2. Unity Manual Physics. URL: https://docs.unity3d.com/Manual/PhysicsSection.html (дата звернення: 20.03.2026).
3. Introduction to Game Development with Unity. Unity Learn. URL: https://learn.unity.com/ (дата звернення: 19.03.2026).
4. Не тільки для геймерів: Як Unity змінює різні індустрії. *Robot Dreams*. 2024. URL: https://robotdreams.cc/uk/blog/705-how-unity-became-a-tool-for-everything (дата звернення: 20.03.2026).

УДК 004.4

ВИКОРИСТАННЯ БАГАТОПОТОЧНОСТІ В СУЧАСНИХ ІГРОВИХ РУШІЯХ (НА ПРИКЛАДІ UNITY)

Різник О.М.
alex07082005m@gmail.com
Черкаський державний технологічний університет, Україна
Метелап В.В.
м. Черкаси, Україна

Сучасні ігрові застосунки належать до класу систем реального часу, для яких критичною є здатність обробляти великі обсяги даних із мінімальними затримками. Зі зростанням складності ігрових сцен, кількості об’єктів, штучного інтелекту та фізичних симуляцій, продуктивність однопотокових обчислень стає недостатньою. У зв’язку з цим актуальним є використання багатопоточності, що дозволяє ефективно задіяти багатоядерні процесори.

Метою даної роботи є аналіз застосування багатопоточності в сучасних ігрових рушіях на прикладі Unity, а також визначення основних підходів до розпаралелення обчислень.

Потік виконання (thread) є мінімальною одиницею виконання програми, яка може працювати паралельно з іншими потоками в межах одного процесу. Використання декількох потоків дозволяє одночасно виконувати незалежні частини програми, що є основою паралельних обчислень.

Слід розрізняти поняття конкурентності та паралелізму. Конкурентність означає здатність системи працювати з кількома задачами, тоді як паралелізм передбачає їх одночасне виконання на різних обчислювальних ядрах/процесорах.

Разом з перевагами багатопоточність створює низку проблем, серед яких:

- стани гонки (race conditions) – некоректний доступ до спільних даних;
- взаємне блокування (deadlock) – ситуація, коли потоки очікують один одного;
- накладні витрати синхронізації – зниження продуктивності через використання механізмів блокування.

Таким чином, ефективне використання багатопоточності потребує спеціальних архітектурних підходів.

Ігрові рушії реалізують так званий ігровий цикл (game loop), який включає обробку логіки, фізики та рендеринг. У класичному підході ці етапи виконуються послідовно в одному потоці, що створює вузьке місце (bottleneck).

Для підвищення продуктивності сучасні рушії використовують багатопоточність для розподілу задач:

- обробка штучного інтелекту;
- фізичні симуляції;
- анімація;
- завантаження ресурсів;
- обчислення великої кількості об’єктів.

Такий підхід дозволяє значно зменшити час виконання кадру та забезпечити стабільну частоту оновлення екрану.

Unity є одним із найпоширеніших ігрових рушіїв, який реалізує сучасні підходи до паралелізації через набір технологій, відомий як Data-Oriented Technology Stack (DOTS).

Unity має свій набір інструментів для оптимізації роботи додатків та прискорення їх роботи через паралельні процеси.

Unity Job System дозволяє розбивати обчислення на невеликі незалежні задачі (jobs), які можуть виконуватися паралельно на різних ядрах процесора. Такі задачі описуються як структури та виконуються через спеціальний планувальник.

Burst – це компілятор, що перетворює проміжний код .NET у високоефективний машинний код із використанням оптимізацій LLVM. Він призначений для роботи з Job System та дозволяє значно підвищити продуктивність CPU-обчислень.

Зокрема, розбиття задач на незалежні job-и та їх компіляція за допомогою Burst дозволяє «ефективно використовувати всі доступні ядра процесора».

ECS є архітектурним підходом, орієнтованим на дані. Він відокремлює дані (компоненти) від логіки (системи), що дозволяє обробляти великі масиви об’єктів більш ефективно.

Згідно з документацією Unity, ECS дозволяє:

- масштабувати симуляції до великої кількості об’єктів;
- ефективно використовувати ресурси процесора та пам’яті;
- підвищити продуктивність завдяки data-oriented підходу.

Також ECS працює у зв’язці з Job System та Burst Compiler, що забезпечує максимальну ефективність використання апаратних ресурсів.

Окрім багатопоточності, Unity підтримує асинхронні механізми (coroutines, async/await), які дозволяють виконувати операції без блокування основного потоку, зокрема:

- завантаження сцен;
- підвантаження ресурсів;
- мережеві запити.

Багатопоточність у Unity використовується для вирішення ряду практичних задач:

- обробка великої кількості NPC;
- процедурна генерація світу;

– фізичні симуляції;
– потокове завантаження контенту.

Застосування ECS дозволяє обробляти «великомасштабні симуляції та значну кількість сутностей» , що є критичним для сучасних ігор.

Попри переваги, багатопоточність у Unity має ряд обмежень:

– більшість API доступна лише з головного потоку;
– складність синхронізації між потоками;
– труднощі відлагодження;
– накладні витрати при надмірній паралелізації.

Крім того, неправильне використання потоків може призвести до зниження продуктивності замість її покращення.

Багатопоточність є ключовим інструментом підвищення продуктивності сучасних ігрових рушіїв. Вона дозволяє ефективно використовувати багатоядерні процесори та забезпечувати стабільну роботу ігор у реальному часі.

На прикладі Unity показано, що ефективна реалізація багатопоточності досягається не лише створенням потоків, а й використанням спеціалізованих архітектурних підходів, таких як Job System, Burst Compiler та ECS.

Таким чином, сучасні ігрові рушії переходять від класичних об’єктно-орієнтованих моделей до data-oriented підходів, що дозволяє досягти значного приросту продуктивності.

Список використаних джерел:

1. Unity Technologies. ECS for Unity. URL: https://unity.com/ecs (дата звернення: 24.04.2026).
2. Unity Technologies. Burst Compiler Documentation. URL: https://docs.unity3d.com (дата звернення: 24.04.2026).
3. Unity Technologies. Job System and ECS Tutorial. URL: https://learn.unity.com (дата звернення: 24.04.2026).
4. Unity Technologies. Data-Oriented Technology Stack (DOTS). URL: https://unity.com (дата звернення: 24.04.2026).