№ 4(58) 2026
НАУКА і ТЕХНІКА СЬОГОДНІ
ISSN 2786-6025 Online
УДК 004.254:004.451.57:004.92
https://doi.org/10.52058/2786-6025-2026-4(58)-3133-3149

Гуменюк Вікторія Віталіївна асистент кафедри комп'ютерних наук та інформаційних технологій фізико-математичного факультету Житомирського державного університету імені Івана Франка, м. Житомир, https://orcid.org/0000-0002-4966-6300

Іванов Дмитро Євгенійович доктор технічних наук, доцент, професор кафедри комп'ютерних наук та інформаційних технологій фізико-математичного факультету Житомирського державного університету імені Івана Франка, м. Житомир, https://orcid.org/0000-0001-9956-6589

ФРАГМЕНТАЦІЯ ТА КЕШУВАННЯ КОМАНДНИХ БУФЕРІВ У СУЧАСНИХ ГРАФІЧНИХ API (ПІДВИЩЕННЯ ПРОДУКТИВНОСТІ РЕНДЕРИНГУ ЧЕРЕЗ ВИКОРИСТАННЯ ВТОРИННИХ БУФЕРІВ (НА ПРИКЛАДІ VULKAN))

Анотація. У статті досліджено вплив фрагментації та кешування командних буферів на продуктивність рендерингу в сучасних графічних API на прикладі Vulkan. Актуальність теми зумовлена зростанням складності графічних сцен, підвищенням вимог до швидкодії рендеринг-систем і необхідністю зниження накладних витрат CPU під час підготовки графічних команд. Метою роботи є визначення умов, за яких використання secondary command buffers забезпечує підвищення ефективності рендерингу, а також уточнення меж доцільності фрагментації та кешування командних буферів у динамічних сценах. У дослідженні використано методи системного, порівняльного та аналітичного аналізу офіційної специфікації Vulkan, рекомендацій Khronos, а також наукових праць, присвячених багатопотоковому рендерингу та оптимізації графічного конвеєра. Узагальнено підходи до організації primary і secondary command buffers, повторного запису, часткового повторного використання буферів і керування command pool. Встановлено, що secondary command buffers забезпечують позитивний ефект у CPU-bound сценаріях із великою кількістю draw calls та збалансованим розподілом команд між потоками. Доведено, що надмірна фрагментація сцени, мала наповненість окремих буферів командами й нераціональне керування їх життєвим циклом призводять до зростання CPU-overhead і нівелюють переваги паралелізації. Наукова новизна роботи полягає в систематизації умов ефективного використання secondary command buffers у Vulkan та в уточненні меж

ISSN 2786-6025 Online

доцільності їх фрагментації й кешування. Практична значущість результатів полягає у можливості їх використання під час проєктування продуктивних Vulkan-рендерингових систем. Перспективи подальших досліджень пов'язані з експериментальною перевіркою отриманих висновків на різних апаратних конфігураціях і типах графічних сцен.

Ключові слова: Vulkan; command buffers; secondary command buffers; primary command buffer; фрагментація командних буферів; кешування командних буферів; багатопотоковий рендеринг; продуктивність рендерингу; CPU-overhead; графічний конвеєр.

Humeniuk Viktoriia Vitaliivna Assistant of the Department of Computer Science and Information Technologies, Faculty of Physics and Mathematics, Zhytomyr Ivan Franko State University, Zhytomyr, https://orcid.org/0000-0002-4966-6300

Ivanov Dmytro Evgeniyovych Doctor of Technical Sciences, Associate Professor, Professor of the Department of Computer Science and Information Technologies, Faculty of Physics and Mathematics, Zhytomyr Ivan Franko State University, Zhytomyr, https://orcid.org/0000-0001-9956-6589

FRAGMENTATION AND CACHING OF COMMAND BUFFERS IN MODERN GRAPHICS APIS (IMPROVING RENDERING PERFORMANCE THROUGH THE USE OF SECONDARY BUFFERS: THE CASE OF VULKAN))

Abstract. The article investigates the impact of command buffer fragmentation and caching on rendering performance in modern graphics APIs, using Vulkan as a case study. The relevance of the topic is determined by the growing complexity of graphic scenes, the increasing performance requirements imposed on rendering systems, and the need to reduce CPU overhead during the preparation of graphics commands. The purpose of the study is to identify the conditions under which the use of secondary command buffers improves rendering efficiency, as well as to clarify the limits of reasonable fragmentation and caching of command buffers in dynamic scenes. The research employs methods of systemic, comparative, and analytical analysis of the official Vulkan specification, Khronos recommendations, and scientific works devoted to multithreaded rendering and graphics pipeline optimization. The study generalizes approaches to the organization of primary and secondary command buffers, re-recording, partial buffer reuse, and command pool management. It has been established that secondary command buffers provide a positive effect in CPU-bound scenarios with a large number of draw calls and a

ISSN 2786-6025 Online
balanced distribution of commands across threads. It is demonstrated that excessive scene fragmentation, low command density within individual buffers, and inefficient lifecycle management increase CPU overhead and neutralize the advantages of parallelization. The scientific novelty of the study lies in the systematization of the conditions for the effective use of secondary command buffers in Vulkan and in clarifying the limits of their reasonable fragmentation and caching. The practical significance of the results lies in the possibility of their application in the design of high-performance Vulkan rendering systems. Prospects for further research are associated with the experimental verification of the obtained conclusions on different hardware configurations and types of graphic scenes.

Keywords: Vulkan; command buffers; secondary command buffers; primary command buffer; command buffer fragmentation; command buffer caching; multithreaded rendering; rendering performance; CPU overhead; graphics pipeline.

Постановка проблеми. У сучасних системах комп'ютерної графіки продуктивність рендерингу визначається не лише обчислювальними можливостями GPU, а й ефективністю підготовки команд на рівні CPU. Зі зростанням складності сцен, кількості draw calls і частки динамічних об'єктів саме запис і керування командними буферами дедалі частіше стають чинниками, що обмежують швидкодію графічного застосунку.

Архітектура Vulkan орієнтована на явне керування ресурсами та підтримує паралельний запис команд у різні буфери, зокрема із застосуванням secondary command buffers, які виконуються з primary command buffer в межах одного проходу рендерингу. Це створює передумови для зменшення CPU frame time і ефективнішого використання багатоядерних процесорів. [1–3]

Водночас практичне використання secondary command buffers і механізмів їх кешування не гарантує автоматичного приросту продуктивності. Офіційні матеріали Vulkan засвідчують, що багатопотоковий запис команд може скорочувати час підготовки кадру, однак надмірна кількість буферів, дрібна фрагментація навантаження та нераціональне керування command pool здатні, навпаки, збільшувати CPU-overhead.

Крім того, у динамічних сценах складні схеми кешування не завжди є виправданими, оскільки повторне використання попередньо записаних буферів потребує додаткового контролю змін сцени та своєчасного перезапису команд. Тому ефективність такого підходу визначається не самим фактом фрагментації, а раціональністю її реалізації.

У зв'язку з цим актуальним є дослідження фрагментації та кешування командних буферів у Vulkan з погляду їх впливу на CPU-складову рендерингу, вибір кількості secondary command buffers, організацію багатопотокового запису та підходи до керування командними пулами. Науково-практичне

ISSN 2786-6025 Online

значення проблеми полягає у виробленні таких рішень для побудови рендеринг-конвеєра, які забезпечують зниження накладних витрат CPU без надмірного ускладнення архітектури системи.

Аналіз останніх досліджень і публікацій. Питання підвищення продуктивності рендерингу в сучасних графічних API, зокрема у Vulkan, висвітлено в офіційній документації Khronos Group, а також у працях C. Ioannidis, A.-M. Boutsi, О. Новікова, К. Коляди, Б. В. Цапка, М. Ю. Процика та Т. О. Коротєєвої. У цих джерелах обґрунтовано переваги Vulkan як низькорівневого API та підтверджено ефективність багатопотокового формування графічних команд. Водночас питання меж ефективності secondary command buffers, їх фрагментації та кешування в динамічних сценах залишаються недостатньо систематизованими і розробленими. [1, 4–10]

Мета статті – дослідження впливу фрагментації та кешування командних буферів на продуктивність рендерингу в сучасних графічних API на прикладі Vulkan та визначення умов, за яких використання secondary command buffers забезпечує зниження накладних витрат CPU і підвищення ефективності підготовки графічних команд.

Виклад основного матеріалу. У Vulkan командний буфер виступає базовою формою подання роботи для GPU: у ньому фіксуються команди прив'язки конвеєрів, дескрипторів, зміни динамічного стану, draw- і dispatch-виклики, копіювання ресурсів та інші операції, які згодом подаються на виконання через чергу *VkQueue*. Специфікація API розрізняє два рівні командних буферів: primary command buffers, які можуть безпосередньо подаватися в чергу й виконувати *secondary command buffers*, і secondary command buffers, які самі в чергу не подаються, а виконуються з *primary command buffer* через *vkCmdExecuteCommands*. Така модель не є формальною деталлю реалізації, а становить одну з ключових архітектурних особливостей Vulkan, оскільки саме через неї досягається декомпозиція великого набору графічних команд і його підготовка в багатопотоковому режимі. [4, 11]

Принциповою рисою організації командних буферів у Vulkan є те, що кожен буфер керує станом незалежно від інших. Специфікація прямо вказує, що між *primary* і *secondary command buffers* немає повного автоматичного наслідування стану, а на початку запису стан командного буфера є невизначеним. Винятки стосуються передусім контексту render pass і деяких спеціалізованих механізмів успадкування, але загальна логіка полягає в тому, що *secondary command buffer* не є «фрагментом» primary у сенсі прозорого продовження поточного стану, а радше окремою одиницею запису, яка має бути підготовлена з урахуванням потрібного контексту виконання. Це накладає додаткові вимоги на організацію рендеринг-конвеєра, але водночас дає розробнику значно точніший контроль над структурою кадру. [4]

ISSN 2786-6025 Online

Основні відмінності між primary і secondary command buffers у Vulkan узагальнено в табл. 1.

Таблиця 1

Порівняльна характеристика primary і secondary command buffers у Vulkan

Характеристика	Primary command buffer	Secondary command buffer
Подання в VkQueue	Може безпосередньо подаватися в чергу.	Безпосередньо в чергу не подається; виконується з primary через vkCmdExecuteCommands.
Роль у кадрі	Координує загальну послідовність виконання команд і може викликати secondary buffers.	Використовується для декомпозиції частин кадру та паралельного запису команд.
Наслідування стану	Після виконання secondary буферів стан primary загалом не вважається автоматично збереженим, окрім визначених винятків.	Повного наслідування стану не має; для render pass застосовується VkCommandBufferInheritanceInfo.
Типове застосування	Монолітний запис кадру, керування render pass, submission у чергу.	Багатопотоковий запис draw calls, поділ сцени на логічні фрагменти.
Основний ризик	Однопотоковий запис може зробити CPU вузьким місцем.	Надмірна кількість буферів і дрібна фрагментація збільшують CPU-overhead.

Складено автором на основі Vulkan Specification, Vulkan Guide та Khronos Vulkan Samples.

Як видно з табл. 1, primary command buffer виконує координуючу функцію в межах кадру та забезпечує подання роботи в чергу, тоді як secondary command buffers доцільно розглядати як інструмент декомпозиції команд і багатопотокового запису. Водночас їх застосування потребує точнішого керування станом, контекстом виконання та кількістю створюваних буферів, оскільки надмірна фрагментація може призводити до зростання накладних витрат CPU.

У межах *render pass secondary command buffers* можуть використовуватися як механізм розподілу робіт між кількома потоками CPU. Для цього primary command buffer ініціює підпрохід із режимом *VK_SUBPASS_CONTENTS_SECONDARY_COMMAND_BUFFERS*, після чого *secondary command buffers* записуються окремо й викликаються з primary командою *vkCmdExecuteCommands* до завершення render pass. За офіційними рекомендаціями Khronos, *secondary command buffers* можуть успадковувати стан render pass через *VkCommandBufferInheritanceInfo*, який передається до *vkBeginCommandBuffer* разом із прапором *VK_COMMAND_BUFFER_USAGE_RENDER*

ISSN 2786-6025 Online

_PASS_CONTINUE_BIT. Саме ця схема є технічною основою багатопотокового формування кадру у Vulkan. [1, 12]

Потреба у фрагментації командних буферів безпосередньо пов'язана з проблемою CPU-overhead. Сучасні сцени, особливо насичені великою кількістю об'єктів і draw calls, навантажують не лише GPU, а й процесорну частину рендеринг-конвеєра, оскільки кожен кадр потребує підготовки команд, розподілу ресурсів, оновлення дескрипторів, керування пулами та синхронізації. Vulkan не приховує ці витрати за абстракціями драйвера, як це характерно для старіших API, а переносить значну частину відповідальності на застосунок.

Саме тому продуктивність рендерингу в Vulkan часто залежить не стільки від пікової потужності GPU, скільки від того, наскільки ефективно організовано запис і повторне використання команд на стороні CPU. Vulkan Guide прямо підкреслює, що багатопотоковість у Vulkan масштабується саме на host-side, тобто дає приріст завдяки кращому використанню ядер CPU, а не за рахунок «автоматичної» внутрішньої багатопотоковості реалізації. [2, 6]

Типову схему взаємодії primary і secondary command buffers у межах багатопотокового формування кадру наведено на рис. 1. [1]

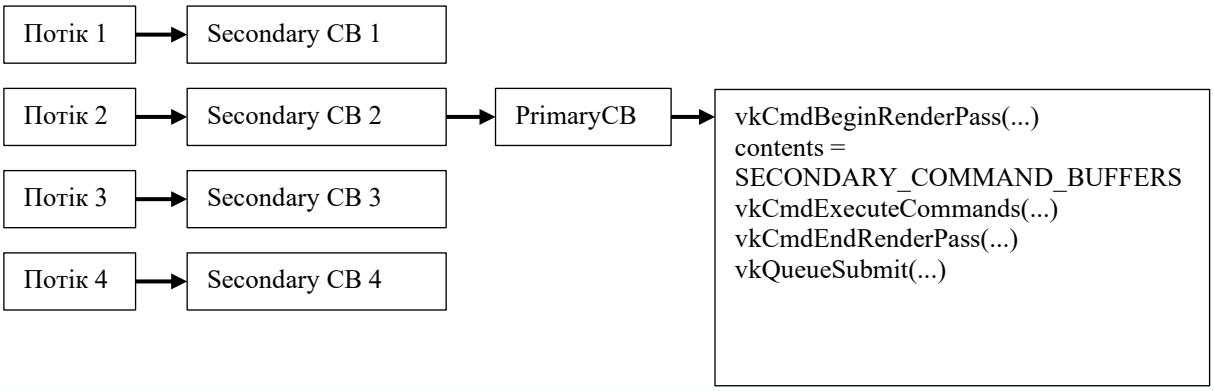

Рисунок 1 – Схема організації primary та secondary command buffers у Vulkan

Складено автором на основі офіційної документації Vulkan та рекомендацій Khronos щодо організації primary і secondary command buffers.

Схема відображає типову модель, у якій secondary command buffers записуються паралельно в кількох потоках, після чого primary command buffer виконує їх у межах render pass і подається в чергу на виконання.

Офіційний performance-sample Khronos демонструє, що багатопотоковий запис draw calls у кілька secondary command buffers здатний зменшувати CPU frame time, а в тестовій сцені з приблизно 1800 draw calls було зафіксовано близько 15 % покращення продуктивності при розподілі навантаження між вісьмома буферами на восьми потоках. [1]

ISSN 2786-6025 Online

Однак сама наявність secondary command buffers ще не гарантує виграшу. Khronos окремо наголошує, що виклики secondary command buffers є дорогими, тому їх кількість у кадрі слід мінімізувати. Якщо фрагментація надто дрібна і кожен secondary buffer містить мало draw calls, GPU може простоювати в очікуванні команд, а сумарні накладні витрати на підготовку й виклик буферів починають переважати очікуваний ефект від паралелізації. Додатково рекомендовано не перевищувати рівень CPU-паралелізму, тобто не використовувати більше буферів, ніж реально доступно потоків, і водночас не створювати ситуації, коли потоків більше, ніж придатних до паралельного запису фрагментів роботи. Отже, фрагментація в Vulkan — це не просто «поділити більше», а знайти збалансований рівень декомпозиції команд. [1]

Істотне значення має й те, як саме організовано багатопотоковий запис. Vulkan допускає паралельний запис різних command buffers, але command pool повинен бути зовнішньо синхронізований і не може одночасно використовуватись кількома потоками. Тому практична модель багатопотокового рендерингу передбачає виділення окремого command pool для кожного потоку, а часто й окремих descriptor pools та пов'язаних кешів на кожний кадр у польоті. Така організація дозволяє уникнути блокувань і зберегти перевагу від паралельного запису, але водночас підвищує вимоги до архітектури застосунку. Інакше кажучи, secondary command buffers ефективні лише тоді, коли їх використання підтримане правильно побудованою інфраструктурою ресурсних пулів.

Ще одним передумовним чинником фрагментації є спосіб життєвого циклу командних буферів. Performance-sample Khronos показує, що часте allocate/free створює найбільший CPU-overhead, тоді як періодичний *vkReset CommandPool()* є вигіднішим за часте *vkResetCommandBuffer()*. Водночас Vulkan Guide застерігає від надмірної віри в повторне використання вже записаних буферів: у динамічних сценах із додаванням і видаленням об'єктів, змінами видимості та варіативністю draw calls така стратегія ускладнює систему кешування і не завжди дає реальний приріст продуктивності. Через це фрагментація в сучасному Vulkan-рендерері має розглядатися не ізольовано, а в поєднанні з політикою перезапису, скидання та вибіркового повторного використання буферів. [1, 5]

Отже, організація командних буферів у Vulkan формує чітку технічну основу для оптимізації CPU-частини рендерингу через фрагментацію команд і багатопотоковий запис. Primary command buffer виконує роль координатора кадру та точки подання роботи в чергу, тоді як secondary command buffers дають змогу розділити формування команд між потоками й зменшити навантаження на один CPU-потік. Разом із тим така модель ефективна лише за умови раціонального розміру фрагментів, обмеженої кількості secondary buffers,

ISSN 2786-6025 Online

окремих пулів для потоків та виваженого підходу до повторного використання буферів. Саме тому фрагментація командних буферів у Vulkan доцільно розглядати як інструмент структурної оптимізації рендеринг-конвеєра, а не як універсальне рішення, що автоматично підвищує продуктивність у будь-якому сценарії.

Найпростішою схемою є запис усіх команд кадру в один primary command buffer. Такий підхід забезпечує мінімальну логічну складність, не потребує викликів vkCmdExecuteCommands, спрощує відстеження стану й робить керування кадром максимально централізованим. Саме цей варіант використовується як базовий сценарій у performance-samples Khronos, де він протиставляється фрагментованому запису через secondary command buffers. Проте монолітна схема має принципове обмеження: вона фактично серіалізує запис команд на CPU, а отже, не дозволяє повноцінно використати багатоядерність сучасних процесорів. C. Ioannidis і A.-M. Boutsi, аналізуючи багатопотокову візуалізацію великих 3D-моделей на Vulkan, прямо виходять із того, що збільшення складності сцени потребує розподілу навантаження між кількома потоками CPU; однопотоковий запис виступає лише вихідною точкою для вимірювання ефекту від подальшої декомпозиції. Саме тому схема з одним primary buffer є не універсальним рішенням, а контрольним варіантом, від якого відштовхується подальше порівняння. [1, 7]

Окрему групу становлять підходи, пов’язані з поділом кадру на незалежні або напівнезалежні частини. У performance-sample Khronos, присвяченому multithreading with multiple render passes, показано два базові варіанти: або кожен render pass записується в окремий primary command buffer, або проходи кодуються в secondary command buffers і потім виконуються з primary buffer. Перший варіант дає ефект тоді, коли великі етапи кадру природно розділені, наприклад тіньовий і основний проходи. У такому разі декомпозиція відбувається на рівні самих етапів рендерингу, а не на рівні дрібних груп draw calls. Другий варіант іде далі: він дозволяє не просто відокремити проходи, а й розподілити запис команд між кількома потоками всередині одного render pass. За даними Khronos, саме використання двох потоків уже зменшує frame time, а secondary command buffers у такій конфігурації дають додаткове зниження часу кадру порівняно з однопотоковим базовим сценарієм. [13]

Найбільш показовим для цієї статті є підхід, у якому opaque-геометрія або інші великі групи об’єктів розподіляються між кількома secondary command buffers. Специфікація Vulkan фіксує, що secondary command buffers виконуються в primary buffer у тому порядку, у якому вони передані до *vkCmdExecuteCommands*, а при записі всередині render pass використовують *VkCommandBufferInheritanceInfo*; при цьому повного автоматичного наслідування стану між primary і secondary buffers немає. Звідси випливає, що

ISSN 2786-6025 Online

фрагментація на secondary buffers не зводиться до механічного розбиття draw calls, а вимагає правильної побудови контексту запису й виконання. У performance-sample command buffer usage and multi-threaded recording показано, що при сцені приблизно на 1800 draw calls розподіл навантаження між вісьмома secondary buffers на вісьмох потоках дає близько 15 % приросту продуктивності, але лише за умови, що на кожен буфер припадає достатній обсяг роботи. Якщо ж secondary buffers занадто багато і кожен містить мало команд, їх виклики починають створювати додатковий overhead.[1, 4, 7]

Логіку вибору між повторним записом і кешуванням command buffers відображено на рис. 2. Схема концентрує увагу не на низькорівневих викликах API, а на архітектурному рішенні, яке визначає життєвий цикл буфера в межах кадру та між кадрами

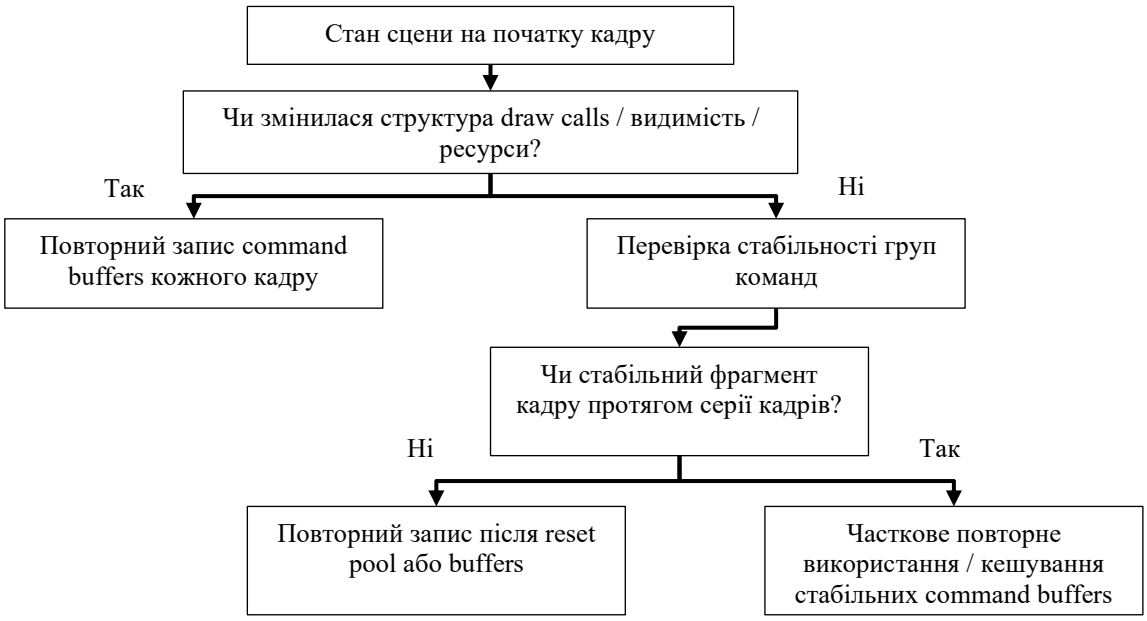

Рисунок 2 – Схема вибору стратегії роботи з command buffers у динамічному кадрі

Складено автором на основі Vulkan Guide, Khronos Vulkan Samples та узагальнення підходів до повторного запису і кешування command buffers у динамічних сценах.

Як показано на рис. 2, повне кешування command buffers є доцільним лише для стабільних послідовностей команд. Vulkan Guide зазначає, що стратегія «записати один раз і використовувати повторно» у динамічних сценах рідко забезпечує очікуваний виграш, оскільки зміни складу об’єктів, видимості та draw calls вимагають інвалідизації кешу й повторного запису. Тому в динамічному рендерингу базовим підходом є перезапис command buffers кожного кадру, тоді як часткове повторне використання виправдане лише для стабільних фрагментів сцени. [4–5]

ISSN 2786-6025 Online

Важливим чинником є і спосіб скидання та повторного використання пам'яті command buffers. За даними Khronos, найвищий CPU-overhead виникає при частому vkAllocateCommandBuffers()/vkFreeCommandBuffers(), тоді як vkResetCommandPool() є ефективнішим за vkResetCommandBuffer() завдяки повторному використанню пам'яті пулу. Водночас багатопотоковий запис потребує окремого command pool для кожного потоку, тому організацію command buffers слід узгоджувати з архітектурою resource pools. [1–2]

Ключове питання дослідження полягає у визначенні умов, за яких використання secondary command buffers забезпечує зниження CPU-витрат і скорочення часу підготовки кадру.

Такий ефект залежить від кількості draw calls, рівня фрагментації сцени, динамічності об'єктів, кількості secondary command buffers і способу керування command pool. [1, 5–6]

Першою базовою закономірністю є залежність ефекту secondary command buffers від щільності графічного навантаження, тобто від кількості draw calls і вартості їх підготовки на CPU. У performance-sample Khronos, присвяченому multi-threaded recording, прямо зазначено, що на сцені з великою кількістю викликів малювання — близько 1800 draw calls — розподіл навантаження між вісьмома secondary command buffers на вісьмох потоках дає приблизно 15 % приросту продуктивності.

Це означає, що при достатньо великому обсязі команд витрати на виклик *vkCmdExecuteCommands* і координацію кількох буферів перекриваються виграшем від паралельної підготовки команд. На думку О. О. Новікова та К. В. Коляди, саме явний контроль над взаємодією CPU і GPU, характерний для Vulkan, створює передумови для такого виграшу, але тільки за умови раціональної побудови рендеринг-конвеєра; без цього такий самий механізм перетворюється на джерело додаткових накладних витрат. [1, 8]

Водночас secondary command buffers не дають позитивного ефекту в ситуаціях, де фрагментація є надмірною. Khronos прямо підкреслює, що кількість secondary command buffers у кадрі повинна залишатися невеликою, оскільки їхні виклики є дорогими, а надто велика кількість буферів переводить застосунок у CPU-bound режим. Іншими словами, якщо сцена розбивається на надто дрібні фрагменти і кожен буфер містить лише невеликий набір draw calls, то overhead на створення, запис, диспетчеризацію і виконання secondary buffers починає переважати виграш від багатопотокового запису.

У цьому полягає друга принципова закономірність: secondary command buffers ефективні не при максимальному, а при збалансованому рівні фрагментації. [7]

Узагальнення впливу ключових параметрів фрагментації на продуктивність рендерингу наведено в табл. 2

ISSN 2786-6025 Online

Таблиця 2

Вплив параметрів фрагментації secondary command buffers на продуктивність рендерингу

Параметр	Характер зміни	Вплив на продуктивність
Кількість draw calls	Зростає	Ефективність secondary buffers підвищується, якщо обсяг команд достатній для паралельного запису
Кількість secondary command buffers	Зростає помірно	Продуктивність поліпшується за умови відповідності кількості буферів реальній кількості робочих потоків
Кількість secondary command buffers	Зростає надмірно	CPU-overhead збільшується, ефект прискорення нівелюється
Динамічність сцени	Низька або локальна	Часткове повторне використання буферів може бути виправданим
Динамічність сцени	Висока	Переважає повторний запис буферів кожного кадру
Спосіб керування пам'яттю буферів	vkResetCommandPool()	Найменші службові витрати на керування буферами
Спосіб керування пам'яттю буферів	vkResetCommandBuffer()	Вищі витрати, ніж при reset pool
Спосіб керування пам'яттю буферів	vkAllocateCommandBuffers() / vkFreeCommandBuffers()	Найгірший сценарій за CPU-overhead

Складено автором за даними Khronos Vulkan Samples, Vulkan Guide, C. Ioannidis, A.-M. Boutsi, О. О. Новікова, К. В. Коляди

Як видно з табл. 3, secondary command buffers створюють позитивний ефект лише за поєднання трьох умов: достатньо великого обсягу команд, помірної кількості фрагментів і раціонального керування пам'яттю command pool. Порушення хоча б однієї з цих умов веде до того, що додаткові накладні витрати на фрагментацію починають переважати очікуваний виграш від паралелізації. [1, 5]

Третя закономірність стосується масштабованості за кількістю потоків. У праці C. Ioannidis та A.-M. Boutsi, де реалізовано багатопотокову візуалізацію великих 3D-моделей на Vulkan, наведено показові результати: час повного рендерингу сцени зменшується з 454,07 мс у однопотоковому режимі до 233,85

ISSN 2786-6025 Online

мс при двох потоках, 173,52 мс при трьох і 125,22 мс при чотирьох потоках. Автори підкреслюють, що продуктивність зростає приблизно лінійно з кількістю ядер у системі, коли робоче навантаження добре розподілене між потоками, а primary і secondary command buffers виконують різні функції: primary buffer координує великі зміни стану, а secondary buffers концентруються на побудові й dispatch draw calls у межах render pass. Ці результати важливі тим, що вони демонструють не абстрактну «теоретичну» перевагу вторинних буферів, а їх реальну залежність від апаратної конфігурації хост-системи й від правильності балансування роботи між потоками.

Динаміку зміни часу рендерингу сцени залежно від кількості CPU-потоків у багатопотоковому Vulkan-конвеєрі наведено на рис. 3.

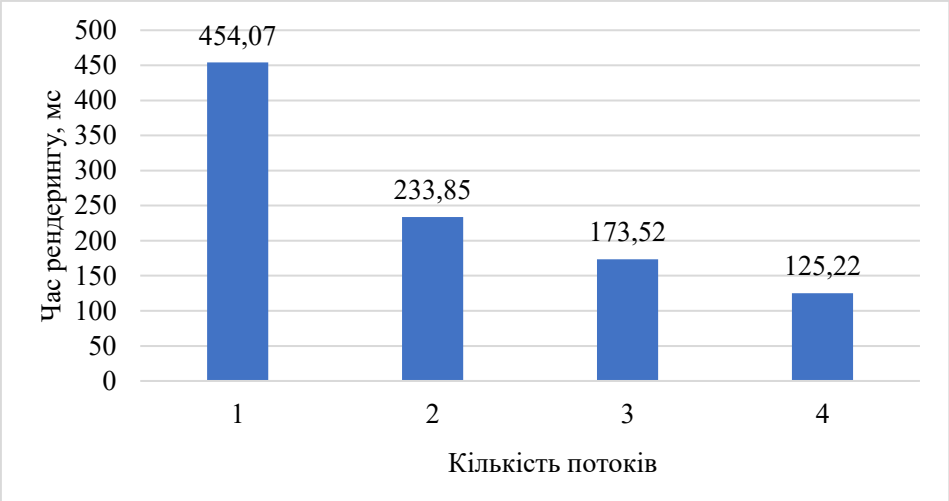

Рисунок 3 – Залежність часу рендерингу сцени від кількості CPU-потоків у багатопотоковому Vulkan-конвеєрі

Побудовано автором за даними C. Ioannidis, A.-M. Boutsi.

Четверта закономірність пов'язана зі ступенем динамічності об'єктів. Vulkan Guide прямо фіксує, що ранні рекомендації щодо «одноразового запису та багаторазового повторного використання command buffers» на практиці рідко дають очікуваний ефект. Причина полягає в тому, що у динамічних сценах постійно змінюються видимість об'єктів, результати frustum culling, склад draw calls, ресурси та набір станів, тому система кешування повинна ще й відстежувати момент, коли буфер втрачає актуальність і потребує повторного запису. Через це повне кешування secondary command buffers не зменшує складність системи, а навпаки ускладнює архітектуру рендерера. У таких умовах оптимальною стратегією виступає повторний запис буферів кожного кадру, тоді як часткове повторне використання допускається лише для стабільних фрагментів — постобробки, статичних проходів або незмінних блоків сцени. [5]

ISSN 2786-6025 Online

П’ята закономірність стосується режиму керування командними буферами і command pool. Дані Khronos показують, що найменші витрати на службові операції забезпечує *vkResetCommandPool()*: у наведеному профілі сукупний час на request + reset command buffers становив 53,3 мс за загального захоплення 11 877 мс, тобто 0,45 %. Для *vkResetCommandBuffer()* ці витрати становили 140,29 мс або 1,16 %, а сценарій з частим *vkAllocate CommandBuffers() / vkFreeCommandBuffers()* дав 3 319,25 мс, тобто 28,8 % всього профілю. Ці числа мають принципове значення: навіть правильно обрана схема фрагментації secondary buffers втрачає сенс, якщо життєвий цикл буферів організовано через часте виділення і звільнення пам’яті. Отже, аналіз продуктивності повинен охоплювати не тільки «скільки буферів створено», а й «як саме вони повторно вводяться в робочий цикл між кадрами». [1]

Отримані результати підтверджують, що вплив secondary command buffers на продуктивність рендерингу визначається не самим фактом їх використання, а характером організації всього рендеринг-конвеєра. У Vulkan secondary command buffers виступають інструментом структурної декомпозиції команд, який дає змогу перенести істотну частину роботи з підготовки кадру в багатопотоковий режим. Водночас Vulkan Guide прямо вказує, що повторне використання command buffers саме по собі рідко забезпечує заявлений приріст швидкодії, якщо сцена містить змінні draw calls, об’єкти додаються або видаляються, а видимість змінюється через culling. Це означає, що практична цінність отриманих результатів полягає у встановленні меж раціонального використання secondary command buffers, а не в проголошенні їх універсальним засобом оптимізації. [5]

Узагальнення результатів дослідження показує, що найкращий ефект secondary command buffers демонструють у тих сценаріях, де навантаження на CPU є значним і рівномірно поділяється між потоками. У performance-sample Khronos зазначено, що для сцени з приблизно 1800 draw calls розподіл навантаження між вісьмома буферами на вісьмох потоках забезпечив близько 15 % покращення продуктивності. Подібний висновок випливає і з праці C. Ioannidis та A.-M. Boutsi, де багатопотоковий Vulkan-конвеєр для великорозмірних 3D-моделей продемонстрував суттєве скорочення часу рендерингу зі зростанням кількості CPU-потоків за умови збалансованого розподілу роботи. Отже, результати статті узгоджуються з відомими підходами: secondary command buffers дають виграш тоді, коли вони обслуговують достатньо великі фрагменти команд і реально знижують навантаження на один CPU-потік. [1, 7]

Разом із тим проведений аналіз дає підстави уточнити відомі рекомендації. Якщо в загальних матеріалах з Vulkan оптимізація часто подається як перехід від однопотокового запису до багатопотокового, то результати цього дослідження показують, що вирішальним є не сам перехід до secondary

ISSN 2786-6025 Online

command buffers, а ступінь фрагментації сцени. За надмірної декомпозиції, коли кожен буфер містить незначну кількість draw calls, зростають витрати на vkCmdExecuteCommands, ускладнюється синхронізація і зникає позитивний ефект від паралелізації. Khronos прямо наголошує, що secondary command buffers мають залишатися відносно небагаточисельними, а кількість буферів не повинна перевищувати рівень реального CPU-паралелізму. Саме ця закономірність пояснює, чому одна й та сама техніка в різних проєктах може або зменшувати frame time, або, навпаки, посилювати CPU-bound характер застосунку. [1, 6]

На думку О. О. Новікова та К. В. Коляди, Vulkan забезпечує інженеру значно більший контроль над взаємодією CPU і GPU, ніж попередні API, але така свобода без правильної організації ресурсів і командних структур може призводити до неоптимального використання засобів платформи. У контексті отриманих результатів цей висновок набуває конкретного змісту: ефективність secondary command buffers залежить не лише від схеми запису команд, а й від архітектури command pools, descriptor pools та буферних кешів на кожен кадр і потік. Офіційний Vulkan Guide так само підкреслює, що один command pool не може використовуватися одночасно в кількох потоках, тому багатопотоковий запис потребує окремих пулів на кожний host-thread. Відтак інтерпретація результатів приводить до чіткого висновку: secondary command buffers є ефективними лише в системі, де багатопотокова підготовка команд підтримана правильною структурою пулів і мінімізованими блокуваннями. [1, 6, 8]

Практичні рекомендації щодо вибору схеми організації command buffers систематизовано в табл. 3.

Таблиця 3

Практичні рекомендації щодо використання command buffers у Vulkan-рендерингу

Сценарій рендерингу	**Рекомендована схема**	**Інтерпретація**
Невелика сцена, помірна кількість draw calls	Один primary command buffer	Додатковий поділ на secondary buffers не компенсує службові витрати
Велика сцена з високим CPU-навантаженням	Secondary command buffers з розподілом між потоками	Паралельний запис знижує CPU frame time за умови збалансованого навантаження
Кілька великих render passes	Окремі primary buffers за проходами або secondary buffers за проходами	Розподіл за природною логікою кадру спрощує багатопотоковий запис
Висока динамічність сцени	Повторний запис буферів кожного кадру	Кешування втрачає ефективність через постійну зміну draw calls

ISSN 2786-6025 Online

Сценарій рендерингу	Рекомендована схема	Інтерпретація
Стабільні фрагменти кадру	Часткове повторне використання secondary buffers	Кешування виправдане лише для незмінних або слабкозмінних блоків
Багатопотоковий режим	Окремий command pool на кожен потік	Усунення зайвих блокувань і зовнішньої синхронізації
Часте оновлення буферів	vkResetCommandPool()	Найнижчі накладні витрати порівняно з reset окремих буферів або allocate/free

Складено автором за даними Vulkan Guide, Khronos Vulkan Samples, C. Ioannidis, A.-M. Boutsi

Як видно з табл. 4, ключовим критерієм вибору виступає не «сучасність» підходу, а відповідність схеми реальному профілю навантаження. Для невеликих сцен із помірною кількістю draw calls один primary command buffer залишається найкращим рішенням, оскільки не створює додаткових витрат на координацію. Для великих сцен secondary command buffers дають відчутний ефект, але лише тоді, коли розподіл роботи між потоками не є формальним, а справді зменшує час запису команд на CPU. Для динамічних сцен визначальною стає не багатопотоковість сама по собі, а вибір між повним повторним записом і частковим кешуванням стабільних блоків. Таким чином, практичне значення результатів полягає у виробленні сценарного підходу до побудови Vulkan-рендерера, де вибір механізму залежить від структури сцени, а не від абстрактних рекомендацій. [1, 5, 13]

Узагальнення проведеного аналізу свідчить, що ефективність secondary command buffers визначається не самим фактом їх використання, а способом організації рендеринг-конвеєра. Найкращий результат забезпечує поєднання збалансованої фрагментації команд, багатопотокового запису та раціонального керування command pool. Для динамічних сцен перевагу має повторний запис буферів, тоді як кешування виправдане лише для стабільних фрагментів кадру.

Висновки. Дослідження показало, що використання secondary command buffers у Vulkan є ефективним засобом підвищення продуктивності рендерингу лише за умови раціональної організації командного навантаження, збалансованої фрагментації сцени та узгодженого багатопотокового запису. Найбільший ефект цей підхід забезпечує в CPU-bound сценаріях із великою кількістю draw calls, де скорочення часу підготовки кадру досягається завдяки паралелізації формування команд. Водночас надмірна фрагментація, мала наповненість буферів командами та нераціональне керування їх життєвим циклом призводять до зростання CPU-overhead і знижують переваги такого підходу. Отже ефективність фрагментації та кешування command buffers

ISSN 2786-6025 Online

визначається не самим фактом їх застосування, а відповідністю обраної стратегії характеру сцени, динаміці draw calls і способу керування command pool.

Література:

1. Command buffer usage and multi-threaded recording. *Vulkan Documentation* : website. URL: https://docs.vulkan.org/samples/latest/samples/performance/command_buffer_usage/README.html (дата звернення: 10.03.2026)

2. Multithreading with Vulkan. *Vulkan Documentation* : website. URL: https://docs.vulkan.org/tutorial/latest/17_Multithreading.html (дата звернення: 10.03.2026)

3. VkCommandBufferInheritanceInfo(3). *Vulkan Documentation*: website. URL: https://docs.vulkan.org/refpages/latest/refpages/source/VkCommandBufferInheritanceInfo.html

4. Command Buffers. *Vulkan Documentation* : website. URL: (дата звернення: 10.03.2026)https://docs.vulkan.org/spec/latest/chapters/cmdbuffers.html

5. Common Pitfalls for New Vulkan Developers. *Vulkan Documentation* : website. URL: https://docs.vulkan.org/guide/latest/common_pitfalls.html (дата звернення: 12.03.2026)

6. Threading. *Vulkan Documentation*: website. URL: https://docs.vulkan.org/guide/latest/threading.html

7. Ioannidis C., Boutsi A. M. Multithreaded rendering for cross-platform 3D visualization based on Vulkan Api. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. 2020, №44. P. 57-62.

8. Новіков О., Коляда К. Практика використання інструментів Vulkan API для підвищення продуктивності програмного забезпечення. *Міжнародний науковий журнал «Грааль науки»*. 2023. №26. С. 237-241.

9. Цапко Б. В. Оптимальне використання OpenGL та Vulkan // Радіоелектроніка та молодь у XXI столітті : матеріали 28-го Міжнар. молодіж. форуму, 16–18 квітня 2024 р. Харків : ХНУРЕ, 2024. Т. 6 С. 293-294.

10. Процик М. Ю., Коротєєва Т. О. Аналіз ефективності роботи програмних засобів для 3D-рендерингу в сфері розроблення відеоігор. *Науковий вісник НЛТУ України*. 2025. №35(3). С. 158-165.

11. vkCmdExecuteCommands(3). *Vulkan Documentation* : website. URL: https://docs.vulkan.org/refpages/latest/refpages/source/vkCmdExecuteCommands.html (дата звернення: 12.03.2026)

12. Render Pass. *Vulkan Documentation*: website. URL: https://github.khronos.org/Vulkan-Site/spec/latest/chapters/renderpass.html (дата звернення: 12.03.2026)

13. Multi-threaded recording with multiple render passes. *Vulkan Documentation* : website. URL: https://docs.vulkan.org/samples/latest/samples/performance/multithreading_render_passes/README.html (дата звернення: 14.03.2026)

References:

1. Command buffer usage and multi-threaded recording. Vulkan Documentation : website. URL: https://docs.vulkan.org/samples/latest/samples/performance/command_buffer_usage/README.html (date of access: 10.03.2026)

2. Multithreading with Vulkan. Vulkan Documentation: website. URL: https://docs.vulkan.org/tutorial/latest/17_Multithreading.html (date of access: 10.03.2026)

ISSN 2786-6025 Online

3. VkCommandBufferInheritanceInfo(3). Vulkan Documentation : website. URL: https://docs.vulkan.org/refpages/latest/refpages/source/VkCommandBufferInheritanceInfo.html

4. Command Buffers. Vulkan Documentation : website. URL: https://docs.vulkan.org/spec/latest/chapters/cmdbuffers.html (date of access: 10.03.2026)

5. Common Pitfalls for New Vulkan Developers. Vulkan Documentation : website. URL: https://docs.vulkan.org/guide/latest/common_pitfalls.html (date of access: 12.03.2026)

6. Threading. Vulkan Documentation : website. URL: https://docs.vulkan.org/guide/latest/threading.html

7. Ioannidis C., Boutsi A. M. Multithreaded rendering for cross-platform 3D visualization based on Vulkan Api. The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. 2020, №44. P. 57-62.

8. Novikov O., Koliada K. Praktyka vykorystannia instrumentiv Vulkan API dlia pidvyshchennia produktyvnosti prohramnoho zabezpechennia. [Practical Applications of Vulkan API Tools for Improving Software Performance] Mizhnarodnyi naukovyi zhurnal «Hraal nauky» - The International Scientific Journal "The Grail of Science". Vol. 26, pp. 237-241, 2023. (Ukr)

9. Tsapko B. V. Optymalne vykorystannia OpenGL ta Vulkan [Optimal Use of OpenGL and Vulkan] // Radioelektronika ta molod u XXI stolitti : materialy 28-ho Mizhnar. molodizh. forumu, 16–18 kvitnia 2024 r - Radio Electronics and Youth in the 21st Century: Proceedings of the 28th International Youth Forum, April 16–18, 2024. Vol. 6, pp. 293-294, 2024. (Ukr)

10. Protsyk M. Yu., Korotieieva T. O. Analiz efektyvnosti roboty prohramnykh zasobiv dlia 3D-renderynhu v sferi rozroblennia videoihor [An Analysis of the Performance of 3D Rendering Software in Video Game Development]. Naukovyi visnyk NLTU Ukrainy - Scientific Bulletin of the National Forestry University of Ukraine. Vol. 35(3), pp. 158-165, 2025.

11. vkCmdExecuteCommands(3). Vulkan Documentation : website. URL: https://docs.vulkan.org/refpages/latest/refpages/source/vkCmdExecuteCommands.html (date of access: 12.03.2026)

12. Render Pass. Vulkan Documentation : website. URL: https://github.khronos.org/Vulkan-Site/spec/latest/chapters/renderpass.html (date of access: 12.03.2026)

13. Multi-threaded recording with multiple render passes. Vulkan Documentation : website. URL: https://docs.vulkan.org/samples/latest/samples/performance/multithreading_render_passes/README.html (date of access: 14.03.2026)

Дата першого надходження статті до видання: 13.04.2026
Дата прийняття статті до друку після рецензування: 26.04.2026