DOI: 10.20998/2079-0023.2025.01.13
УДК 004.432.3

В. В. ЯЦЕНКО, кандидат технічних наук (PhD), доцент, доцент кафедри економічної кібернетики, Сумський державний університет, Суми, Україна; e-mail: v.yatsenko@biem.sumdu.edu.ua; ORCID: https://orcid.org/0000-0003-2316-3817
Д. В. ПАРХОМЕНКО, здобувач вищої освіти, Сумський державний університет, Суми, Україна; e-mail: dmytro.parkhomenko@student.sumdu.edu.ua; ORCID: https://orcid.org/0009-0003-5279-1879
О. С. КУШНЕРЬОВ, доктор філософії (PhD), старший викладач кафедри економічної кібернетики, Сумський державний університет, Суми, Україна; e-mail: o.kushnerov@biem.sumdu.edu.ua; ORCID: https://orcid.org/0000-0001-8253-5698
В. В. КОЙБІЧУК, кандидатка економічних наук (PhD), доцентка, завідувачка кафедри економічної кібернетики, Сумський державний університет, Суми, Україна v.koibichuk@biem.sumdu.edu.ua; ORCID: https://orcid.org/0000-0002-3540-7922
К. Г. ГРИЦЕНКО, кандидат технічних наук (PhD), доцент, доцент кафедри економічної кібернетики, Сумський державний університет, Суми, Україна; k.hrytsenko@biem.sumdu.edu; ORCID: https://orcid.org/0000-0002-7855-691X

ПОРІВНЯННЯ СУЧАСНИХ ІГРОВИХ РУШІЇВ З ВЛАСНИМ ЯДРОМ ДЛЯ НАТИВНОЇ РОЗРОБКИ ІГОР НА ПЛАТФОРМІ ANDROID

Сучасна ігрова індустрія дедалі більше орієнтується на мобільні платформи, зокрема пристрої з архітектурою ARM, яка є домінуючою у смартфонах і планшетах. Розробники активно адаптують свої рушії та інструменти під цю архітектуру, зважаючи на її енергоефективність та широке розповсюдження. У цьому контексті створення власного ігрового ядра, яке можна безпосередньо встановити на Android-пристрій без додаткових рушіїв, відкриває нові можливості для оптимізації, швидшого прототипування та повного контролю над продуктивністю на рівні пристрою. Такий підхід особливо актуальний на фоні зростання популярності незалежної розробки ігор та необхідності легких рішень без зайвих залежностей. У дослідженні представлено порівняльний аналіз сучасних ігрових рушіїв (Unity, Unreal Engine, Godot, Cocos2d-x, Defold) та власного ігрового ядра, орієнтованого на пряме встановлення і запуск на Android-пристроях з архітектурою ARM без проміжного рушія. У роботі розглянуто переваги архітектури ARM, включаючи енергоефективність, масштабованість і широку підтримку в мобільних пристроях, що робить її доцільною платформою для нативної розробки ігор. Особливу увагу приділено технічному порівнянню можливостей рушіїв, включно з вагою застосунків, швидкістю запуску, гнучкістю API, рівнем доступу до системних ресурсів і підтримкою низькорівневих мов. Виявлено, що хоча традиційні рушії забезпечують багатий функціонал і простоту розробки, вони обмежують контроль над апаратною частиною та призводять до збільшення розміру APK. Натомість власне ядро, створене спеціально для ARM-пристроїв, забезпечує мінімальний обсяг, миттєвий запуск і максимальну продуктивність завдяки прямому доступу до графічних API (OpenGL ES/Vulkan) і системних ресурсів Android. У роботі проаналізовано придатність мов програмування Java, Kotlin, C++ та Rust у контексті розробки ігор для Android, окреслено перспективи використання Vulkan як високопродуктивного графічного API, а також зроблено висновки щодо доцільності ядроцентричного підходу для створення легковагових, оптимізованих мобільних ігор і інструментів.

Ключові слова: ігровий рушій, ARM, Android, Vulkan, низькорівнева розробка, Java, C++, Rust, енергоефективність, мобільна оптимізація.

Вступ. Сучасний ринок демонструє стійку тенденцію до зростання мобільного сегменту, де Android-пристрої з архітектурою ARM займають лідируючі позиції. Мобільна платформа поступово витісняє традиційні формати, такі як персональні комп'ютери та ігрові консолі, завдяки широкій доступності, енергоефективності, компактності, а також зручності використання у повсякденному житті.

Швидкий розвиток мобільного «заліза», підтримка сучасних графічних бібліотек і стабільне інтернет-з'єднання роблять смартфони повноцінними ігровими платформами. Це зумовлює необхідність адаптації підходів до розробки ігор – від вибору інструментів, мов програмування та середовищ розробки до архітектурних рішень програмного забезпечення, що мають враховувати обмежені ресурси мобільних пристроїв, енергоспоживання та оптимізацію продуктивності.

Аналіз останніх досліджень і публікацій. Кількість публікацій щодо застосування ігрових рушіїв у різноманітних галузях постійно зростає (рис. 1). Аналіз наукових праць за останні 15 років у БД Scopus виявив 6931 наукову працю, більшість з яких належить до галузей Computer Science, Engineering та Mathematics.

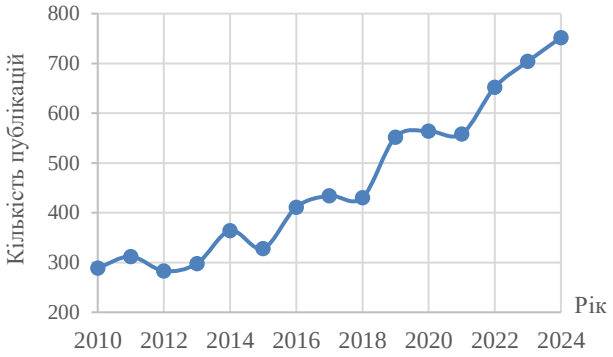

Рис. 1. Кількість публікацій з тематики ігрових рушіїв за 2010–2024 рр.

Лідером за кількістю публікацій є США – 1491 робота, другу позицію займає Китай – 702 роботи, а третю – Велика Британія з 503 дослідженнями (рис. 2). Тематика ігрових рушіїв залишається недостатньо дослідженою українськими науковцями, які публікують

значно меншу кількість робіт у журналах, що індексуються в БД Scopus – до 5 статей в рік.

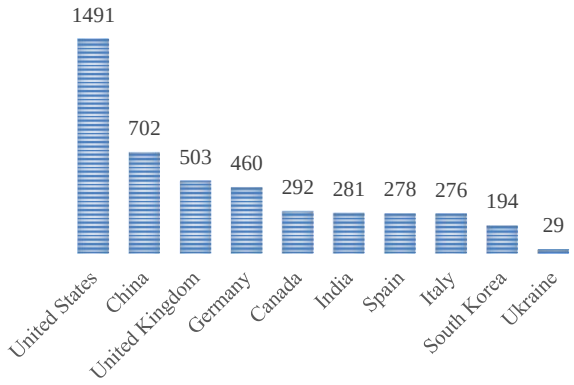

Рис. 2 Кількість публікацій за країнами з тематики ігрових рушіїв за 2010-2024 рр.

У дослідженні для візуалізації взаємозв'язків між поняттям «ігровий рушій» та іншими науковими категоріями застосовано програмне забезпечення VOSviewer. За допомогою цього інструменту ідентифіковано п'ять тематичних кластерів, представлених на схемі різними кольорами (зеленим, червоним, синім, жовтим та фіолетовим), де розмір кожного елемента відображає частоту його згадування в проаналізованих наукових публікаціях (рис. 3).

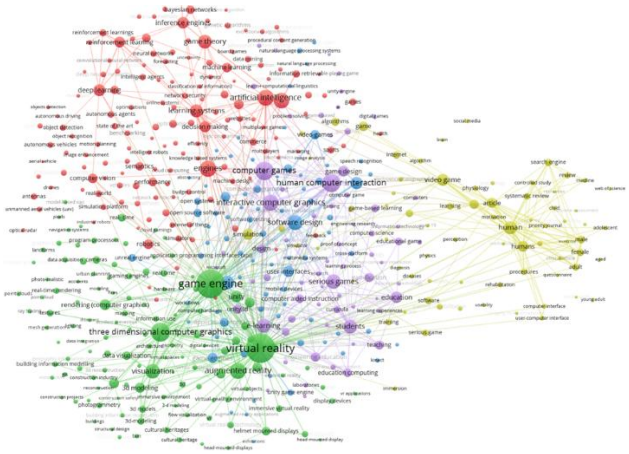

Рис. 3. Наукова бібліографія поняття «game engine» (ігровий рушій) за період з 2010 по 2024 рр.

В першому кластері (зеленому) дослідження фокусуються на питаннях, пов'язаних з використанням ігрових рушіїв для створення віртуальної реальності (VR), доповненої реальності (AR) та тривимірної (3D) графіки. Теми включають рендеринг, текстурування, моделювання, візуалізацію даних, інтерфейси користувача для VR/AR, а також застосування ігрових рушіїв у різних галузях, таких як архітектура, медицина та освіта [1].

Другий за розміром по числу згадувань кластер (червоний) об'єднує публікації, що досліджують застосування штучного інтелекту та машинного навчання в контексті ігрових рушіїв. Основні теми включають розробку інтелектуальних агентів, прийняття рішень, комп'ютерний зір для ігрових цілей, навчання з підкріпленням, обробку природної мови для взаємодії з гравцем [2].

Наступна сукупність досліджень (фіолетовий кластер) об'єднує роботи, що вивчають комп'ютерні ігри як об'єкт дослідження та як інструмент для вивчення людсько-комп'ютерної взаємодії (HCI). Ключові теми включають інтерактивну комп'ютерну графіку, ігровий досвід, гейміфікацію навчального процесу, серйозні ігри, аналіз ігрових даних, а також використання ігрових рушіїв для створення прототипів та експериментів в HCI [3, 4].

Дослідження, пов'язані з розробкою ігрових рушіїв та інструментів для створення ігор згруповані в синій кластер. Автори досліджують архітектуру ігрових рушіїв, розробку плагінів та розширень, інструменти для моделювання та анімації, питання продуктивності та оптимізації, кросплатформну та нативну розробку, тестування, особливості ігор на мобільних пристроях, а також використання різних програмних парадигм у розробці ігор [5, 6].

Кластер позначений жовтим кольором репрезентує публікації, присвячені аналізу людського фактору у взаємодії з ігровими технологіями, зокрема у сфері медицини, психології та реабілітації. Дослідження показують, як ігрові рушії використовуються у створенні інтерфейсів для пацієнтів, симуляторів терапевтичного характеру, а також у клінічних експериментах щодо впливу гейміфікації на мотивацію та ефективність лікування [7].

Аналіз еволюції наукових досліджень, пов'язаних з ігровими рушіями та суміжними темами за останні 15 років за допомогою VOSviewer (рис. 4) дозволив зробити такі висновки:

1. Зберігається інтерес до ключових тем: ігровий рушій, комп'ютерна графіка, віртуальна реальність та людсько-комп'ютерна взаємодія.

2. Дослідження застосування ігрових рушіїв розширюються за межі розваг у різноманітні галузі: освіту, охорону здоров'я, архітектуру, соціальні науки.

3. Зміщається фокус досліджень на нові технології такі як штучний інтелект, віртуальна та доповнена реальність.

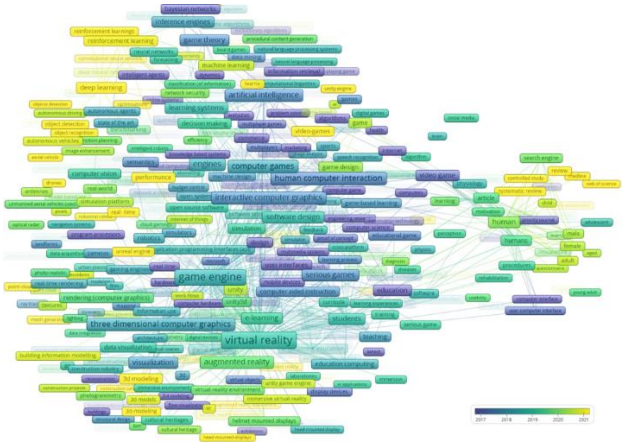

Рис. 4. Візуалізаційна схема контекстуально-часового виміру досліджень з питань ігрових рушіїв за 2010–2024 рр.

4. Рух від комп'ютерної графіки, рендерингу, моделювання та базових інструментів розробки ігор до імерсивних технологій.

5. Дослідження в галузі людсько-комп'ютерної взаємодії, пов'язані з іграми, еволюціонують від загальних принципів до більш специфічних аспектів ігрового досвіду, мотивації гравців та використання ігор для досліджень у самій HCI.

6. Фокус досліджень в галузі розробки ігор зміщується на більш складні аспекти, такі як створення розширюваних архітектур ігрових рушіїв, розробка спеціалізованих інструментів, оптимізація продуктивності для нових платформ та використання сучасних парадигм програмування.

Метою цього дослідження є технічний аналіз доцільності розробки власного ігрового ядра для ARM-пристроїв у порівнянні з традиційними ігровими рушіями. У роботі також розглядаються переваги нативної розробки, потенціал використання низькорівневих мов програмування та перспективи застосування високопродуктивних графічних API, таких як Vulkan, OpenGL ES.

Виклад основного матеріалу. У контексті стрімкої мобілізації обчислювальних платформ архітектура ARM поступово витісняє традиційні x86-рішення у сфері ігрової розробки завдяки енергоефективності, малій площі кристала та широкій інтеграції в мобільні пристрої. Порівняльний аналіз x86, ARM та нової x86S демонструє перспективність власних ігрових ядер, оптимізованих безпосередньо під ARM, як альтернативу універсальним, але важким рушіям (табл. 1) [8, 9, 10].

Водночас, варто зазначити, що розробка архітектури x86S була припинена через брак фінансування [11], що ще раз підкреслює вразливість традиційних десктоп-орієнтованих підходів до довготривалої мобільної оптимізації.

У контексті зростаючого попиту на мобільні ігри та переходу обчислювальних платформ на ARM-архітектуру актуальним стає порівняння традиційних ігрових рушіїв із кастомними низькорівневими ядрами. Такі рушії, як Unity, Unreal Engine, Godot, Defold та Cocos2d-x, забезпечують високу гнучкість і багатофункціональність, проте мають значні обмеження: велику вагу застосунків, залежність від фреймворків, а також порівняно низький рівень контролю над апаратними ресурсами (табл. 2) [12 ,13, 14, 15, 16].

Натомість власноруч розроблене ігрове ядро, яке встановлюється безпосередньо на Android-пристрій без проміжного рушія, демонструє надзвичайно малу вагу, миттєвий запуск та повну керованість логікою гри і

Таблиця 1 – Порівняння архітектур процесорів

Характеристика	x86	ARM	x86S (x86 Simplified)
Походження	Intel (з 1978 р.)	Acorn/ARM Ltd (з 1983 р.)	Intel (2023 р.)
Цільове середовище	ПК, сервери	Мобільні пристрої, IoT	Сучасні ПК, спрощення для x86
Енергоефективність	Низька	Висока	Краща за x86, але нижча за ARM
Комплексність інструкцій	CISC	RISC	CISC (зі спрощеннями)
Зворотна сумісність	Висока (аж до 16-біт DOS)	Обмежена	Немає з 16/32-біт, лише 64-біт підтримка
Підтримка ОС	Windows, Linux	Android, Linux, iOS	Windows 11+, Linux
Продуктивність на ватт	Низька	Висока	Середня
Придатність для Android	Потребує емуляції	Нативна	Теоретично можлива, але практично – ні
Масштабованість	Добра для серверів	Від мікроконтролерів до серверів	Обмежена поки тільки новими CPU Intel

Таблиця 2 – Порівняння головних характеристик ігрових рушіїв

Характеристика	Unity	Unreal Engine	Godot	Defold	Cocos2d-x
Ліцензія	Пропрієтарна	Пропрієтарна	Open Source	Open Source	Open Source
Мова програмування	C#, JS	C++/Blueprints	GDScript/C#	Lua	C++
Підтримка 3D	Так	Так	Так	Ні	Обмежено
Вага APK/інсталятора	Велика (10–40+ МБ)	Дуже велика (50–150 МБ)	Мала (5–20 МБ)	Дуже мала (<5 МБ)	Середня (5–20 МБ)
Швидкість запуску	Середня	Низька	Висока	Дуже висока	Висока
Робота напряму на Android	Через рушій	Через рушій	Через рушій	Через рушій	Через рушій
Продуктивність	Середня	Висока	Висока	Висока	Висока
Навчальна крива	Помірна	Висока	Низька	Низька	Середня
Відкрите API	Частково	Так	Так	Так	Так
Контроль над системою	Обмежений	Частковий	Високий	Обмежений	Високий

взаємодією з платформою. Це відкриває нові перспективи для розробки легковагових ігор та системного програмного забезпечення на ARM, особливо у сегменті low-end пристроїв та автономних ігрових рішень.

Порівняльний аналіз також свідчить, що хоча навчальна крива у роботі з таким ядром є вищою, отриманий рівень оптимізації, продуктивності та незалежності перевершує класичні рушії у низці ключових аспектів.

Аналіз сучасних архітектур та ігрових рушіїв засвідчує, що попри гнучкість, функціональність і візуальні можливості традиційних рушіїв, жоден з них не створений з урахуванням прямої установки на ARM-пристрої під Android без посередництва рушія або віртуального середовища. Всі популярні рушії залишаються десктоп-орієнтованими за своєю структурою, хоч і мають мобільні збірки, однак потребують складної абстракції між ігровою логікою та платформою. Натомість власне ігрове ядро, розроблене спеціально для прямої роботи на Android у середовищі ARM, усуває ці бар'єри, пропонуючи надзвичайно високу продуктивність, мінімальний обсяг застосунку, миттєвий запуск і повний контроль над апаратною частиною. Це робить його перспективним варіантом для створення low-level ігор, навчальних систем, інструментів розробника або навіть мікро-ОС для ігрових застосунків. У майбутньому розвиток таких ядер може закласти фундамент нової ніші в ігровій індустрії – ядроцентричних платформ, де рушій вже не є необхідністю, а лише варіантом.

Таблиця 3 ілюструє порівняльний аналіз основних компонентів популярних ігрових рушіїв – Unity, Unreal Engine, Godot, Defold, Cocos2d-x. У таблиці відображено наявність або відсутність критично важливих функціональних блоків, таких як рендеринг, фізика, UI, скриптинг, підтримка сенсорів, можливість гарячого перезавантаження тощо. Також оцінено загальний розмір рушія та рівень доступу до низькорівневих системних ресурсів. Це дозволяє виявити ключові переваги пропонованого рішення, зокрема його мінімалізм, модульність і орієнтованість на мобільну продуктивність без втрати контролю над апаратним забезпеченням.

Java є офіційно підтримуваною мовою для Android-розробки, яка компілюється в байт-код і виконується на віртуальній машині Android Runtime (ART) (табл. 4). Завдяки повній інтеграції з Android Studio та широкій екосистемі бібліотек, Java залишається зручною у використанні, особливо для новачків. Безпека пам'яті в Java реалізована через автоматичне збирання сміття (GC), що мінімізує витоки, хоча може спричиняти паузи при високому навантаженні [17]. Продуктивність Java дещо нижча порівняно з нативними мовами через додаткові накладні витрати віртуальної машини [18], однак вона входить до числа найенергоефективніших мов [19].

Kotlin, як сучасна альтернатива Java, також працює на ART і є повністю підтримуваною Google [20]. Вона має кращу ергономіку коду та покращену безпеку обробки null-значень, що призводить до меншої кількості збоїв у застосунках.

C++ використовується через Android NDK для задач, де критичні продуктивність і доступ до апаратного рівня. Код компілюється безпосередньо для архітектури ARM, забезпечуючи максимальну швидкість виконання, однак складність розробки, відсутність автоматичного керування пам'яттю та зниження безпеки є суттєвими недоліками [18]. У той час як Java і Kotlin більше орієнтовані на високорівневу розробку, C++ дозволяє використовувати низькорівневі графічні API як OpenGL ES чи Vulkan без обгорток, що важливо для ігрової індустрії.

Rust, хоча й новіший у екосистемі Android, демонструє високу продуктивність і безпеку пам'яті завдяки системі запозичень і статичній перевірці [21].

Його підтримка в Android активно зростає, однак інтеграція ще потребує використання NDK та створення біндінгів до Java. Rust стає особливо приваб-

Таблиця 3 – Порівняльний аналіз основних компонентів популярних ігрових рушіїв

Характеристика	Unity	Unreal Engine	Godot	Defold	Cocos2d-x
Працює напряму на Android без рушія	Не реалізовано	Не реалізовано	Не реалізовано	Не реалізовано	Не реалізовано
Рендеринг OpenGL ES / Vulkan	Наявне	Наявне	Наявне	Наявне	Наявне
Вбудована фізика	Реалізовано	Реалізовано	Реалізовано	Реалізовано	Реалізовано
Вага APK < 5 МБ	Відсутнє	Відсутнє	Відсутнє	Реалізовано	Відсутнє
Підтримка Lua	Відсутнє	Відсутнє	Відсутнє	Реалізовано	Реалізовано
Повний контроль над ресурсами	Відсутнє	Частково	Реалізовано	Відсутнє	Реалізовано
Власна система ресурсів	Відсутнє	Реалізовано	Реалізовано	Реалізовано	Реалізовано
Простий UI-модуль	Реалізовано	Реалізовано	Реалізовано	Реалізовано	Відсутнє
Вбудована IDE	Відсутнє	Відсутнє	Реалізовано	Відсутнє	Відсутнє
Гаряча перезагрузка	Частково	Реалізовано	Реалізовано	Реалізовано	Відсутня
Скриптовий рушій	C#	Blueprints/C++	GDScript	Lua	C++/Lua
Доступ до сенсорів Android	Через API	Через API	Через API	Через API	Через API
Мінімум залежностей	Частково	Частково	Частково	Реалізовано	Реалізовано

Таблиця 4 – Порівняльний аналіз мов програмування для Android розробки

Критерій	Java (SDK)	Kotlin (SDK)	C++ (NDK)	Rust (NDK)
Продуктивність на ARM	Середня (JIT/AOT)	Подібна до Java	Висока (native)	Висока (native)
Інтеграція	Повна (Android SDK)	Повна (Android SDK)	Через NDK (JNI)	Обмежена (через NDK)
Безпека пам'яті	Автоматично: збирач сміття (GC)	Автоматично: GC, null safety	Вручну: malloc / free	Автоматично: borrow checker + owership
Простота розробки	Висока	Дуже висока	Низька	Середня
Бібліотеки	Багато високорівневих + NDK	Багато високорівневих, підтримка всіх Java бібліотек	Багато низькорівневих	Зростає (crates.io)
Енергоефективність	Середня–висока	Середня–висока	Дуже висока	Дуже висока

ливим для критично важливих компонентів, де потрібна стабільність, швидкодія та енергоощадність.

Таким чином, для створення перших версій мобільного застосунку доцільно використовувати **Java** (або **Kotlin**). Ці мови забезпечують повну інтеграцію з Android SDK, мають простий цикл розробки, автоматичне керування пам'яттю (через GC) і доступ до широкого спектру бібліотек та інструментів Android Studio. Це дозволяє зосередитися на функціональності, дизайні та стабільності без значних витрат на низькорівневу оптимізацію. За потреби підвищення продуктивності, окремі компоненти можна реалізувати на C++ (через Android NDK) або Rust. Такий підхід забезпечить ефективність критичних частин, зберігаючи основну логіку на Java або Kotlin.

Vulkan та OpenGL ES – це два основні графічні API, які використовуються в розробці Android-застосунків, зокрема в ігровій індустрії (табл. 5). Vulkan є низькорівневим API, що забезпечує значно вищу продуктивність завдяки ефективному багатопотоковому рендерингу та мінімальному навантаженню на CPU. Наприклад, у грі «The Machines» на Huawei Mate 9 при використанні OpenGL ES спостерігалося близько 17 FPS, тоді як з Vulkan – понад 35 FPS [22]. Натомість OpenGL ES має простішу однопотокову архітектуру, яка легша в реалізації, але менш ефективна при складних сценах. Щодо сумісності, OpenGL ES підтримується практично всіма Android-пристроями, включаючи старі моделі. Vulkan же доступний лише з Android 7 (API 24), однак уже підтримується понад 85% сучасних пристроїв, особливо 64-бітних з Android 10 і вище [23].

Таблиця 5 – Порівняльний аналіз основних компонентів популярних ігрових рушіїв

Критерій	OpenGL ES	Vulkan
Продуктивність	Нижча (вищі затримки)	Вища (низький cpu overhead)
Сумісність пристроїв	Дуже висока (всі Android)	~85% пристроїв (Android 7+)
Енергоспоживання	Вище (неефективне)	Нижче (~15% економія)
Складність	Низька (простий API)	Висока (низькорівнева робота)
Можливості оптимізації	Обмежені стандартом	Широкі (багатоядерність, тощо)

За енергоспоживанням Vulkan також має перевагу: завдяки можливості ефективно розподіляти навантаження між ядрами процесора, він забезпечує до 15 % економії енергії в середньому, а в окремих випадках – до 400 % [22]. Попри це, розробка з Vulkan суттєво складніша: вона вимагає глибокого розуміння архітектури GPU, ручного керування ресурсами та побудови графічного pipeline. Натомість OpenGL ES завдяки вищому рівню абстракції дозволяє швидше створювати прототипи та простіші графічні додатки. Vulkan дає доступ до широких можливостей оптимізації, включно з bindless-текстуруванням, декількома GPU-командними чергами та трасуванням променів, що дозволяє створювати сучасну графіку з високою продуктивністю, тоді як OpenGL ES обмежений базовим функціоналом і повільнішими циклами оновлення.

На початковому етапі доцільно обрати OpenGL ES завдяки його простоті, широкій підтримці на більшості пристроїв та швидкому запуску. Проте при масштабуванні й переході на сучасні 64-бітні Android-пристрої варто розглянути Vulkan – він надає значно вищу продуктивність, краще енергоспоживання і глибший контроль над GPU. Це особливо актуально для застосунків, де графічна складність чи FPS є критично важливими.

Висновки. У результаті проведеного дослідження було встановлено, що сучасні універсальні ігрові рушії, попри свою багатофункціональність, не призначені для прямої роботи на Android-пристроях без проміжного середовища. Вони залишаються переважно десктоп-орієнтованими рішеннями з численними абстракціями, що знижують контроль над апаратним рівнем, збільшують вагу застосунків і сповільнюють запуск. Навпаки, розробка власного ігрового ядра з прямою інсталяцією на ARM-пристрої з Android відкриває нові можливості – повний контроль над ресурсами, мінімізація залежностей, висока продуктивність і енергоефективність, що особливо важливо для low-end пристроїв і автономних застосунків. Порівняльний аналіз архітектур (x86, ARM, x86S), мов програмування (Java, Kotlin, C++, Rust), графічних API (OpenGL ES, Vulkan), а також популярних рушіїв (Unity, Unreal, Godot, Defold, Cocos2d-x) довів доцільність створення

легковагових кастомних рішень у контексті стрімкої мобілізації ринку. Хоча створення власного ядра вимагає глибших технічних знань і має вищу навчальну криву, такий підхід виправданий у випадках, де критичні оптимізація, розмір, швидкість запуску та апаратна керованість. Таким чином, розробка власного ядра для прямої інсталяції на Android не лише є технічно можливою, але й стратегічно доцільною для певних ніш – інді-розробки, освітніх ігор, експериментальних платформ, системних утиліт. У перспективі такий підхід може закласти основу для нової генерації «ядроцентричних» мобільних рушіїв, де рушій є не універсальним фреймворком, а компактним, високоефективним середовищем, максимально адаптованим до платформи виконання.

Список використаної літератури

1. Keil J., Edler D., Schmitt T., Dickmann F. Creating Immersive Virtual Environments Based on Open Geospatial Data and Game Engines. *Journal of Cartography and Geographic Information*. 2021. Vol. 71. P. 53–65. DOI: https://doi.org/10.1007/s42489-020-00069-6.
2. Osborne P., Nõmm H., Freitas A.. A Survey of Text Games for Reinforcement Learning Informed by Natural Language. *Transactions of the Association for Computational Linguistics*. 2022. Vol. 10. P. 873–887. DOI: https://doi.org/10.1162/tacl_a_00495.
3. Carlos Marín-Lora, Miguel Chover, José M. Sotoca, Luis A. García. A game engine to make games as multi-agent systems. *Advances in Engineering Software*. 2020. Vol. 140. Article 102732. DOI: https://doi.org/10.1016/j.advengsoft.2019.102732.
4. Escobar-Castillejos D., Noguez J., Cárdenas-Ovando R.A., Neri L., Gonzalez-Nucamendi A., Robledo-Rella V. Using Game Engines for Visuo-Haptic Learning Simulations. *Applied Sciences*. 2020. Vol. 10. Article 4553. DOI: https://doi.org/10.3390/app10134553.
5. Vohera C., Chheda H., Chouhan D., Desai A., Jain V. Game Engine Architecture and Comparative Study of Different Game Engines. 2021 *12th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, Kharagpur, India. 2021. P. 1–6, DOI: 10.1109/ICCCNT51525.2021.9579618.
6. Jangra S., Singh G., Mantri A., Angra S. and Sharma B. Interactivity Development Using Unity 3D Software and C # Programming. *14th International Conference on Computing Communication and Networking Technologies (ICCCNT)*. Delhi, India, 2023. P. 1–6. DOI: 10.1109/ICCCNT56998.2023.10308030.
7. Zhang C., Yu S., Ji J. CFI: a VR motor rehabilitation serious game design framework integrating rehabilitation function and game design principles with an upper limb case. *J. NeuroEngineering Rehabil*. 2024. Vol. 21, artcle 113. DOI: https://doi.org/10.1186/s12984-024-01373-2.
8. Chirita A., Neluș-Constantin B., Dragos C. *Intel x86 and ARM processors : A survey on architectural differences*. 2022. URL: https://www.researchgate.net/publication/362105591_Intel_x86_and_ARM_processors_A_survey_on_architectural_differences (accessed: 12.05.2025).
9. Bibi S., Asghar M., Chaudhry M. U., Hussan N., Yasir M. A Review of ARM Processor Architecture History, Progress and Applications. *Journal of Applied and Emerging Sciences*. 2021. Vol. 10, no. 02. P. 171–179. URL: https://www.researchgate.net/publication/354193451_A_Review_of_ARM_Processor_Architecture_History_Progress_and_Applications (accessed: 12.05.2025).
10. *Intel Developer Zone. Envisioning the Future*: Simplified Architecture. 2023. URL: https://www.intel.com/content/www/us/en/developer/articles/technical/envisioning-future-simplified-architecture.html (accessed: 10.05.2025).
11. Tom's Hardware. *Intel Terminates x86S Initiative: Unilateral Quest to De-bloat x86 Instruction Set Comes to an End*. 2023. URL: https://www.tomshardware.com/pc-components/cpus/intel-terminates-x86s-initiative-unilateral-quest-to-de-bloat-x86-instruction-set-comes-to-an-end (accessed: 05.2025).
12. *Unity Technologies. Compare Plans*. URL: https://unity.com/products/compare-plans (accessed: 10.05.2025).
13. *Epic Games. Unreal Engine End User License Agreemen*t. URL: https://www.unrealengine.com/en-US/license (accessed: 10.05.2025).
14. *Godot Engine. Godot – Open Source Game Engine [GitHub]*. URL: https://github.com/godotengine/godot (accessed: 10.05.2025).
15. *Defold Foundation. Defold – Game Engine [GitHub]*. URL: https://github.com/defold (accessed: 10.05.2025)..
16. *Cocos2d-x.org. Cocos2d-x [GitHub]*. URL: https://github.com/cocos2d/cocos2d-x (accessed: 10.05.2025).
17. *Google Security Blog. Memory-safe languages in Android 13 .2022*. URL: https://security.googleblog.com/2022/12/memory-safe-languages-in-android-13.html (accessed: 10.05.2025)..
18. *Android Authority. Java vs C++ App Performance*. URL: https://www.androidauthority.com/java-vs-c-app-performance-689081 (accessed: 10.05.2025).
19. *LeanIX Engineering. Sustainable Green Software Engineering*. URL: https://engineering.leanix.net/blog/sustainable-green-software-engineering (accessed: 10.05.2025).
20. *Android Developers. Kotlin for Android Development*. URL: https://developer.android.com/kotlin (accessed: 10.05.2025)..
21. *Android Source. Building Rust Modules for Android*. URL: https://source.android.com/docs/setup/build/rust/building-rust-modules/overview (accessed: 10.05.2025)..
22. *ARM Community. Initial Comparison of Vulkan API vs OpenGL ES API on ARM*. URL: https://community.arm.com/arm-community-blogs/b/mobile-graphics-and-gaming-blog/posts/initial-comparison-of-vulkan-api-vs-opengl-es-api-on-arm (accessed: 10.05.2025).
23. *Android Developers. Vulkan API Overview*. URL: https://developer.android.com/games/develop/vulkan/overview (accessed: 10.05.2025).

References (transliterated)

1. Keil J., Edler D., Schmitt T., Dickmann F. Creating Immersive Virtual Environments Based on Open Geospatial Data and Game Engines. *Journal of Cartography and Geographic Information*. 2021, vol. 71, pp.53–65. DOI: https://doi.org/10.1007/s42489-020-00069-6.
2. Osborne P., Nõmm H., Freitas A.. A Survey of Text Games for Reinforcement Learning Informed by Natural Language. *Transactions of the Association for Computational Linguistics*. 2022, vol. 10, pp. 873–887. DOI: https://doi.org/10.1162/tacl_a_00495.
3. Carlos Marín-Lora, Miguel Chover, José M. Sotoca, Luis A. García. A game engine to make games as multi-agent systems. *Advances in Engineering Software*. 2020, vol. 140, artcle 102732. DOI: https://doi.org/10.1016/j.advengsoft.2019.102732.
4. Escobar-Castillejos D., Noguez J., Cárdenas-Ovando R.A., Neri L., Gonzalez-Nucamendi A., Robledo-Rella V. Using Game Engines for Visuo-Haptic Learning Simulations. *Applied Sciences*. 2020, vol. 10, artcle 4553. DOI: https://doi.org/10.3390/app10134553.
5. Vohera C., Chheda H., Chouhan D., Desai A., Jain V. Game Engine Architecture and Comparative Study of Different Game Engines. 2021 *12th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, Kharagpur, India, 2021, pp. 1–6, DOI: 10.1109/ICCCNT51525.2021.9579618.
6. Jangra S., Singh G., Mantri A., Angra S. and Sharma B. Interactivity Development Using Unity 3D Software and C # Programming. *14th International Conference on Computing Communication and Networking Technologies (ICCCNT)*. Delhi, India, 2023, pp. 1–6. DOI: 10.1109/ICCCNT56998.2023.10308030.
7. Zhang C., Yu S., Ji J. CFI: a VR motor rehabilitation serious game design framework integrating rehabilitation function and game design principles with an upper limb case. *J. NeuroEngineering Rehabil*. 2024, vol. 21, article 113. DOI: https://doi.org/10.1186/s12984-024-01373-2.
8. Chirita A., Neluș-Constantin B., Dragos C. *Intel x86 and ARM processors : A survey on architectural differences*. 2022. Available. at: https://www.researchgate.net/publication/362105591_Intel_x86_and_ARM_processors_A_survey_on_architectural_differences (accessed: 12.05.2025).
9. Bibi S., Asghar M., Chaudhry M. U., Hussan N., Yasir M. A Review of ARM Processor Architecture History, Progress and Applications. *Journal of Applied and Emerging Sciences*. 2021, vol. 10, no. 02, pp. 171–179. Available at: https://www.researchgate.net/publication/354193451_A_Review_of_ARM_Processor_Architecture_History_Progress_and_Applications (accessed: 12.05.2025).

10. *Intel Developer Zone. Envisioning the Future: Simplified Architecture*. 2023. Available at: https://www.intel.com/content/www/us/en/developer/articles/technical/envisioning-future-simplified-architecture.html (accessed: 10.05.2025).
11. Tom's Hardware. *Intel Terminates x86S Initiative: Unilateral Quest to De-bloat x86 Instruction Set Comes to an End*. 2023. Available at: https://www.tomshardware.com/pc-components/cpus/intel-terminates-x86s-initiative-unilateral-quest-to-de-bloat-x86-instruction-set-comes-to-an-end (accessed: 05.2025).
12. *Unity Technologies. Compare Plans*. Available at: https://unity.com/products/compare-plans (accessed: 10.05.2025).
13. *Epic Games. Unreal Engine End User License Agreement*. Available at: https://www.unrealengine.com/en-US/license (accessed: 10.05.2025).
14. *Godot Engine. Godot – Open Source Game Engine [GitHub]*. Available at: https://github.com/godotengine/godot (accessed: 10.05.2025).
15. *Defold Foundation. Defold – Game Engine [GitHub]*. Available at: https://github.com/defold (accessed: 10.05.2025)..
16. *Cocos2d-x.org. Cocos2d-x [GitHub]*. Available at: https://github.com/cocos2d/cocos2d-x (accessed: 10.05.2025).
17. *Google Security Blog. Memory-safe languages in Android 13 .2022*. Available at: https://security.googleblog.com/2022/12/memory-safe-languages-in-android-13.html (accessed: 10.05.2025)..
18. *Android Authority. Java vs C++ App Performance*. Available at: https://www.androidauthority.com/java-vs-c-app-performance-689081 (accessed: 10.05.2025).
19. *LeanIX Engineering. Sustainable Green Software Engineering*. Available at: https://engineering.leanix.net/blog/sustainable-green-software-engineering (accessed: 10.05.2025).
20. *Android Developers. Kotlin for Android Development*. Available at: https://developer.android.com/kotlin (accessed: 10.05.2025)..
21. *Android Source. Building Rust Modules for Android*. Available at: https://source.android.com/docs/setup/build/rust/building-rust-modules/overview (accessed: 10.05.2025)..
22. *ARM Community. Initial Comparison of Vulkan API vs OpenGL ES API on ARM*. Available at: https://community.arm.com/arm-community-blogs/b/mobile-graphics-and-gaming-blog/posts/initial-comparison-of-vulkan-api-vs-opengl-es-api-on-arm (accessed: 10.05.2025).
23. *Android Developers. Vulkan API Overview*. Available at: https://developer.android.com/games/develop/vulkan/overview (accessed: 10.05.2025).

Надійшла (received) 16.05.2025

UDC 004.432.3

V. V. YATSENKO, Candidate of Technical Sciences (PhD), Docent, Associate Professor at the Department of Economic Cybernetics, Sumy State University, Sumy, Ukraine; e mail: v.yatsenko@biem.sumdu.edu.ua; ORCID: https://orcid.org/0000-0003-2316-3817
D. V. PARKHOMENKO, Higher Education Student, Sumy State University, Sumy, Ukraine;
e mail: dmytro.parkhomenko@student.sumdu.edu.ua; ORCID: https://orcid.org/0009-0003-5279-1879
O. S. KUSHNEROV, Doctor of Philosophy (PhD), Senior Lecturer at the Department of Economic Cybernetics, Sumy State University, Sumy, Ukraine; e mail: o.kushnerov@biem.sumdu.edu.ua; ORCID: https://orcid.org/0000-0001-8253-5698
V. V. KOIBICHUK, Candidate of Economic Sciences (PhD), Docent, Head of the Economic Cybernetics Department, Sumy State University, Sumy, Ukraine; e mail: v.koibichuk@biem.sumdu.edu.ua; ORCID: https://orcid.org/0000-0002-3540-7922
K. H. HRYTSENKO, Candidate of Technical Sciences (PhD), Docent, Associate Professor at the Department of Economic Cybernetics, Sumy State University, Sumy, Ukraine; e mail: k.hrytsenko@biem.sumdu.edu.ua; ORCID: https://orcid.org/0000-0002-7855-691X

COMPARISON OF MODERN GAME ENGINES WITH A CUSTOM CORE FOR NATIVE GAME DEVELOPMENT ON THE ANDROID PLATFORM

The modern game industry is increasingly focused on mobile platforms, particularly devices based on the ARM architecture, which dominates the smartphone and tablet markets. Developers are actively adapting their engines and tools to this architecture, taking into account its energy efficiency and widespread adoption. In this context, the development of a custom game core that can be installed directly on an Android device without the need for additional engines opens new opportunities for optimization, faster prototyping, and full control over device-level performance. This approach is especially relevant in light of the growing popularity of independent game development and the demand for lightweight solutions without unnecessary dependencies. This study presents a comparative analysis of modern game engines (Unity, Unreal Engine, Godot, Defold, Cocos2d-x) and a custom-developed game core designed for direct installation and execution on Android devices with ARM architecture, without relying on any intermediate engine. The paper examines the advantages of ARM architecture, including energy efficiency, scalability, and broad support in mobile devices, making it a suitable platform for native game development. Particular attention is paid to the technical comparison of engine capabilities, including application size, launch speed, API flexibility, access to system resources, and support for low-level languages. It has been revealed that although traditional engines offer extensive functionality and ease of development, they limit hardware-level control and significantly increase APK size. On the other hand, a custom core, specifically designed for ARM devices, provides minimal size, instant launch, and maximum performance due to direct access to graphical APIs (OpenGL ES/Vulkan) and Android system resources. The study also analyzes the suitability of programming languages such as Java, Kotlin, C++, and Rust for Android game development. It outlines the potential of Vulkan as a high-performance graphics API and discusses the feasibility of a core-centered approach for creating lightweight, optimized mobile games and tools.

Keywords: game engine, ARM, Android, Vulkan, low-level development, Java, C++, Rust, energy efficiency, mobile optimization.

Повні імена авторів / Author's full names

Автор 1 / Author 1: Яценко Валерій Валерійович / Yatsenko Valerii Valeriiovych
Автор 2 / Author 2: Пархоменко Дмитро Володимирович/ Parkhomenko Dmytro Volodymyrovych
Автор 3 / Author 3: Кушнерьов Олександр Сергійович / Kushnerov Oleksandr Serhiiovych
Автор 4 / Author 4: Койбічук Віталія Василівна / Koibichuk Vitaliia Vasylivna
Автор 5 / Author 5: Гриценко Костянтин Григорович / Hrytsenko Kostiantyn Hryhorovych