DOI https://doi.org/10.32782/2078-0877-2026-26-2-6
УДК 004.932:004.93’1:004.272

Є. В. Міська, аспірант ORCID: 0009-0008-6719-2701
Державний університет імені Івана Франка
e-mail: misko-y@zu.edu.ua

АРХІТЕКТУРНІ ПІДХОДИ ДО РОЗШИРЕННЯ КІНЕМАТИКИ ПЕРСОНАЖА В СЕРЕДОВИЩІ UNREAL ENGINE

Анотація. У статті досліджено архітектурні обмеження базового класу UCharacterMovementComponent в Unreal Engine. Виявлені обмеження критичні для мережевих систем, де непередбачувані розриви кінематики ускладнюють алгоритми екстраполяції та компенсації затримки. Запропоновано підхід, заснований на перехопленні фізичних розрахунків під час колізії та використанні векторної алгебри замість непередбачуваної фізики твердих тіл. Розроблено математичну модель збереження імпульсу: використовуючи скалярний та векторний добутки нормалі поверхні, алгоритм обчислює дотичну траєкторію без втрати початкової швидкості. Сформульовано диференціальну модель контролю швидкості (Soft Cap), яка замінює базовий ліміт MaxWalkSpeed на алгоритм постійного асимптотичного зниження імпульсу. Інтеграція методу з проекцією вектора гравітації дозволила реалізувати фізично коректне накопичення інерції залежно від топології рівня. Практична імплементація довела перевагу Data-Driven підходу, забезпечивши збереження плавності кінематичних функцій та повну алгоритмічну керованість поведінки контролера.

Ключові слова: Unreal Engine, архітектура програмних рушіїв, кінематика персонажа, векторна алгебра, збереження імпульсу, локомоція, розробка систем.

Постановка проблеми. У проектуванні динамічних систем руху персонажа (локомоції) головним інженерним питанням є детерміноване управління кінематикою, яке безпосередньо впливає на чутливість керування (responsiveness) та узгодженість станів у розподіленому середовищі. Компонент руху є зв'язком між дискретним введенням користувача та неперервною зміною стану об'єкта в тривимірному просторі. Питання полягає у вирішенні конфлікту між швидкістю реакції на керування (responsiveness) та необхідністю підтримки реалістичної траєкторії руху. Для динамічних проектів, де динамічна система базується на інерції (momentum-based movement), система повинна не просто переміщувати об’єкт, а й коректно зберігати та трансформувати його кінетичну енергію під час взаємодії з оточенням.

У сучасних програмних рушіях, зокрема в Unreal Engine, локомоція зазвичай спирається на кінематичні контролери (Character Controllers), а не на повну динамічну симуляцію твердих тіл (Rigid Body Dynamics). Такий підхід потрібен для абсолютного контролю над поведінкою персонажа та детермінованості обчислень для алгоритмів мережевої синхронізації [1–3]. Використання кінематичних перевірок дозволяє уникнути типових проблем фізичних рушіїв, як нумерична нестабільність симуляції (jittering) та недетерміновані реакції при зіткненні, проте воно накладає суворі обмеження на обробку імпульсів.

Базовий модуль Unreal Engine – клас UCharacterMovementComponent (CMC) – це складна система, оптимізована під стандартні сценарії: ходьбу, біг та прості стрибки. Проте внутрішня логіка CMC має спрямованість на стандартні кінематичні моделі ухил. Наприклад, під час обробки зіткнень із вертикальними поверхнями рушій застосовує функцію SlideAlongSurface, яка автоматично обнуляє вектор швидкості, спрямований у бік перешкоди. Для користувача це виглядає як миттєва зупинка при торканні стіни під гострим кутом, що повністю руйнує кінематичний процес у проектах, де локомоція має бути безперервною.

Стандартна реалізація обмежень швидкості в CMC працює за принципом примусового обрізання (Hard Clamp). Як тільки швидкість перевищує ліміт MaxWalkSpeed, система примусово обрізає вектор, не враховуючи джерела прискорення. У контексті розробки розширеної кінематики, такої як Wallrun або високошвидкісне ковзання, такі обмеження стають критичним бар'єром. Виникає потреба в модифікації життєвого циклу оновлення стану персонажа (Physics Tick), де розробник може втрутитися в розрахунки до того, як рушій застосовує свої стандартні методи гасіння інерції. Це вимагає розробки архітектурних надбудов, які дозволять системі динамічно розрізняти типи поверхонь та адаптувати вектори руху в реальному часі, зберігаючи при цьому цілісність колізійної моделі.

Аналіз останніх досліджень. Проблематика розширення кінематики віртуального персонажа активно розглядається. В офіційній документації Epic Games детально описано принципи функціонування базового класу UCharacterMovementComponent (CMC) [4, 5]. Зазначається, що стандартна модель непружних зіткнень рушія застосовує алгоритм ковзання вздовж поверхні, який автоматично відсікає вектор швидкості, перпендикулярний до перешкоди.

У дослідженнях S. Prasanth розглядається вибір між Rigid Body та кінематичними контролерами [6]. Доведено, що фізичні системи на базі сил часто виявляються непередбачуваними, оскільки на них впливають невеликі неточності кроку симуляції. Такі фізичні симуляції також важливо передбачити (екстраполювати) [7–9]. Натомість Data-Driven підхід із жорстким маніпулюванням векторами швидкості залишається стандартом індустрії [10].

Підходи до створення відчуття інерції без використання чистої фізики мас описані в матеріалах GDC на прикладі розробки обчислювального середовища Mirror's Edge [11]. Ключовим фактором названо алгоритмічне збереження кінетичної енергії на рівні векторів. Еволюція підходів до локомоції в Unreal Engine також стимулювала розробку нового експериментального плагіна Mover [12], який пропонує модульну систему станів замість монолітного коду CMC.

Формулювання цілей статті (постановка завдання). Мета роботи полягає у формулюванні архітектурних розширень базового компонента руху Unreal Engine для подолання його кінематичних обмежень та забезпечення детермінованого збереження імпульсу умовного віртуально гоперсонажа. Це передбачає створення алгоритмів збереження імпульсу та впровадження детермінованих моделей контролю швидкості, що дозволяють обійти обмеження стандартних кінематичних контролерів без переходу до високонавантажених симуляцій фізики твердих тіл.

Для досягнення мети необхідно вирішити такі архітектурні проблемні моменти на рівні базового компонента руху ігрового рушія:

1. Дослідити механізм імпульсного гасіння вбудованого контролера руху при взаємодії з вертикальною геометрією та сформулювати модель, яка забезпечує детерміноване збереження кінетичної енергії при виконанні маневру – Wallrun.

2. Замінити базове жорстке обрізання швидкості системним лімітом MaxWalkSpeed на алгоритмічну модель поступового зниження імпульсу (Soft Cap).

3. Розробити алгоритм перевірки простору для безпечної динамічної зміни розмірів колізійного обмежувача під час трансформацій (Shape-shifting), усунувши ризик проникнення крізь геометрію.

Основна частина. Аналіз обмежень базової архітектури UCharacterMovementComponent. Підсистема руху в Unreal Engine працює за принципом дискретних часових кроків. Ключовим вузьким місцем для динамічної локомоції є фаза виконання руху. Під час переміщення об'єкта рушій використовує геометричне зондування (Capsule Sweep). Якщо фіксується зіткнення, алгоритм перераховує вектор швидкості, видаляючи всю кінетичну енергію, що спрямована в бік перешкоди. При взаємодії зі стіною під гострим кутом це призводить до майже повної зупинки персонажа.

Алгебраїчна модель збереження імпульсу (Wallrun). Для реалізації механіки бігу по стіні зі збереженням інерції розроблено алгоритм перехоплення розрахунків у кадрі реєстрації колізії (Impact Frame) [10]. Ініціація стану відбувається через аналіз геометрії перешкоди. Система вилучає вектор нормалі поверхні стіни. Для перевірки кута наближення обчислюється скалярний добуток між вектором погляду та нормаллю стіни.

Основою алгоритму є знаходження дотичної траєкторії. Для цього застосовується векторний добуток (Cross Product) нормалі стіни та світового вектора вертикалі (Up Vector) [13]. Отриманий дотичний вектор вказує напрямок вздовж стіни. Оскільки він може бути спрямований у протилежний бік, виконується скалярне порівняння зі старим вектором швидкості користувача.

Відновлення кінетичної енергії відбувається шляхом множення нормалізованого дотичного вектора на модуль швидкості, що існував до удару. Додатково в алгоритм інтегровано вектор компенсації мікровідривів. До загального вектора швидкості додається локальна сила, спрямована проти нормалі стіни, що надійно притискає колізійну капсулу до поверхні.

Диференціальна модель деградації імпульсу (Soft Cap). Базовий клас руху використовує жорстке обрізання: якщо швидкість перевищує MaxWalkSpeed, модуль вектора миттєво прирівнюється до ліміту (Hard Clamp). Для розв'язання цієї проблеми розроблено підсистему поступового гасіння швидкості (Soft Cap). Замість жорсткого присвоєння використовується диференціальний алгоритм лінійної інтерполяції. Якщо швидкість перевищує ліміт, система розраховує нову цільову швидкість з урахуванням часу кадру та коефіцієнта деградації. Це забезпечує плавне зниження швидкості.

Поведінка контролера на схилах інтегрована з цією моделлю. Алгоритм проектує вектор гравітації на нормаль підлоги, отримуючи вектор гравітаційного ковзання. Скалярний добуток цього вектора з напрямком руху визначає контекст топології. При русі вниз до вектора швидкості додається прискорення, а коефіцієнт деградації обнуляється, імітуючи відсутність тертя. При русі вгору активується підвищений коефіцієнт гальмування.

Топологічна валідація при зміні габаритів (Shape-shifting). Динамічне розширення колізійного обмежувача (збільшення CapsuleHalfHeight) часто створює просторові проблеми. Базова реакція рушія на такий стан – агресивне виштовхування об'єкта, що призводить до провалювання крізь геометрію.

Проблема вирішується імплементацією предиктивної перевірки простору. Перед фактичною зміною параметрів колізії логіка контролера викликає метод геометричного тестування (OverlapBlockingTestByChannel). Система віртуально проектує збільшену капсулу у поточні координати персонажа. Трансформація превентивно відхиляється, якщо функція повертає наявність перетину з геометрією.

Методологія та результати експериментів. Для верифікації розроблених архітектурних підходів було створено тестовий полігон у середовищі Unreal Engine. Об'єктом дослідження виступав модифікований компонент UCharacterMovementComponent. Важливо було порівняння збереження кінетичної енергії між базовою моделлю локомоції (Default CMC) та розширеною архітектурою (Extended CMC із застосуванням векторного Wallrun та Soft Cap).

Експеримент проводився при частоті оновлення 60 Гц. Базова максимальна швидкість (MaxWalkSpeed) була встановлена на позначці 600 одиниць (uu/s). Тестовий маршрут включав розгін, стрибок під кутом 45 градусів на вертикальну площину та приземлення на схил із кутом 30 градусів. Заміри вектора швидкості проводилися у трьох фазах. Результати профілювання наведені у таблиці (табл. 1).

Таблиця 1

Порівняльний аналіз збереження кінетичної енергії
у базовій та розширеній архітектурі локомоції

Фаза тестування	Базова архітектура (Default CMC)	Розширена архітектура (Extended CMC)
Швидкість перед зіткненням	600 uu/s	600 uu/s
Швидкість вздовж стіни (Wallrun)	185 uu/s (Втрата 31 % інерції)	598 uu/s (Втрата 0.4 % інерції)
Приземлення на схил (0.0 сек)	210 uu/s (Швидкість падіння)	650 uu/s (Імпульс збережено)
Рух по схилу вниз (через 1.0 сек)	600 uu/s (Спрацював Hard Clamp)	940 uu/s (Робота Soft Cap)

Згідно з результатами, базова архітектура автоматично поглинає 69 % кінетичної енергії через застосування непружної колізії. При подальшому русі по схилу швидкість миттєво обмежується дефолтним лімітом (600 uu/s). Застосування розробленої моделі дозволило зберегти 99,6 % початкового імпульсу під час проходження вздовж стіни. На похилій поверхні алгоритм Soft Cap продемонстрував здатність накопичувати інерцію (швидкість зросла до 940 uu/s) без розривів кінематичної функції.

Висновки. Проведене дослідження підтверджує, що базова архітектура компонента руху в Unreal Engine має серйозні обмеження для систем високодинамічної локомоції. Вбудовані алгоритми обробки колізій неминуче призводять до втрати інерції персонажа. Запропоновані архітектурні підходи дозволяють обійти ці обмеження. Перехоплення розрахунків у момент колізії та застосування векторного аналізу простору забезпечило збереження майже 100 % кінетичної енергії при контакті з перешкодами. Інтеграція моделі поступового зниження імпульсу адаптувала кінематику до топології рівня, реалізувавши фізично-коректне ковзання на похилих поверхнях із приростом швидкості до 56 % понад базовий ліміт. Цей алгоритмічний підхід значно кращий за непередбачувану фізику твердих тіл при розробці контролерів персонажа. Перспективи подальших розвідок у даному напрямку полягають у декомпозиції розроблених алгоритмів для їх інтеграції в масові багатокористувацькі репліковані середовища, а також в оптимізації серверної частини коду задля забезпечення високої детермінованості станів у системах із великою кількістю одночасних підключень.

Список використаних джерел

1. Клейпул М., Клейпул К. Затримка та дії гравця в онлайн-іграх. *Communications of the ACM*. 2006. Т. 49, № 11. С. 40–45. DOI: https://doi.org/10.1145/1167838.1167860

2. Лю С. та ін. Огляд та таксономія методів компенсації затримки для мережевих комп'ютерних ігор. *ACM Computing Surveys*. 2022. Т. 54, № 11s. С. 1–34. DOI: https://doi.org/10.1145/3519023

3. Мотоо Т. та ін. Компенсація мережевої затримки на стороні клієнта для онлайн-шутерів. *IEEE Access*. 2021. Т. 9. С. 125674–125689. URL: https://scispace.com/pdf/client-side-network-delay-compensation-for-online-shooting-4m5jqc0ica.pdf

4. Компонент руху персонажа. Документація Unreal Engine. *Epic Developer Community*. URL: https://dev.epicgames.com/documentation/unreal-engine/movement-components-in-unreal-engine (дата звернення: 10.04.2026).

5. Довідник API Unreal Engine. Клас UCharacterMovementComponent. URL: https://dev.epicgames.com/documentation/unreal-engine/API/Runtime/Engine/UCharacterMovementComponent (дата звернення: 14.04.2026).

6. Прасант С. Великі дебати: Rigidbody проти Character Controller для руху гравця в Unity. *Medium*. 2023. URL: https://sivakumar-prasanth.medium.com/the-great-debate-rigidbody-vs-character-controller-for-player-movement-in-unity-c551a52ff340

7. Фідлер Г. Мережева фізика. *Gaffer On Games*. 2014. URL: https://gafferongames.com/categories/networked-physics/

8. Харитонов В. Ю. Адаптивний алгоритм екстраполяції для спільних віртуальних середовищ. *Proceedings of the 11th ACM SIGGRAPH International Conference*. 2012. С. 255–261.

9. Хамаданіан П. та ін. Ekho: Синхронізація медіа в хмарних іграх між декількома кінцевими точками. *ACM SIGCOMM 2023 Conference*. 2023. С. 1–17.

10. Векторна математика для початківців геймдеву. *Game Developer*. URL: https://www.gamedeveloper.com/programming/vector-maths-for-game-dev-beginners

11. О'Коннор Р. Створення руху від першої особи для Mirror's Edge. *GDC Vault*. 2017. URL: https://gdcvault.com/play/1387/Creating-First-Person-Movement-for

12. Особливості та концепції плагіна Mover в Unreal Engine. *Epic Games Developers*. URL: https://dev.epicgames.com/documentation/unreal-engine/mover-features-and-concepts-in-unreal-engine

13. Березький О. В. Методи та алгоритми комп'ютерної графіки в інформаційних системах. Тернопіль : ТНЕУ, 2018. 240 с.

Дата першого надходження статті до видання: 24.03.2026
Дата прийняття статті до друку після рецензування: 22.04.2026
Дата публікації (оприлюднення) статті: 25.05.2026

Ye. Misko
Zhytomyr Ivan Franko State University

ARCHITECTURAL APPROACHES TO EXTENDING CHARACTER KINEMATICS IN THE UNREAL ENGINE ENVIRONMENT

Summary

The article examines the architectural limitations of the base class UCharacterMovementComponent in Unreal Engine, specifically concerning complex kinematic movements and momentum preservation. While highly optimized for standard bipedal locomotion, this component exhibits rigid constraints during high-velocity, physics-based interactions. The identified limitations are critical for multiplayer networked systems, where unpredictable kinematic discontinuities and hard-coded collision responses severely complicate server-side extrapolation and client-side latency compensation algorithms, often leading to visual desynchronization in fast-paced scenarios.

To address these fundamental issues, an innovative architectural approach is proposed based on intercepting low-level physical calculations during collision events. Instead of relying on the engine's default, often unpredictable rigid body dynamics for character collision resolution, the proposed architecture utilizes precise vector algebra. A robust mathematical model of momentum preservation has been developed and integrated into the custom movement logic. By using the scalar and vector products of the impacted surface normal, the algorithm calculates the optimal tangent trajectory. This allows the virtual character to glide along complex geometry without the abrupt loss of initial kinetic energy that typically occurs in the base engine implementation upon hitting an obstacle.

Furthermore, a differential speed control model, defined as a “Soft Cap,” was formulated. This model systematically replaces the basic MaxWalkSpeed absolute limit, which traditionally truncates velocity vectors instantly and artificially. Instead, the Soft Cap employs an algorithm for the constant asymptotic reduction of momentum. This allows characters to temporarily exceed standard speed limits while smoothly decaying the excess velocity over time, drastically improving the fluidity of mechanics. Integrating this method with continuous gravity vector projection allowed for physically correct inertia accumulation strongly dependent on the level’s topology, dynamically translating gravitational pull into forward momentum on descents.

Practical implementation proved the significant superiority of the Data-Driven approach. It ensures absolute smoothness of kinematic functions and full algorithmic controllability of the controller’s behavior, providing a highly scalable foundation for modern, movement-intensive mechanics.

Keywords: Unreal Engine, game engine architecture, character kinematics, vector algebra, momentum preservation, locomotion, game development.