frame rendering time depending on the number of occlusion queries. A method is proposed to estimate the number of occlusion queries for efficient rendering of dynamic scenes. Computational experiments are carried out to show the relevance of the proposed model and the effectiveness of the developed method when displaying large dynamic scenes. Section 1 provides an overview of related work as well as general purposes of given paper and its structure. Section 2 describes the proposed performance model and method used to calculate the number of occlusion queries for efficient rendering of dynamic scenes. Section 3 presents the performance analysis, which contains derived and measured rendering time when rendering scenes and employing frustum culling, occlusion queries. Section 4 summarizes the main conclusions.

Keywords: 3d rendering; command buffer recording; occlusion culling; hardware occlusion queries; performance model of graphics pipeline.

For citation: Gonakhchian V.I. Performance model of graphics pipeline for single-pass dynamic 3d scene rendering scheme. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 4, 2020. pp. 53–72 (in Russian). DOI: 10.15514/ISPRAS–2020–32(4)–4

1. Введение

Рендеринг трехмерных сцен является одной из ключевых задач компьютерной графики и широко применяется в таких предметных областях, как научная визуализация, автоматизация проектирования, инженерии и производства (CAD/CAM/CAE), компьютерные игры и анимация, виртуальная и дополненная реальность. В связи с перманентным ростом сложности сцен повышаются требования и к эффективности программных и аппаратных средств рендеринга. Несмотря на обширные исследования в этой области и большое количество опубликованных работ, задачам рендеринга больших динамических сцен уделяется относительно мало внимания. Вместе с тем, класс приложений, оперирующих с подобными сценами, чрезвычайно широк, что определяет актуальность темы исследований. Особую важность тема приобретает в связи со стремительным развитием технологий информационного моделирования зданий и сооружений (BIM), предусматривающих, в частности, интерактивные графические средства динамического моделирования процессов возведения сложных строительных объектов и реализации масштабных инфраструктурных программ.

В работе рассматриваются задачи отображения динамических трехмерных сцен с использованием современных видеокарт (GPU) и графических интерфейсов к ним (OpenGL, DirectX, Vulkan). Видеокарты обеспечивают высокоэффективную поточно-параллельную обработку графических элементов (треугольников, пикселей) для достижения требуемой частоты генерации изображений. Графические интерфейсы позволяют задать описание трехмерной сцены и передать его на видеокарту, установить программы для геометрических преобразований, проецирования треугольников на экранную плоскость, расчета функций освещения, а также сгенерировать итоговые изображения.

Однако при отображении сложных сцен, содержащих большое количество графических элементов и предполагающих использование сложных моделей материалов и освещения, видеокарты часто не справляются с большим объемом вычислений. Подобные ограничения делают невозможными разработку и применение интерактивных графических приложений, предполагающих генерацию и вывод изображений с частотой, соответствующей скорости отклика на пользовательские события.

Одним из традиционных подходов к повышению производительности рендеринга является уменьшение числа растеризуемых графических элементов за счет предварительного анализа и удаления невидимых поверхностей. Распространенные варианты методов удаления невидимых поверхностей описаны в работах [1-5]. В методах с использованием

потенциально видимых множеств объектов, порталов, расширенных проекций (Potentially visible sets, Portals and mirrors, Extended projections) выполняется предварительный анализ видимости элементов из разных пространственных ячеек. При изменении положения камеры и смене занимаемой ячейки уточняется множество видимых элементов. Данные методы эффективны для статических сцен, в которых статус видимости объектов не меняется в зависимости от происходящих в сцене событий и может быть вычислен заранее.

Метод иерархического буфера глубины (Hierarchical z-buffer) [6], метод исключения на основе бинарного пространственного разбиения (BSP tree culling) [7] отбирают объекты сцены, невидимые из заданного положения камеры, с помощью специальных пространственных структур данных. Проверки видимости, выполняясь на центральном процессоре (CPU), позволяют составить буфер команд только для видимых объектов и, тем самым, уменьшить нагрузку на видеокарту. Ограниченные вычислительные ресурсы CPU являются наиболее узким местом для подобных реализаций, обычно проявляющимся при отображении больших сцен.

Методы, описанные в работах [8-10], выполняют проверки видимости с учетом имеющихся в сцене больших объектов-преград (large occluders). На этапе предобработки подобные преграды, если присутствуют в сцене, выявляются. На этапе отображения определяется статус перекрытий объектов, для чего они сортируются в порядке удаленности от камеры. Методы имеют отмеченные ограничения, а анализ видимости объектов с учетом множественных преград (occluder fusion) может требовать значительных вычислительных ресурсов.

Одним из общих подходов к ускорению методов удаления невидимых элементов является использование структур пространственной декомпозиции сцены, таких как окто-деревья и k-d-деревья [11-12]. Для этих же целей используются иерархии ограничивающих объемов BVH (Bounding Volume Hierarchies), основанные на произвольно ориентированных и ориентированных вдоль координатных осей параллелепипедах OBB (Oriented Bounding Box), AABB (Axis Aligned Bounding Box), а также на многогранниках с фиксированным количеством ориентаций граней k-DOP. Структуры декомпозиции сцены и объектов, выполняющие функции пространственных индексов, позволяют существенно сократить объем вычислений за счет быстрого вынесения отрицательного вердикта относительно видимости объектов, что достигается за счет недорогих, иерархически организованных проверок видимости ограничивающих их пространственных ячеек или объемов.

Важным достоинством методов пространственной декомпозиции и индексирования является применимость к динамическим сценам, хотя в этом случае необходимо эффективно обновлять соответствующие пространственные структуры при наступлении соответствующих изменений в сцене. Возможные способы решения этой проблемы предложены в работах [13-14]. Метод временного ограничивающего объема (Temporal Bounding Volume) вычисляет результирующий параллелепипед, ограничивающий объект с учетом всех его положений на траектории движения, и подготавливает необходимые структуры для эффективного рендеринга сцены с удалением невидимых элементов. Метод успешно применяется для сцен с детерминированным характером динамики, однако имеет ограничения для сцен, в которых события происходят случайно или моделируются специальными условиями.

В современных видеокартах реализован функционал для выполнения аппаратных проверок видимости (hardware occlusion queries) [15]. В ряде случаев он позволяет повысить эффективность отображения сцен, однако регулярное использование аппаратных проверок может приводить к деградации производительности, принимая во внимание дополнительные расходы на подготовку и осуществление самих проверок. Поэтому необходимыми становятся оценки эффективности применения аппаратных проверок с учетом вычислительных затрат и изменений в сцене. В частности, такие оценки могут применяться для отложенных проверок

видимости для элементов структур пространственной декомпозиции, описанных в работах [16-17].

Важными факторами, влияющими на эффективность процесса рендеринга динамических сцен, являются способы составления и отправки командных буферов [19]. Как правило, описание объектов сцены формируется в виде буферов команд рендеринга, которые записываются в основную память один раз и отправляются на видеокарту по шине. Однако при отображении динамических сцен и использовании проверок видимости требуется повторная запись, которая может занимать продолжительное время. При незначительных локальных изменениях на каждом временном шаге моделирования динамической сцены можно сократить затраты на составление и отправку буферов за счет их фрагментации и кэширования в основной памяти. В этом случае требуется обновление только тех буферов, объекты которых подверглись изменениям на текущем временном шаге.

Таким образом, для эффективного рендеринга больших динамических сцен с использованием современных видеокарт перспективным представляется применение следующих методов и техник:

- удаление невидимых объектов с использованием методов пространственной декомпозиции и индексирования;
- аппаратные проверки видимости с учетом временной когерентности статуса видимости;
- фрагментация и кэширование командных буферов с учетом локальных изменений в сцене.

Поскольку данные методы выполняются одновременно в вычислительной системе, важно сбалансированное использование ее ресурсов при конвейерной обработке и передаче графических данных. В частности, должна быть сбалансирована загрузка CPU и GPU процессоров. Вместе с тем, вариативность в количестве и сложности индивидуальных объектов сцены, характере и интенсивности динамики делает подобную балансировку крайне сложной или даже невозможной при фиксировании конкретных методов и техник. Обеспечить высокую производительность вычислительной системы на широком классе задач рендеринга больших динамических сцен представляется возможным в результате адаптивного управления графическим конвейером, предусматривающего выбор и настройку альтернативных базовых методов и техник с учетом доступных ресурсов системы и особенностей отображаемой сцены на конкретном интервале модельного времени в конкретной пространственной области. Требуется разработать модель производительности графического конвейера для оценки затрачиваемых ресурсов при выполнении рендеринга динамических сцен с использованием описанных методов.

Рассмотрим существующие работы, применяющие оценки вычислительных затрат в процессе рендеринга. В работе [20] предложен адаптивный метод рендеринга с заданной частотой кадров. Отображение каждого объекта выполняется с подходящим уровнем детализации (level of detail) для поддержания целевой частоты кадров. Как правило, уровень детализации выбирается на основе расстояния до объекта и размера объекта, но это не всегда дает равномерную частоту кадров. Частота кадров может сильно меняться, когда появляются новые объекты или изменяются их уровни детализации. Для устранения этого недостатка можно адаптивно изменять пороговые размеры для вывода уровней детализации. Эта техника хорошо справляется со сценами, в которых видимость объектов слабо меняется при переходе к следующему кадру. В некоторых сценах количество объектов значительно меняется при движении камеры. Для таких сцен требуется выполнить предварительную оценку затрат на отображение. Исходя из этой оценки, можно выбрать подходящие уровни детализации объектов. Для каждого объекта осуществляется выбор уровня детализации и алгоритма затенения (равномерное затенение, затенение по Гуро, использование текстурных карт). Задача сводится к поиску значений, которые осуществимы с целевой частотой кадров и соответствуют максимальному качеству изображения. Рассматривается упрощенная модель

графического конвейера, которая включает в себя обработку вершин, полигонов и фрагментов. Предполагается, что отправка команд рендеринга происходит мгновенно, на GPU процессоре выполняется только рассматриваемое приложение. Время работы конвейера определяется временем самого медленного этапа. Для оценки времени обработки графических элементов предлагается использовать линейную комбинацию количества полигонов и вершин. Получаем формулу:

$$Cost(O, L, R) = max\begin{pmatrix} C_1 Poly(O, L) + C_2 Vert(O, L) \\ C_3 Pix(O) \end{pmatrix},$$

где O — объект,

L — уровень детализации объекта,

$Poly(O, L)$ — количество полигонов объекта O,

$Vert(O, L)$ — количество вершин объекта O,

C_1, C_2, C_3 — константы рендеринга, которые предлагается определять экспериментальным путем.

В работе [21] предложена модель производительности для определения времени рендеринга трехмерной сцены. Рассматривается общая формула для вычисления времени рендеринга:

$$t = RT(SG, RA, HW, ST),$$

где SG — обход объектов сцены (scene graph),

RA — процедура по отображению объекта (rendering action),

HW — аппаратное обеспечение (hardware),

ST — текущее состояние операционной системы и аппаратного обеспечения (state).

Для упрощения этой формулы вводится обозначение для массива объектов $X = (x_1, \dots, x_n)$. Считается, что каждый объект описывается своими атрибутами и геометрическим представлением $x_i = (g_i, a_i)$. Предполагается, что процедура по отображению объекта определяется его атрибутами. Тогда формула выглядит следующим образом: $t = RT(X, HW, ST)$. Для каждого кадра выполняется обход объектов сцены X, установка состояния GPU процессора согласно атрибутам a_i, отправка геометрии g_i. Драйвер отправляет составленные команды в командный буфер (очередь FIFO), видеокарта читает отправленные команды и выполняет этапы конвейера GPU для преобразования, растеризации, вычисления цвета и вывода на экран треугольников объектов. Графические элементы состоят из вершин и индексов, геометрическое представление хранится либо в памяти AGP, либо в видеопамяти. Предлагаются следующие формулы для оценки временных затрат CPU и GPU процессоров:

$$RT = ET_{system} + \max(ET^{CPU}, ET^{GPU}),$$
$$ET^{CPU} = ET^{CPU}_{nr} + ET^{CPU}_{r} + ET^{CPU}_{mm} + ET^{CPU}_{idle},$$
$$ET^{GPU} = ET^{GPU}_{fs} + ET^{GPU}_{r} + ET^{GPU}_{mm} + ET^{GPU}_{idle},$$

где nr — затраты, не относящиеся к рендерингу,

r — затраты на рендеринг,

fs — затраты на установку состояния,

mm — затраты на обращения к памяти,

$idle$ — время простоя.

В наилучшем сценарии, CPU и GPU процессоры работают параллельно без простоев (с загруженностью 100%). Зачастую этот сценарий является недостижимым. Задержка при выполнении обхода сцены на центральном процессоре вызывает простой в ожидании команд. Долгая обработка элементов на конвейере GPU вызывает простой центрального процессора. Для достижения наилучшей производительности необходимо сократить время простоя. Используется следующий метод оценки времени рендеринга. Пусть задано количество

вершин, треугольников, пикселей. Тогда время рендеринга вычисляется как максимум времени преобразования вершин, растеризации треугольников, вычисления цвета пикселей. Для получения консервативной оценки предлагается суммировать времена работы разных этапов конвейера.

2. Модель производительности графического конвейера

В данном разделе предлагается модель производительности графического конвейера, которая расширяет существующие модели, описанные в работах [20-21]. Новизна заключается в том, что учитываются следующие аспекты рендеринга:

- затраты на запись и отправку командных буферов, в том числе с использованием рассматриваемых техник;
- затраты на отправку, выполнение, получение результатов аппаратных проверок видимости.

Выводятся формулы для расчета времени рендеринга использованием альтернативных базовых методов и техник:

- удаление невидимых объектов с использованием методов пространственной декомпозиции и индексирования;
- отложенные аппаратные проверки видимости;
- фрагментация и кэширование командных буферов с учетом локальных изменений в сцене.

Полученные формулы позволяют своевременно выбирать наиболее эффективные способы реализации рендеринга в рамках однопроходной схемы в процессе отображения динамических сцен.

2.1 Исследуемая модель графического конвейера

Рассмотрим модель графического конвейера, которая описывает рендеринг за один проход по объектам и графическим элементам. На рис. 1 изображены основные этапы рендеринга, которые включают обработку объектов на CPU и GPU процессорах.

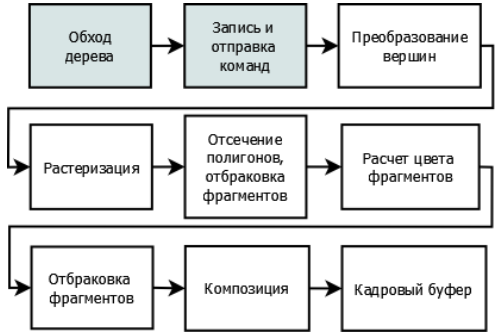

Рис. 1. Основные этапы рендеринга объектов сцены. Первые два этапа выполняются на центральном процессоре. Остальные этапы выполняются на GPU процессоре
Fig. 1. Main stages of 3D rendering. First two stages execute on CPU. Other stages execute on GPU

На вход подается сцена, включающая множество объектов, которые представлены в виде треугольных сеток с заданным положением и материалом. Применяется структура пространственного разбиения для сокращения затрат при выполнении отсечений (frustum culling) и аппаратных проверок видимости (hardware occlusion queries). На первом этапе

выполняется отбор узлов структуры, которые попадают в область видимости камеры. Затем на центральном процессоре проводится запись и отправка команд рендеринга. Команда рендеринга содержит информацию о конфигурации конвейера GPU процессора и смещение в памяти, по которому хранятся вершины (индексы вершин) объекта. Команды посылаются по шине на GPU процессор.

2.2 Исследуемая схема рендеринга

В данной работе рассматривается прямая однопроходная схема рендеринга динамических трехмерных сцен. Это означает, что записывается одна команда рендеринга для каждого объекта сцены, командный буфер составляется на центральном процессоре, осуществляется один проход по графическим элементам для генерации кадрового буфера. Рассматриваемые способы реализации процессов рендеринга заключаются в выполнении следующих шагов:

1. Изменение видимости, положений, материалов объектов сцены при изменении анимационного времени.
2. Получение результатов аппаратных проверок видимости.
3. Ожидание готовности командного буфера.
4. Обход структуры пространственного разбиения и выполнение отсечений.
5. Запись команд рендеринга для отдельных объектов сцены.
6. Запись ссылок на вспомогательные буфера в главный командный буфер.
7. Запись буфера для выполнения аппаратных проверок видимости.
8. Отправка главного командного буфера на выполнение.

Шаги 1, 3, 8 выполняются для всех способов. Запись команд рендеринга может проводиться напрямую в главный командный буфер (шаг 5) или может выполняться запись командных буферов для групп объектов сцены (шаг 6). Для ускорения рендеринга может использоваться структура пространственного разбиения H (шаг 4), могут выполняться аппаратные проверки видимости (шаги 2, 7).

2.3 Описание динамической сцены

Пусть кортеж S содержит описание динамической сцены:

$$S =< Mesh, MeshInst, Mat, Tex, Idx, Vert, TC, Norm, Vp, t_{anim}, TrnTl, VisTl, MatTl, batchSize, visFrags, d_{anim} >, \quad (1)$$

где $Mesh$ — множество треугольных сеток,

$MeshInst$ — множество объектов сцены, которые имеют заданное положение, материал, видимость,

Mat — множество материалов,

Tex — множество текстур,

Idx — множество индексов, которое задается в случае использования проиндексированных множеств вершин, нормалей, текстурных координат,

$Vert$ — множество вершин,

TC — множество текстурных координат,

$Norm$ — множество нормалей,

Vp — текущее положение камеры (viewpoint), которое задается с помощью матрицы размерности 4×4,

t_{anim} — время анимации, для которого определяются характеристики объектов сцены,

$TrnTl$ — множество временных событий, которые задают положение объектов сцены,

$VisTl$ — множество временных событий, которые задают видимость объектов сцены,

$MatTl$ — множество временных событий, которые задают материалы объектов сцены.

Также в кортеж S добавлены переменные, которые зависят от положения камеры Vp и анимационного времени t_{anim}:

$batchSize$ — среднее количество вершин в видимых объектах сцены во время t_{anim},

$visFrags$ — суммарное количество фрагментов видимых объектов сцены во время t_{anim},

d_{anim} — средняя доля объектов сцены, свойства которых изменились за последние несколько кадров.

Кортеж S задает описание всех объектов сцены (видимых и невидимых). Для сцен с объектами, прошедшими проверки видимости, в момент времени t_{anim} при положении камеры Vp будем использовать обозначение S_{vis}, если не оговорено иначе. Под проверками видимости подразумевается метод удаления объектов, не попадающих в область видимости камеры, и метод выполнения аппаратных проверок видимости ограничивающих параллелепипедов AABB узлов дерева относительно буфера глубины (hardware occlusion query).

2.4 Оценка потребляемой памяти

Данные для выполнения рендеринга записываются в память. При использовании разных способов рендеринга расходуется различный объем памяти. Предварительная запись командных буферов, использование дополнительных уровней детализации объектов (levels of details) повышают объем расходуемой памяти. Если памяти не хватает, то, как правило, выполняется завершение программы. Чтобы убедиться, что памяти достаточно для выполнения рендеринга, выведем формулы для вычисления требуемого объема памяти. Объем памяти, который занимает полигональное представление сцены, выражается с помощью формулы:

$$M_{geom}(S, HW) = |\mathrm{Vert}|\mathrm{M}(vert_1, HW) + |\mathrm{Idx}|\mathrm{M}(idx_1, HW) + |\mathrm{TC}|\mathrm{M}(tc_1, HW) + |\mathrm{Norm}|\mathrm{M}(norm_1, HW), \quad (2)$$

где HW — конфигурация вычислительной системы (CPU и GPU),

$|Vert|$, $|Idx|$, $|TC|$, $|Norm|$ — количества вершин, индексов, текстурных координат, нормалей,

$M(a, HW)$ — объем памяти, который занимает элемент a.

Объем памяти, который занимают текстуры сцены, выражается с помощью формулы:

$$M_{tex}(\mathrm{S}, \mathrm{HW}) = \sum_{i=1}^{|Tex|} W(tex_i)H(tex_i)BPP(tex_i, HW), \quad (3)$$

где $W(tex)$ — ширина текстуры tex,

$H(tex)$ — высота текстуры tex,

$BPP(tex)$ — объем памяти, который занимает один пиксель текстуры tex.

Объем памяти, который занимают буфера рендеринга с матрицами и материалами, выражается с помощью формулы:

$$M_{uniforms}(\mathrm{S}, \mathrm{HW}) = \sum_{i=1}^{|MeshInst|} M_{ubo}(Mat(inst_i), HW), \quad (4)$$

где $M_{ubo}(mat, HW)$ — объем памяти, который занимает буфер, хранящий значения переменных для шейдера, соответствующего данному материалу,

$Mat(inst)$ — материал объекта сцены $inst$.

В процессе рендеринга записываются буфера с командами рендеринга. Каждая команда содержит информацию для отображения объекта сцены с заданным положением и материалом. Объем памяти, который занимает буфер, содержащий команды для рендеринга объектов сцены, выражается с помощью формулы:

$$M_{cmds}(\mathrm{S},\mathrm{HW}) = \sum_{i=1}^{|MeshInst|} M_{cmd}(Mat(inst_i), HW), \quad (5)$$

где $M_{cmd}(mat, HW)$ — объем памяти, который занимает команда рендеринга объекта сцены с материалом mat.

Пусть множество объектов сцены S разбито на подмножества $\mathrm{G} = \{g_i : \{inst_{i_1}, \dots, inst_{i_n}\}\}$. Будем считать, что для каждого подмножества g_i записывается отдельный командный буфер (secondary command buffer). Минимальный объем памяти, который занимает командный буфер, составляет $M_{\mathrm{sec\_buf}}(HW)$. Команды рендеринга подмножества g_i могут занимать больший объем памяти. В этом случае производится расширение командного буфера. Объем памяти, который занимают буфера, содержащие команды рендеринга для подмножеств g_i, выражается с помощью формулы:

$$M_{grouped\_cmds}(\mathrm{G},\mathrm{HW}) = \sum_{i=1}^{|G|} \max\left(M_{\mathrm{sec}\_buf}(HW), M_{cmds}(g_i,\mathrm{HW})\right). \quad (6)$$

Для выполнения рендеринга необходимо задать конфигурацию графического конвейера, которая используется при отображении объекта сцены с заданным материалом. Зачастую используется одна конфигурация конвейера на материал. Конфигурации всех конвейеров, как правило, составляются заранее и хранятся в памяти для сокращения затрат в процессе рендеринга. Объем памяти, который занимают конфигурации графических конвейеров программного интерфейса, выражается с помощью формулы:

$$M_{rendering}(\mathrm{S},\mathrm{HW}) = \sum_{i=1}^{|Mat|} \mathrm{M}_{pipeline}(m_i, HW), \quad (7)$$

где $\mathrm{M}_{pipeline}(m_i, HW)$ — объем памяти, который требуется для хранения описания конфигурации конвейера для вывода объектов сцены с материалом m_i.

Объем памяти, который требуется для хранения описаний объектов сцены, выражается с помощью формулы:

$$M_{instances}(\mathrm{S},\mathrm{HW}) = |MeshInst|M(inst_1, HW) + |Mat|M(mat_1, HW), \quad (8)$$

где $|MeshInst|$, $|Mat|$ — количества объектов сцены, материалов,

$M(inst, HW)$ — объем памяти, который занимает объект сцены при использовании вычислительной системы HW,

$M(mat, HW)$ — объем памяти, который занимает материал при использовании вычислительной системы HW.

При выполнении рендеринга может использоваться структура пространственного разбиения для кластеризации объектов сцены. Объем памяти, который требуется для хранения пространственного индекса, выражается с помощью формулы:

$$M_{hierarchy}(\mathrm{H},\mathrm{HW}) = |\mathrm{H}|M(n_1, HW), \quad (9)$$

где H — множество узлов структуры пространственного разбиения (дерева),

$M(n, HW)$ — объем памяти, который занимает узел структуры пространственного разбиения при использовании вычислительной системы HW.

Зачастую геометрия сцены, текстуры и буфера с матрицами и описанием материалов ($M_{geom}(S, HW)$, $M_{tex}(S, HW)$, $M_{uniforms}(S, HW)$) хранятся в видеопамяти. Остальные данные ($M_{cmds}(S, HW)$, $M_{grouped\_cmds}(G, HW)$, $M_{rendering}(S, HW)$, $M_{instances}(S, HW)$, $M_{hierarchy}(H, HW)$) хранятся в основной памяти (RAM). Когда недостаточно видеопамяти, можно выполнить запись в основную память. Таким образом, на этапе подготовки к рендерингу определяется доступная память и проводится запись данных либо в видеопамять, либо в основную память.

2.5 Оценка времени рендеринга

Графический конвейер выполняет работу для выявления треугольников, которые не попадают в область видимости камеры. Для каждой вершины треугольников выполняется преобразование для перевода в плоскость изображения. Только после растеризации выполняется отсечение треугольников, которые не попали в область видимости камеры. Таким образом, расходуются лишние вычислительные ресурсы на отправку команд и преобразование вершин треугольников, которые не выводятся на экран. Подобная ситуация возникает и при отображении сложных сцен с большим количеством перекрывающихся поверхностей. Для определения видимости фрагментов используется метод z-буфера. Однако до применения метода z-буфера необходимо выполнить преобразование вершин, растеризацию, расчет цвета фрагментов. Таким образом, часть ресурсов тратится на обработку фрагментов, которые отсекаются методом z-буфера и в конечном итоге не попадают в кадровый буфер. Эта проблема особенно проявляется в сценах с большим количеством перекрывающихся поверхностей. Объем вычислений можно сократить с помощью методов удаления невидимых поверхностей, но они не всегда эффективны. Актуальной является разработка модели производительности рендеринга, в рамках которой можно оценить время выполнения различных этапов рендеринга. Это позволит определить сценарии, в которых методы удаления невидимых поверхностей являются эффективными.

Рассмотрим формулы для оценки времени выполнения приведенных способов реализации процессов рендеринга. Время выполнения рендеринга на центральном процессоре выражается с помощью формулы:

$$\begin{aligned} T_{CPU}(S, S_{vis}, H, H_{vis}, H_{upd}, H_q, h, g, q, HW) = \\ T_{update}(S, HW) + [h = 1]T_{update}(H, HW) + \\ [h = 0]T_{fc}(S, HW) + [h = 1](T_{traverse}(H, HW) + T_{fc}(H, HW)) + \\ [g = 0]T_{buf}(S_{vis}, HW) + [g = 1](T_{buf}(H_{upd}, HW) + T_{\mathrm{sec\_buf}}(H_{vis}, HW)) + \\ [q = 1](T_{buf}(H_q, HW) + T_{recv\_query}(H_q, HW)), \end{aligned} \quad (10)$$

где S_{vis} — кортеж с информацией об объектах сцены, которые прошли проверки видимости и находятся в области видимости камеры,

H_{vis} — множество узлов структуры пространственного разбиения в области видимости камеры,

H_{upd} — множество узлов структуры пространственного разбиения, в которых произошли изменения после движения объектов сцены,

H_q — множество узлов, для которых выполняются аппаратные проверки видимости,

q — параметр, который определяет, используются ли проверки видимости,

$[q = 1]$ — выражение, которое равняется 1, если $q = 1$ (скобка Айверсона),

h — параметр, который определяет, используется ли структура пространственного разбиения H,

g — параметр, который определяет, используется ли составление и отправка вспомогательных буферов (secondary command buffers) для узлов структуры пространственного разбиения H,

$T_{update}(S, HW)$ — время изменения свойств объектов сцены с помощью структур $TrnTl$, $VisTl$, $MatTl$ при изменении анимационного времени,

$T_{update}(H, HW)$ — время обновления пространственного индекса при изменении анимационного времени,

$T_{fc}(H, HW)$ — время выполнения отсечений узлов, не попадающих в область видимости камеры,

$T_{traverse}(H, HW)$ — время обхода структуры пространственного разбиения H,

$T_{buf}(S, HW)$ — суммарное время составления и отправки командных буферов со всеми объектами сцены S,

$T_{\text{sec\_buf}}(H, HW)$ — суммарное время отправки вспомогательных командных буферов для узлов структуры пространственного разбиения H,

$T_{recv\_query}(H, HW)$ — время получения всех результатов проверок видимости.

Время выполнения рендеринга на GPU процессоре выражается с помощью формулы:

$$T_{GPU}(S, H_q, q, HW) = \max\left(T_{state}(S, HW), T_{vert}(S, HW), T_{frag}(S, Vp, HW)\right) + [q = 1]\max\left(T_{state}(H_q, HW), T_{vert}(H_q, HW), T_{frag}(H_q, Vp, HW), T_{exec\_query}(H_q, HW)\right), \quad (11)$$

где $T_{state}(S, HW)$ — суммарное время установки состояний конвейера для всех материалов, используемых при отображении сцены S,

$T_{vert}(S, HW)$ — суммарное время преобразования вершин объектов сцены S,

$T_{frag}(S, Vp, HW)$ — суммарное время обработки фрагментов объектов сцены S,

$T_{exec\_query}(H, HW)$ — время выполнения проверок видимости узлов структуры пространственного разбиения H.

Итоговое время рендеринга сцены S при использовании структуры пространственного разбиения H выражается с помощью формулы:

$$T(S, S_{vis}, H, H_{vis}, H_{upd}, H_q, h, g, q, HW) = \begin{cases} T_{CPU}(S, S_{vis}, H, H_{vis}, H_{upd}, H_q, h, g, q, HW) + T_{GPU}(S_{vis}, H_q, q, HW), \text{один гл. ком. буфер} \\ \max\left(T_{CPU}(S, S_{vis}, H, H_{vis}, H_{upd}, H_q, h, g, q, HW), T_{GPU}(S_{vis}, H_q, q, HW)\right), \text{иначе.} \end{cases} \quad (12)$$

В формуле (12) выполняется расчет времени рендеринга для двух сценариев. В первом сценарии используется один главный командный буфер (при этом может использоваться много вспомогательных буферов). Запись команд возможна только тогда, когда чтение буфера завершено на GPU процессоре. Это означает, что необходимо выполнять синхронизацию доступа к буферу, при этом работа на CPU и GPU будет выполняться последовательно. Во втором сценарии используется несколько командных буферов и выполняется параллельная работа над разными буферами (рис. 2).

Для вычисления затрат при использовании различных способов реализации процессов рендеринга в приведенные формулы добавлены параметры h, g, q. Если $h = 0, g = 0, q = 0$, получаем формулу для расчета времени рендеринга без использования структуры пространственного разбиения, без записи команд рендеринга для узлов дерева (фрагментации), без проверок видимости узлов дерева. Если $h = 1, g = 0, q = 0$, получаем формулу для расчета времени рендеринга с использованием структуры пространственного разбиения для выполнения отсечений, без записи команд рендеринга для узлов дерева, без проверок видимости узлов дерева. Если $h = 1, g = 1, q = 0$, получаем формулу для расчета времени рендеринга с использованием структуры пространственного разбиения для

выполнения отсечений, с составлением команд рендеринга для узлов дерева, без проверок видимости узлов дерева. Если $h = 1, g = 1, q = 1$, получаем формулу для расчета времени рендеринга с использованием структуры пространственного разбиения для выполнения отсечений, с составлением команд рендеринга для узлов дерева, с проверками видимости узлов дерева.

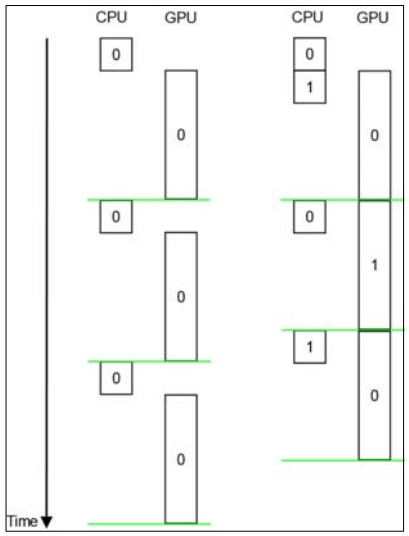

Рис. 2. Синхронизация доступа к главным командным буферам. Номерами помечены командные буфера. На CPU производится запись, на GPU — выполнение. Использование одного главного буфера (слева). Использование двух главных буферов для параллелизации записи и выполнения буфера команд (справа). Зеленая линия показывает момент, когда доступен для записи следующий командный буфер

Fig. 2. Synchronizing access to primary command buffers. Numbers are used to mark different command buffers. CPU performs writing, GPU — execution. Employing one primary command buffer (left). Employing two command buffers to parallelize writing and execution of commands (right). Green line shows the moment of time when next buffer is ready for writing

2.6 Время выполнения этапов графического конвейера

Оценим время выполнения различных этапов рендеринга. Время изменений свойств объектов сцены с помощью структур $TrnTl$, $VisTl$, $MatTl$ зависит от количества изменений. Время изменений вычисляется по формуле:

$$T_{update}(S, HW) = d_{anim}|MeshInst|t_{update}(HW), \quad (13)$$

где $t_{update}(HW)$ — среднее время записи свойств (матрицы преобразования, материала, видимости) для одного объекта сцены. Оно также включает время обращения к памяти UBO, в которой хранятся значения переменных шейдерных программ.

Время обхода структуры пространственного разбиения вычисляется по формуле:

$$T_{traverse}(H, HW) = |H|t_{traverse}(HW), \quad (14)$$

где $t_{traverse}(HW)$ — время обхода одного узла на данной вычислительной системе.

Время выполнения отсечений узлов структуры пространственного разбиения вычисляется по формуле:

$$T_{fc}(H, HW) = |H|t_{fc}(HW), \quad (15)$$

где $t_{fc}(HW)$ — время отсечения одного узла на данной вычислительной системе.

Время записи командного буфера для всех объектов сцены вычисляется по формуле:

$$T_{buf}(MeshInst, HW) = |MeshInst| t_{buf}(HW), \quad (16)$$

где $t_{buf}(HW)$ — время записи и отправки одной команды на данной вычислительной системе.

Время отправки вспомогательных командных буферов вычисляется по формуле:

$$T_{\text{sec\_buf}}(H, HW) = |H| t_{\text{sec\_buf}}(numInstPerNode, HW), \quad (17)$$

$t_{\text{sec\_buf}}(numInstPerNode, HW)$ — время отправки одного вспомогательного командного буфера с количеством команд $numInstPerNode$.

Времена выполнения этапов конвейера GPU вычисляются по формулам:

$$T_{state}(S, HW) = |Mat| t_{state}(HW), \quad (18)$$

$$T_{vert}(S, HW) = \sum_{i=1}^{|MeshInst|} NumVerts(inst_i) t_{vert}(Mat(inst_i), batchSize, |MeshInst|, HW), \quad (19)$$

$$T_{frag}(S, Vp, HW) = \sum_{i=1}^{|MeshInst|} NumFrags(inst_i, Vp) t_{frag}(Mat(inst_i), visFrags, HW), \quad (20)$$

$$T_{exec\_query}(H, HW) = |H| t_{exec\_query}(HW), \quad (21)$$

$$T_{recv\_query}(H, HW) = |H| t_{recv\_query}(HW), \quad (22)$$

где $T_{state}(HW)$ — время установки состояния графического конвейера на данной вычислительной системе,

$t_{vert}(mat, batchSize, numObjects, HW)$ — время работы вершинного шейдера данного материала с данным количеством вершин в объекте $batchSize$ и количеством объектов в сцене $numObjects$ (важность этих параметров объясняется ниже),

$NumVerts(inst)$ — количество вершин в объекте $inst$,

$t_{frag}(mat, visFrags, HW)$ — время работы фрагментного шейдера данного материала при обработке одного фрагмента (важность параметра $visFrags$ объясняется ниже),

$NumFrags(inst, Vp)$ — количество фрагментов объекта, которые видимы при данном положении камеры,

$t_{exec\_query}(HW)$ — время выполнения проверки видимости на этой вычислительной системе,

$t_{recv\_query}(HW)$ — время получения результата проверки видимости на данной вычислительной системе.

2.7 Вычисление покрываемого объектом количества фрагментов

Количество фрагментов, которые объект (узел дерева) занимает на экране, можно определить путем проекции его ограничивающего параллелепипеда AABB на экран. Доля видимых фрагментов рассчитывается по формуле:

$$NumFrags = w\,h \left(\frac{1}{\sqrt{w/h}\; 2\, tg\left(fov/2\right)} \right)^2 \frac{bb_x bb_y}{d^2}, \quad (23)$$

где w — ширина экрана,

h — высота экрана,

fov — вертикальный угол обзора,

bb_x — ширина видимой стороны ограничивающего параллелепипеда AABB объекта сцены,

bb_y — высота видимой стороны ограничивающего параллелепипеда AABB объекта сцены,

d — расстояние от камеры до ближайшей грани ограничивающего параллелепипеда AABB объекта сцены.

Вместо произведения $bb_x bb_y$ можно использовать среднюю площадь грани $A_{bb}/6$.

2.8 Генерация тестовых наборов для вычисления параметров модели

Оценим скорость работы вычислительной системы HW при выполнении этапов рендеринга. Современные видеокарты содержат большое количество потоковых процессоров, которые выполняют параллельную обработку данных. Итоговое время вычислений зависит от объема работы (количества вершин, треугольников и объектов в сцене).

Производится генерация трехмерных сцен с различными параметрами:

- количество вершин, приходящихся на объект ($batchSize$),
- количество объектов ($numObjects$),
- материал.

Параметр $batchSize$ имеет значение для производительности, потому что при выполнении рендеринга производится установка значений переменных для каждого объекта (uniform variables). Частое изменение этих переменных ухудшает производительность рендеринга. Произведение параметров $batchSize * numObjects$ определяет объем работы (суммарное количество вершин).

Пусть характерный размер объекта равняется a. Тогда длина, ширина и высота генерируемой сцены вычисляются, как $\sqrt[3]{numObjects}\; a$. Объекты равномерно заполняют весь объем сцены. Используется один материал для всех объектов. Предлагается выполнять генерацию таких сцен при увеличении значений параметров $batchSize$, $numObjects$ с некоторым шагом до тех пор, пока выполняются условия:

- Геометрия сцены помещается в доступную видеопамять.
- Командный буфер помещается в доступную основную память.

Для оценки времени выполнения этапов рендеринга нужно взять тестовую сцену с близким значением параметров $batchSize$, $numObjects$.

Для оценки $t_{frag}(mat, visFrags, HW)$ достаточно произвести вывод пары треугольников на весь экран и измерить время работы конвейера GPU. Обозначим количество обработанных фрагментов при выполнении рендеринга сцены, как $visFrags$. Тогда время обработки фрагмента равняется времени работы конвейера, поделенному на значение $visFrags$. Также производится оценка времени обработки фрагмента в случаях, когда пара треугольников занимает только часть экрана: $3/4$, $1/2$, $1/4$, $1/8$, $1/16$. В качестве значения $t_{frag}(mat, visFrags, HW)$ для некоторой входной сцены берется время обработки фрагмента при выполнении рендеринга тестовой сцены с наиболее близким значением $visFrags$.

Для вычисления $t_{state}(HW)$ выполняется рендеринг одного треугольника, что вызывает задержку, связанную со сменой состояния конвейера. При этом количество обрабатываемых фрагментов треугольника должно быть незначительным ($NumFrags(inst, Vp) \leq 10$). Воспользуемся формулой (23) и получим расстояние от камеры, на котором выполняется это условие:

$$d \geq \frac{h}{2\, tg\left(fov/2\right)} \sqrt{\frac{bb_x bb_y}{10}}. \quad (24)$$

2.9 Метод оценки количества аппаратных проверок видимости для эффективного выполнения рендеринга

Аппаратные проверки видимости позволяют выполнить отбраковку большого количества невидимых треугольников, чтобы сократить время работы графического конвейера. Однако на выполнение аппаратных проверок видимости также расходуются вычислительные ресурсы. Необходимо оценивать эффективность аппаратных проверок видимости в процессе отображения динамической сцены.

В отличие от существующих методов предложенный в этом разделе метод предназначен для динамических сцен. В течение некоторого времени накапливаются данные о видимости объектов. Затем производится вычисление и адаптивное обновление количества проверок видимости в процессе отображения. Использование аналитических формул позволяет сократить затраты и получить релевантные оценки для текущего состояния сцены.

Выделяются следующие стадии метода:

1. Применение ограниченного количества проверок видимости $N_{queries}$. Постепенное накопление данных о видимости объектов.
2. Вывод количества проверок видимости $N_{queries}^{efficient}$ для эффективного рендеринга.
3. Применение нового количества проверок видимости $N_{queries}^{efficient}$ до тех пор, пока это выгодно и количество изменившихся объектов незначительно.

Во время первой стадии производится обход дерева в ширину для приоритетного отбора узлов с наибольшим количеством объектов. При этом пропускаются узлы, для которых не выполняется критерий отбора:

$$t_{\text{sec\_buf}}(HW) + \max\left(t_{state}(HW), NumVerts(node)\hat{t}_{vert}, NumFrags(node, Vp)\hat{t}_{frag}\right) \geq 10\left(t_{buf}(HW) + t_{exec\_query}(HW) + t_{recv\_query}(HW)\right),$$

где $\hat{t}_{vert}$ — среднее время преобразования вершины для всех материалов сцены,

$\hat{t}_{frag}$ — среднее время расчета цвета фрагмента для всех материалов сцены.

Отметим, что в случае использования нескольких главных командных буферов работа на CPU и GPU выполняется параллельно, и в этой формуле нужно использовать максимум времени работы на CPU и GPU.

Проверяется видимость отобранных узлов. Собираются данные, задающие зависимость количества невидимых элементов (узлов дерева, объектов, вершин, фрагментов) в сцене от количества проверок видимости. Исходя из этого, выводится количество проверок видимости для эффективного рендеринга.

Определим количество проверок видимости с помощью предложенной модели производительности графического конвейера. Сначала рассмотрим сценарий, когда этапы рендеринга на CPU и GPU выполняются последовательно. Самым долгим этапом конвейера GPU является преобразование вершин или расчет цвета фрагментов. Тогда время рассматриваемых этапов определяется по формуле:

$$T(x) = c(x)t_c + s(x)t_s + e(x)t_e,$$

где x — количество проверок видимости узлов структуры пространственного разбиения,

$c(x)$ — количество записываемых команд рендеринга,

$s(x)$ — количество отправляемых командных буферов,

$e(x)$ — количество графических элементов на лимитирующем этапе конвейера (вершин или фрагментов),

t_c — время записи одной команды рендеринга,

t_s — время отправки вспомогательного командного буфера,

t_e — время выполнения лимитирующего этапа конвейера (преобразования вершины, расчета цвета фрагмента).

Накопленные данные о видимости узлов дерева позволяют определить функции $c(x)$, $s(x)$, $e(x)$. Количество невидимых узлов, объектов, вершин (фрагментов) монотонно растет с увеличением количества проверок видимости. Зачастую рост происходит быстрее при малых x, а затем останавливается при больших x. Приближенное описание таких данных задается с помощью логарифмической зависимости $c_1 + c_2 \ln(1 + x)$, где c_1 и c_2 — коэффициенты, которые вычисляются методом наименьших квадратов [22].

Накопив достаточное количество данных о видимости узлов дерева, определим коэффициенты регрессии и подставим в функции:

$$c(x) = \alpha_n(o_0 - o_1 - o_2 \ln(1 + x)) + x,$$
$$s(x) = (n_0 + 1 - s_1 - s_2 \ln(1 + x)),$$
$$e(x) = (e_0 + px - e_1 - e_2 \ln(1 + x)),$$

где α_n — средняя доля перезаписываемых командных буферов узлов дерева,

o_0 — текущее количество объектов в сцене,

n_0 — количество непустых узлов окто-дерева,

e_0 — количество вершин (фрагментов) при выполнении рендеринга сцены,

p — количество вершин (фрагментов), приходящихся на один узел окто-дерева.

Определение минимума $T(x)$ вместе с ограничениями на количество проверок видимости $0 \leq x \leq n_0$ является задачей нелинейного программирования. Функция $T(x)$ является непрерывной и дифференцируемой на множестве ограничений, поэтому минимум является либо стационарной точкой, либо лежит на границе множества ограничений [23]. Для нахождения стационарной точки решим уравнение $T'(x) = 0$ и получим:

$$x_1^* = \frac{\alpha_n o_2 t_c + s_2 t_s + e_2 t_e}{p t_e + t_c} - 1.$$

Сравнив $T(0)$, $T(n_0)$, $T(x_1^*)$, найдем минимальное время выполнения рендеринга сцены и соответствующее количество проверок видимости x.

Аналогичные вычисления можно провести и для сценария с параллельной работой CPU и GPU. Тогда время рендеринга определяется по формуле: $T(x) = \max(T_{CPU}(x), T_{GPU}(x))$. Нужно определить минимум двух функций:

$$T_{CPU}(x) = c(x)t_c + s(x)t_s,$$
$$T_{GPU}(x) = e(x)t_e.$$

Уравнение $T'_{CPU}(x) = 0$ имеет решение: $x_2^* = \frac{\alpha_n o_2 t_c + s_2 t_s}{t_c} - 1$. Сравнив $T_{CPU}(0)$, $T_{CPU}(n_0)$, $T_{CPU}(x_2^*)$, найдем минимальное время выполнения рендеринга на CPU и соответствующее значение x_2. Уравнение $T'_{GPU}(x) = 0$ имеет решение: $x_3^* = \frac{e_2}{p} - 1$. Сравнив $T_{GPU}(0)$, $T_{GPU}(n_0)$, $T_{GPU}(x_3^*)$, найдем минимальное время выполнения рендеринга на GPU и соответствующее значение x_3. После этого сравним $T(0)$, $T(n_0)$, $T(x_2)$, $T(x_3)$ и возьмем количество проверок видимости x, которое соответствует минимуму $T(x)$.

3. Вычислительные эксперименты

Проведем вычислительные эксперименты для подтверждения эффективности разработанной модели производительности графического конвейера и предложенного метода. Характеристики тестовой вычислительной системы: Intel Core i7-7700 3.6GHz, 16GB RAM, NVIDIA Geforce GTX 1070.

Для тестирования производительности методов используются сцены (рис. 3), характеристики которых приведены в табл. 1.

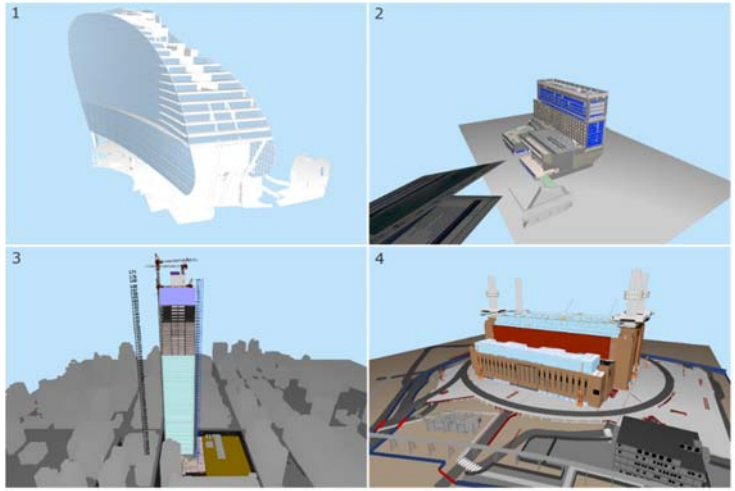

Рис. 3. Тестовые сцены 1–4
Fig. 3. Test scenes 1–4

Табл. 1. Характеристики тестовых сцен
Table 1. Properties of test scenes

Сцена	Количество вершин	Количество треугольников	Количество объектов
1	15037746	5012582	50521
2	32483139	10827713	71961
3	34609623	11536541	109991
4	94388454	31462818	270431

Проведем тестирование с использованием способа рендеринга с записью команд без кэширования. Проводилось отсечение объектов, не попадающих в область видимости камеры, а затем выполнялась запись командного буфера. Время работы рассчитывалось по формуле $T(S, S_{vis}, \emptyset, \emptyset, \emptyset, \emptyset, h = 0, g = 0, q = 0, HW)$. В табл. 2 приведены результаты тестирования с использованием тестовых сцен 1–4 при задании конечного времени анимации.

Табл. 2. Сравнение измеренного и рассчитанного времени рендеринга при использовании способа рендеринга с записью команд без кэширования
Table 2. Comparison of measured and calculated frame rendering time when employing rendering process without prerecorded commands

Сцена	Измеренное время рендеринга, мс.	Рассчитанное время рендеринга, мс.	Отклонение (%)
1	15.16	12.92	15
2	20.75	20.28	2
3	19.81	26.76	35
4	69.87	65.75	6

Проведем тестирование с использованием способа рендеринга с записью команд без кэширования и аппаратными проверками видимости каждого объекта. Проводилось получение результатов проверок видимости, отсечение объектов, не попадающих в область

видимости камеры, а затем выполнялась запись командного буфера и отправка новых проверок видимости для каждого объекта. Время работы рассчитывалось по формуле $T(S, S_{vis}, \emptyset, \emptyset, \emptyset, S_{vis}, h = 0, g = 0, q = 1, HW)$. В табл. 3 приведены результаты тестирования с использованием тестовых сцен 1–4 при задании конечного времени анимации.

Табл. 3. Сравнение измеренного и рассчитанного времени рендеринга при использовании способа рендеринга с записью команд без кэширования и аппаратными проверками видимости каждого объекта
Table 3. Comparison of measured and calculated frame rendering time when employing rendering process with occlusion queries per object and without prerecorded commands

Сцена	Измеренное время рендеринга, мс.	Рассчитанное время рендеринга, мс.
1	19.53	19.31
2	24.09	24.61
3	41.01	45.58
4	83.14	92.75

Таким образом, результаты тестирования предложенной модели производительности рендеринга показали, что ее можно применять для получения достаточно точных оценок времени рендеринга при использовании разнообразных сцен и способов рендеринга.

Для проверки эффективности метода, предложенного в разделе 2.9, проводится тестирование с использованием динамических сцен 1–4. В процессе анимации сцен происходит добавление новых объектов, удаление временных объектов, изменение цвета объектов. Тестирование заключается в отображении сцены с фиксированным положением камеры, проигрывании анимации и измерении времени рендеринга с применением следующих способов реализации рендеринга:

- Frustum Culling. Выполняется обход окто-дерева (single reference octree) с отсечением узлов, не попадающих в камеру. Проводится запись команд в главный командный буфер без кэширования.
- Рендеринг с применением предложенного метода по расчету количества проверок видимости для узлов окто-дерева.

В табл. 4 приведены результаты тестирования. Таким образом, расчет и адаптивное изменение количества проверок видимости узлов окто-дерева в процессе рендеринга позволяет получить прирост производительности до 2.3 раз по сравнению с распространенным методом Frustum Culling.

Табл. 4. Среднее время рендеринга при отображении динамических сцен
Table 4. Average frame rendering time when displaying dynamic scenes

Сцена	Frustum Culling, мс.	Рендеринг с применением предложенного метода, мс.
1	17.88	7.51
2	43.58	26.79
3	56.89	31.33
4	105.44	52.29

4. Заключение

Предложена и исследована математическая модель производительности графического конвейера для однопроходной схемы рендеринга динамических трехмерных сцен. Модель позволяет оценивать требуемые ресурсы (время обработки и передачи графических данных, объем основной и графической памяти) в зависимости от применяемых базовых методов и

характеристик отображаемой сцены. Для получения релевантных оценок на оборудовании пользователя параметры модели калибруются с использованием тестовых наборов. Предложен метод оценки количества аппаратных проверок видимости для эффективного выполнения рендеринга динамических сцен. Проведены вычислительные эксперименты, которые показали релевантность предложенной модели и эффективность разработанного метода при отображении больших динамических сцен.

Список литературы / References

[1]. Cohen-Or D., Chrysanthou Y. L., Silva C. T., Durand F. A survey of visibility for walkthrough applications. IEEE Transactions on Visualization and Computer Graphics, vol. 9, no. 3, 2003, pp. 412–431.

[2]. Airey J. Increasing Update Rates in the Building Walkthrough System with Automatic Model-Space Subdivision and Potentially Visible Set Calculations. PhD diss., University of North Carolina, Chappel Hill, 1991.

[3]. Teller S. J. Visibility computations in densely occluded polyhedral environments. PhD diss., University of California at Berkeley, 1992.

[4]. Luebke D., Georges C. Portals and mirrors: Simple, fast evaluation of potentially visible sets. In Proc. of the 1995 Symposium on Interactive 3D Graphics, 1995, pp. 105-106.

[5]. Durand F., Drettakis G., Thollot J., Puech C. Conservative visibility preprocessing using extended projections. In Proc. of the 27th Annual Conference on Computer Graphics and Interactive Techniques, 2000, pp. 239–248

[6]. Greene N., Kass M., Miller G. Hierarchical Z-buffer visibility. In Proc. of the 20th Annual Conference on Computer Graphics and Interactive Techniques, 1993, pp. 231–238.

[7]. Bittner J., Havran V., Slavik P. Hierarchical visibility culling with occlusion trees. In Proc. of the Computer Graphics International, 1998, pp. 207-219.

[8]. Coorg S., Teller S. Temporally coherent conservative visibility. Computational Geometry, vol. 12, no. 1-2, 1999, pp. 105–124.

[9]. Coorg S., Teller S. Real-time occlusion culling for models with large occluders. In Proc. of the 1997 Symposium on Interactive 3D Graphics,1997, pp. 83–90.

[10]. Hudson T., Manocha D., Cohen J., Lin M., Hoff K., Zhang H. Accelerated occlusion culling using shadow frustra. In Proc. of the 13th Annual ACM Symposium on Computational Geometry, 1997, pp 1–10.

[11]. Glassner A. S. Space subdivision for fast ray tracing. IEEE Computer Graphics and applications, vol. 4, no. 10, 1984, pp. 15–24.

[12]. Pharr M., Jakob W., Humphreys G. Physically based rendering: From theory to implementation. 3rd ed., Morgan Kaufmann, 2016, 1266 p.

[13]. Sudarsky O., Gotsman C. Dynamic scene occlusion culling. IEEE Transactions on Visualization and Computer Graphics, vol. 5, no. 1, 1999, pp. 13–29.

[14]. Morozov S., Semenov V., Tarlapan O., Zolotov V. Indexing of Hierarchically Organized Spatial-Temporal Data Using Dynamic Regular Octrees. Lecture Notes in Computer Science, vol 10742, 2018, pp. 276–290.

[15]. NV_occlusion_query. URL: https://www.khronos.org/registry/OpenGL/extensions/NV/NV_occlusion_query.txt, accessed 26.08.2020.

[16]. Bittner J., Wimmer M., Piringer H., Purgathofer W. Coherent Hierarchical Culling: Hardware Occlusion Queries Made Useful. Computer Graphics Forum, vol. 23, issue 3, 2004, pp. 615-624.

[17]. Mattausch O., Bittner J., Wimmer M. CHC++: Coherent Hierarchical Culling Revisited. Computer Graphics Forum. vol. 27, issue 2, 2008, pp. 221–230.

[18]. Guthe M., Balázs Á., Klein R. Near Optimal Hierarchical Culling: Performance Driven Use of Hardware Occlusion Queries. In Proc. of the 17th Eurographics Symposium on Rendering, 2006, pp. 207–214.

[19]. Gonakhchyan V.I. Efficient command buffer recording for accelerated rendering of large 3d scenes. In Proc. of the 12th International Conference on Computer Graphics, Visualization, Computer Vision and Image Processing, 2018, pp. 397–402.

[20]. Funkhouser T.A., Séquin C.H. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In Proc. of the 20th Annual Conference on Computer Graphics and Interactive Techniques, 1993, pp. 247–254.

[21]. Wimmer M., Wonka P. Rendering time estimation for real-time rendering. In Proc. of the 14th Eurographics Workshop on Rendering, 2003, pp. 118–129.

[22]. Weisstein, Eric W. Least Squares Fitting – Logarithmic. From MathWorld – A Wolfram Web Resource. URL: https://mathworld.wolfram.com/LeastSquaresFittingLogarithmic.html. accessed 26.08.2020.

[23]. Черняк А.А., Черняк Ж.А., Метельский Ю.М., Богданович С.А. Методы оптимизации: теория и алгоритмы. 2 изд., Юрайт, 2017, 357 стр. / Chernyak A.A., Chernyak Zh.A., Metelsky Yu.M., Bogdanovich S.A. Optimization methods: theory and algorithms. 2nd ed., Urait, 2017, 357 p. (in Russian).

Информация об авторах / Information about authors

Вячеслав Игоревич ГОНАХЧЯН – младший научный сотрудник отдела системной интеграции и прикладных программных комплексов. Сфера научных интересов: компьютерная графика, вычислительная геометрия.

Viacheslav Igorevich GONAKHCHIAN – junior research fellow of the department of System integration and multi-disciplinary collaborative environments. Research interests: computer graphics, computational geometry.