№ 8(49) 2025
НАУКА і ТЕХНІКА СЬОГОДНІ
серії: право, економіка, педагогіка, техніка, фізико-математичні науки

УДК 004.932:004.93’1:004.272

https://doi.org/10.52058/2786-6025-2025-8(49)-1578-1588

Місько Єгор Валерійович асистент кафедри комп’ютерних наук та інформаційних технологій, Житомирський Державний університет імені Івана Франка, м. Житомир, https://orcid.org/0009-0008-6719-2701

МЕТОДИ СИНХРОНІЗАЦІЇ ФІЗИЧНИХ ОБЧИСЛЕНЬ ІЗ ВИКОРИСТАННЯМ РУШІЯ CHAOS У БАГАТОКОРИСТУВАЦЬКИХ ІГРОВИХ СИСТЕМАХ

Анотація. Проблема синхронізації фізичних обчислень у багатокористувацьких ігрових системах з рушієм Chaos (Unreal Engine) полягає в усуненні розбіжностей станів об’єктів, спричинених мережевою затримкою, високою швидкістю руху й недетермінованістю багатопотокових симуляцій. Розглянуто ключові підходи до мережевої фізики - сервер-авторитет, клієнтське передбачення, інтерполяція/екстраполяція та rollback/Resimulation - з акцентом на їхніх компромісах між узгодженістю, чутливістю управління й ресурсними витратами.

Описано архітектурні особливості Chaos: асинхронний фізичний потік, інструменти руйнування, модулі Cloth і Vehicles, а також Network Physics Component, що об’єднує передбачення та перепрорахунок через збереження історії станів. Проаналізовано режими реплікації Predictive Interpolation та Resimulation: перший забезпечує миттєву локальну реакцію гравця з подальшою серверною корекцією, другий дає вищу точність через відкат і перепрорахунок, але значно підвищує витрати CPU та пам’яті.

Виділено практичні налаштування для підвищення надійності синхронізації: фіксований час кроку симуляції, синхронізація генератора випадкових чисел, розумне обмеження кількості одночасно динамічних тіл і кешування станів з урахуванням RTT. Наведено основні обмеження Chaos у мережевому контексті - недетермінованість між платформами і потоками, велике навантаження при детальній симуляції та можливі візуальні артефакти під час корекцій стану - і порівняно його підхід з альтернативами (детерміновані рушії або спрощені моделі фізики), що вказує на компроміс між реалістичністю й узгодженістю. Рекомендовано такі напрями вдосконалення: оптимізація мультипоточних алгоритмів Chaos, впровадження гібридних моделей авторитетності з адаптивним делегуванням симуляції, жанрово-орієнтована конфігурація параметрів реплікації та дослідження апаратного прискорення/детермінованих підсистем для критичних взаємодій.

Ключові слова: Chaos, Unreal Engine, мережева фізика, реплікація, прогнозна інтерполяція, ресимуляція, клієнтське передбачення, сервер-авторитет.

Misko Yehor Valeriiovych Assistant Department of Computer Science and Information Technology, Zhytomyr Ivan Franko State University, Zhytomyr, https://orcid.org/0009-0008-6719-2701

METHODS FOR SYNCHRONISING PHYSICS CALCULATIONS USING THE CHAOS ENGINE IN REPLICATED GAMING SYSTEMS

Abstract. The problem of synchronising physical calculations in multi-user gaming systems with the Chaos engine (Unreal Engine) lies in eliminating discrepancies in object states caused by network latency, high speed of movement, and the non-determinism of multi-threaded simulations. Key approaches to network physics are considered-server authority, client prediction, interpolation/extrapolation, and rollback/resimulation - with an emphasis on their trade-offs between consistency, control sensitivity, and resource costs.

The architectural features of Chaos are described: asynchronous physics stream, destruction tools, Cloth and Vehicles modules, as well as the Network Physics Component, which combines prediction and recalculation by storing state history. The Predictive Interpolation and Resimulation replication modes are analysed: the former provides an instant local response from the player with subsequent server correction, while the latter offers higher accuracy through rollback and recalculation, but significantly increases CPU and memory costs.

Practical settings for improving synchronization reliability are highlighted: fixed simulation step time, random number generator synchronization, reasonable limitation of the number of simultaneously dynamic bodies, and state caching with RTT taken into account.

The main limitations of Chaos in a network context are listed - non-determinism between platforms and threads, high load during detailed simulation, and possible visual artefacts during state corrections - and compares its approach with alternatives (deterministic engines or simplified physics models), indicating a trade-off between realism and consistency. The following areas for improvement are recommended: optimisation of Chaos multithreaded algorithms, implementation of hybrid authority models with adaptive simulation delegation, genre-oriented configuration of replication parameters, and research into hardware acceleration/deterministic subsystems for critical interactions.

Keywords: Chaos, Unreal Engine, network physics, replication, predictive interpolation, resimulation, client prediction, server authority.

Постановка проблеми. У сучасній індустрії відеоігор все більшого значення набувають реалістичні фізичні симуляції, особливо у багатокористувацьких іграх. Однак при цьому виникає критична задача - як узгодити результати фізичних обчислень на різних машинах (синхронізувати їх між клієнтами та сервером). В Unreal Engine ця задача вирішується через систему реплікації: «Replication - це процес синхронізації даних і викликів між клієнтами та сервером».[1] Нещодавно в Unreal Engine 5 було запроваджено новий фізичний рушій Chaos, побудований «з нуля» для потреб ігор наступного покоління. Chaos включає підтримку мережевої фізики серед своїх функціональних можливостей.[2] Проте основна проблема полягає в тому, що традиційна мережева фізика стикається з труднощами: при зіткненнях об'єктів (особливо високошвидкісних) та наявності мережевих затримок реплікація таких взаємодій значно ускладнюється. Іншими словами, швидкість руху об'єкта і затримка мережі можуть призводити до розбіжностей у фізичному стані між клієнтами.[3] Це обмежує якість та точність спільної симуляції. Таким чином, ключова проблема цього дослідження полягає в пошуку ефективних методів синхронізації фізичних обчислень з використанням рушія Chaos у багатокористувацьких ігрових системах і в оцінці можливостей Chaos для створення мережевих ігор з коректною реплікацією фізики.

Аналіз останніх досліджень і публікацій. Аспекти мережевої фізики відображені переважно у практичній документації та освітніх матеріалах, зокрема в документації Unreal Engine. Зокрема, офіційний огляд мережевої фізики пояснює, що «мережева фізика - це частина мережевої інфраструктури UE, що забезпечує роботу симуляцій фізики в мультиплеєрі».[2] У документах зазначено, що головним викликом є саме взаємодія об'єктів при зіткненнях: «реплікація таких взаємодій стає складнішою, чим швидше рухається об'єкт і чим більша затримка мережі». В Unreal Engine описано кілька режимів реплікації фізики (Default, Predictive Interpolation, Resimulation) для покращення точності клієнтських симуляцій.[3] Зокрема, режим Predictive Interpolation дозволяє клієнту передбачати локальні впливи, а Resimulation - частково перезапускати симуляцію на клієнті для коригування стану. Однак у цій документації ще мало прикладних оцінок того, наскільки сам рушій Chaos спрощує чи ускладнює реплікацію.

Нещодавній огляд на порталі daily.dev виділяє основні виклики мережевої фізики: затримку, помилки синхронізації, взаємодію зіткнень, масштабування. Автори зазначають, що для усунення цих проблем рекомендують застосовувати клієнтське передбачення та серверну авторитетність[4]. Інші джерела радять сувору синхронізацію руху через

фіксований часовий крок і серверні перевірки, а також використання інтерполяції для плавності відображення.[3] Незважаючи на ці загальні рекомендації, спеціальні дослідження методів синхронізації фізики саме з використанням рушія Chaos у мережевому середовищі поки що не поширені. Більшість наявних матеріалів висвітлює ідеї загальносистемного підходу (як-от клієнт-серверна архітектура, компенсація затримки, режим Predictive Interpolation), однак глибокий аналіз впровадження і оптимізації фізичного рушія Chaos у мультиплеєрному контексті залишається відкритим завданням.

Мета статті - дослідження особливостей і механізмів синхронізації фізичних обчислень з використанням рушія Chaos у багатокористувацьких ігрових системах, а також оцінка потенціалу та обмежень рушія Chaos в Unreal Engine при розробці мережевих реплікованих ігор.

Виклад основного матеріалу. Chaos - це новий фізичний рушій в Unreal Engine. Він інтегрований в UE5 (та доступний експериментально в UE4.26+)[2], як заміна старому PhysX від NVIDIA. Chaos розрахований на високо-продуктивну симуляцію - особливо для складних руйнувань і взаємодій у реальному часі.[5] Система містить такі основні компоненти: жорсткі тіла (Rigid Body), система руйнування (Destruction) з інструментами фраґментації геометрії, моделювання тканини (Cloth), ragdoll-анімацію та рухомі транспортні засоби (Vehicles). Усе це дозволяє реалізувати реалістичні об’єктні взаємодії- від коректної обробки зіткнень до складних динамік руйнування (наприклад, будівель, що розлітаються на осколки за фізичними законами).[6]

Архітектурно Chaos працює в окремому асинхронному потоці фізики, що відокремлює обчислення фізики від основного ігрового потоку. Це дає змогу рухові не блокувати рендеринг та логіку гри. На відміну від PhysX (який за замовчуванням працював у потоці гри), у Chaos є свій фізичний тред для симуляції. Такий підхід спрощує реплікацію: Epic зазначають, що «асинхронний потік дозволяє мережеву реплікацію» фізики, тоді як PhysX цього не підтримував без обхідних рішень. Крім того, власна розробка Epic дає їм більший контроль та можливість оптимізації, вирішуючи деякі відомі проблеми PhysX (наприклад, нестабільні результати зіткнень).[7]

Unreal Engine також має потужний мережевий фреймворк. За замовчуванням використовується сервер-авторитет: сервер симулює фізику, а клієнти отримують оновлені стани об’єктів (положення, швидкості тощо). UE підтримує механізми реплікації - якщо в актора ввімкнено реплікацію руху і його корінний компонент із фізикою активний, рушій автоматично надсилає дані клієнтам. Наприклад, «режим за замовчуванням» реплікації змінює швидкість об’єкта на клієнті так, щоб в середньому за півпроходу мережевого циклу його положення відповіло серверному.[3] При цьому клієнт робить наперед прогноз на пів RTT, щоб згладити затримку. Unreal також підтримує

більш просунуті режими мережевої фізики: режим *Predictive Interpolation* враховує локальну швидкість клієнта й дозволяє застосовувати передбачувані сили на об'єкти, а режим *Resimulation* дозволяє клієнту моделювати фізику наперед і в разі розбіжностей «відкотитися» до збереженого стану і перемоделювати з урахуванням серверного кроку. Ці механізми забезпечують більш плавну взаємодію об'єктів і кращу стійкість при зіткненнях, але вимагають додаткового кешування історії фізичного стану.[3] В цілому, система реплікації Unreal Engine дозволяє гнучко налаштувати модель авторитетності й синхронізації фізики у мультиплеєрі.

Методи синхронізації фізичних обчислень у мультиплеєрних ігрових системах

В мультиплеєрних іграх синхронізація фізики найчастіше базується на моделі сервер-авторитет. Сервер виконує повну симуляцію і відсилає клієнтам лише необхідні стани об'єктів (координати, швидкості). Клієнт, навпаки, лише відображає цей стан та використовує інтерполяцію або екстраполяцію для гладкого руху на своєму екрані. Такий підхід гарантує, що всі гравці бачать узгоджений світ, але введення гравця сприймається з затримкою не менше RTT (часу повного кола «запит-відповідь»).[8] Через це локальна реакція гравця затримується, що може знижувати відчуття керованості.

Щоб зменшити затримку на дії гравця, застосовується клієнтське передбачення (Client-Side Prediction). У цій схемі клієнт одразу застосовує введення гравця до локальної копії фізичної симуляції, а потім надсилає події серверу. Якщо сервер при отриманні даних індентичних вхідних дій дає інший результат (через мережеву затримку), клієнт отримує «глибокий стан» від сервера і застосовує виправлення з відповідним «відкатом» та повторною симуляцією (rollback) в історії кадрів.[9] Таким чином, клієнт бачить миттєву реакцію (відчуття відсутності лагу при натисканні), але потім може відбутися корекція положення об'єкта, яка зазвичай усувається плавною інтерполяцією.[8] Прикладом є ігри жанру FPS (First Person Shooter, Шутер від першої особи): постріл фіксується одразу на клієнті, а потім сервер перевіряє «відкотом» (lag compensation), чи був він точним.[9]

В окремих випадках застосовують метод повного відкату (Rollback): клієнт моделює фізику з власними введеннями і періодично отримує стан від сервера. Якщо виявлено невідповідність, клієнт «відмотує» до попереднього стану, коригує його та знову симулює вперед.[9] Цей метод вимагає детермінованості симуляції та великого обсягу обчислень (зберігання історії фізики на час хоча б RTT)[3]. В деяких іграх (наприклад, Rocket League) використовується саме такий підхід: фізика виконується на дуже високій частоті і при надходженні відмінностей усі тіла повторно симулюються.[10] Проте він дуже вимогливий до ЦП і пам'яті, що обмежує його застосування.

Крім того, у мультиплеєрі завжди виникає питання детермінізму фізичної симуляції. Оскільки реальні фізичні рушії (PhysX, Bullet тощо) використовують арифметику з плаваючою комою і багатопоточність, ідентичні початкові умови можуть давати різні результати на різних машинах. Як зауважено в обговореннях Unreal, PhysX *не є детерміністичним*, тобто однакові сили можуть породити трохи відмінний результат при повторі.[11] Це призводить до розбіжностей станів на сервері і клієнтах, тому зазвичай симулюють фізику лише на одному комп'ютері (найчастіше на сервері) і передають результати іншим. Увімкнення детермінованого режиму (фіксований час кроку, те сама послідовність випадкових чисел) знижує проблему, але повністю її не усуває при складних взаємодіях і високій затримці. Відомо, що з ростом затримки та швидкості рухів помилки реплікації зростають, погіршуючи точність синхронізації.[11]

Основні методи синхронізації:

Сервер-авторитет: сервер симулює і надсилає стан об'єктів, клієнти інтерполюють рух. Гарантує узгодженість, але локальна взаємодія затримується на час мережі.[3][8]

Клієнтське передбачення (предикція): клієнт відразу застосовує введення локально, сервер потім надсилає підтвердження і корекції (rollback) при необхідності.[9] Забезпечує негайну реакцію гравця, але потребує складного коду для корекції (імплементації механізмів відкату та інтерполяції).

Інтерполяція та екстраполяція: клієнт переміщує віддалені об'єкти плавно між отриманими станами (та можливим прогнозом). Це згладжує рух, але не дозволяє клієнту змінювати фізику до підказки від сервера.

Компенсація затримки (Lag Compensation): зазвичай використовується в шутерах, коли сервер «відкочує» стан на момент отримання вистрілу, щоб перевірити влучання з урахуванням latency[9]

Кожен підхід має плюси й мінуси. Наприклад, режим з детермінованим одночасним виконанням на всіх машинах (lockstep) дозволяє синхронізувати без передачі деталей фізики, але вимагає повного контролю над кожним об'єктом (приклад - системи на зразок Photon Quantum, де симуляція повністю детермінована).[12] В інших випадках обирають гібридні стратегії, довіряючи серверу фізичну симуляцію та даючи клієнту лише привілей координувати власний рух і лінійно передбачати оберти.

Використання рушія Chaos для реалізації синхронізації фізичних обчислень

У Unreal Engine з Chaos з'явилися спеціальні інструменти для мережевої фізики. Головний - Network Physics Component (компонент мережевої фізики).[13] Цей компонент дозволяє передавати на сервер керуючі сигнали (прискорення, оберт) і зберігати історію станів фізики на клієнті. Він

автоматично об'єднує передбачення та перепрорахунок: сервіс записує вхідні дані й результати симуляції для кожного кадру, а при отриманні нових серверних даних за потреби виконує «відкат» до історичного стану і симулює вперед.

При налаштуванні мережевої фізики з Chaos розробник додає Network Physics Component до свого актора (наприклад, фізичного Pawn). Потім у налаштуваннях цього актора вибирають режим реплікації «Predictive Interpolation» або «Resimulation»:

Predictive Interpolation (середовище із серверним авторитетом) - клієнт змінює швидкість об'єкта локально, а сервер відправляє дані з коригуваннями. Цей режим дозволяє застосовувати передбачувані сили (наприклад, поштовхи) на стороні клієнта до отримання інформації від сервера.[3] Сервер і клієнт потім узгоджують швидкість та положення.

Resimulation (для Physics Pawn) - клієнт знаходиться на півхвилинному часі попереді і зберігає історію станів як «кільце буфер».[3] Коли клієнт отримує оновлення від сервера, він порівнює серверні дані з відповідним станом у буфері. Якщо є велика різниця, запускається перепрорахунок: симуляція повертається назад до цього збереженого стану, виправляються помилки згідно з серверними даними, і далі симулюється до поточного моменту.[3] Наприкінці об'єкт може опинитися в іншому місці, тож для уникнення ривків включається плавна інтерполяція переходу. Режим Resimulation вимагає додаткових ресурсів (більше процесорного часу та пам'яті на буфер), але дозволяє клієнту повністю симулювати свою «фізичну Pawn» з введеннями користувача.

Важливі налаштування синхронізації з Chaos:

Фіксований час кроку: усі клієнти і сервер мають обчислювати фізику з однаковим фіксованим дельта-часом (наприклад, 60 або 120 кадрів на секунду). Це зменшує розбіжності при різних швидкостях оновлень.

Синхронізація генератора випадкових чисел: якщо у фізиці використовуються випадкові величини, їх потрібно або генерувати на сервері з передачею насіння, або запускати з одним спільним початковим насінням на всіх клієнтах. Інакше об'єкти можуть отримувати різні випадкові сили на різних машинах.

Вибір авторитетного вузла: зазвичай сервер залишається єдиним джерелом істини для фізики. Якщо ж потрібен «клієнт-авторитет» (наприклад, у випадках з низькою доступністю серверів), необхідно ретельно контролювати надійність таких передач та ризик читингу.

Приклад використання Network Physics Component описано у документації та статтях спільноти.[13] Розробник визначає структури даних для входів та станів симуляції, а рушій сам відправляє їх по мережі і здійснює необхідні перетікання (forward predict і rollback) «під капотом». Крім того, у

стандартному реплікаційному механізмі UE можна просто увімкнути *Replicate Movement* на компоненті з фізикою, і рушій автоматично оновлюватиме положення на клієнтах за схемою Predictive Interpolation.[3] У будь-якому випадку, при роботі з Chaos важливо усвідомлювати, де саме виконується симуляція (сервер чи клієнт) та як обробляти конфлікти станів.

Потенціал та обмеження рушія Chaos у мережевих іграх

Можливості Chaos: рушій забезпечує дуже високу точність і деталізацію фізики. Його система руйнування дозволяє створювати реалістичні деструктивні середовища: об'єкти можна заздалегідь дробити на фрагменти з налаштованою логікою розпаду.[6] Це значно підвищує відчуття взаємодії з ігровим світом. Крім того, Chaos підтримує складні налаштування транспортних засобів, деталізовані обмеження між тілами та інтегровані симуляції тканини, що дозволяють робити багатопараметричні моделі об'єктів у грі.[6] Оскільки рушій «рідний» для UE, він легко масштабується під великі сцени і отримує оптимізації від Epic (наприклад, техніки LOD для фізики та спеціальні шейдери для прискорення симуляції). Таким чином, для багатокористувацьких ігор Chaos відкриває потенціал для якісного «багатоклієнтського» руйнування і взаємодії - наприклад, розбиті уламки коректно реагують на вогонь гравців або взаємодіють з персонажами.

Обмеження: попри переваги, Chaos має і суттєві виклики. По-перше, за замовчуванням його симуляція недетермінована на різних машинах.[11] Навіть при однакових початкових умовах не гарантується ідентичність результатів між серверами і клієнтами, особливо якщо використовуються багатопоточні обчислення. Це може призводити до розбіжностей станів, тому мережевий режим Chaos спроектований так, що *лише один вузол* (зазвичай сервер) виконує фізику, а інші приймають коригування. Нестабільність може також виникати через округлення чисе - наприклад, Unity чи Unreal на різних платформах можуть трохи по-різному обчислювати силу тяжіння чи гравітацію.

По-друге, суть високоточної фізики - велике навантаження на процесор. Для мережевої синхронізації потрібна фіксована частота кроків і кешування історії станів (як у режимі Resimulation), що додатково збільшує витрати. Так, документація Unreal попереджає, що режим Resimulation «витрачає CPU і пам'ять» на зберігання історії фізики кожен кадрd.[3]

У великих світах із багатьма динамічними об'єктами це може стати вузьким місцем: як зауважено експертами, якщо гра дуже завантажена симуляцією фізики, повний відкат і перепрорахунок кожного кадру стає практично нездійсненним через витрати ЦП.[9]

Таким чином, для стабільної роботи мережевої фізики з Chaos часто потрібно обмежувати кількість одночасно рухомих тіл і ретельно налаштовувати цілі для CPU.

Третя проблема - розбіжність станів на клієнті та сервері. Навіть за використання Resimulation не гарантується безшовна конвергенція. Після перепрорахунку об'єкти можуть «стрибнути» в нову позицію, що помітно гравцеві. Unreal надає інструменти інтерполяції для згладжування таких змін, але повністю уникнути артефактів важко.[3] Крім того, висока затримка мережі або висока швидкість руху об'єктів може приводити до великих корекцій: чим більший RTT чи швидкість, тим сильніше страждає якість реплікації.

Нарешті, порівнюючи Chaos із іншими підходами до розподіленої фізики, слід відзначити: багато конкурентних рішень фактично відмовляються від складної фізики заради стабільності. Наприклад, системи на зразок Photon Quantum забезпечують 100% детерміновану симуляцію,[12] але їх фізика має бути строго визначена розробником (часто з простішими моделями). Інші ігри (першоособові шутери) ігнорують реальну фізику - вони описують рух і зіткнення власноруч і передають лише анімацію чи позицію. Інший спосіб - пакети на кшталт SnapNet доповнюють механізми Unreal (з глибоким кешуванням і оптимізованим відкатом), але суть залишається аналогічною до вбудованих методів. У деяких розробках застосовуються додаткові апаратні й програмні оптимізації: наприклад, Rocket League використовує рушій Bullet і постійно симулює фізику на клієнтах, періодично роблячи відкат - це дає чудову точність, але «надзвичайно дорогу» з точки зору обчислень реалізацію.[10] Тобто, інші підходи або спрощують фізику заради синхронності, або ж покладаються на величезні ресурси для підтримки детальних симуляцій (як це роблять в про-рівні esports із клієнтським передбаченням і компенсацією затримки).

Висновки. Використання рушія Chaos разом з можливостями мережевого фреймворку Unreal Engine дає гнучкі інструменти для синхронізації фізики у багатокористувацьких іграх. У розглянутих рішеннях найефективнішою виявилася модель з центральною симуляцією на сервері і локальним клієнтським передбаченням (Predictive Interpolation). Завдяки цьому клієнти можуть миттєво реагувати на свої введення, а сервер своєчасно надсилає корекції. Режим Resimulation з історією станів дозволяє досягти ще більшої точності (наприклад для фізичних Pawn), однак вимагає додаткових ресурсів. В цілому, рушій Chaos успішно виконує поставлену задачу: він підтримує високоточну фізику та надає можливість використовувати передбачення і відкат для компенсації затримок. Разом з тим виявлені обмеження - передусім нестабільність симуляції в різних потоках та значні витрати CPU під час синхронізації. Для покращення системи рекомендується подальше опрацювання таких аспектів: розвиток мультипоточних алгоритмів у Chaos (щоб зменшити навантаження на головний потік і скоротити час відкатів), дослідження додаткових методів узгодження станів (наприклад,

гібридних моделей з частково клієнтським авторитетом) та оптимізація параметрів мережевої реплікації (затримка, інтерполяція) під конкретні жанри ігор. Також перспективним є вивчення досвіду інших рушіїв і фреймворків (наприклад, інструментів з детермінованою фізикою), щоб винести кращі практики для майбутніх версій Chaos і Unreal Engine.

Література:

1. Epic Games. Мережеві можливості та мультиплеєр в Unreal Engine [Електронний ресурс]. dev.epicgames.com – Epic Games. Режим доступу: https://dev.epicgames.com/documentation/en-us/unreal-engine/networking-and-multiplayer-in-unreal-engine

2. Epic Games. Фізика в Unreal Engine [Електронний ресурс]. dev.epicgames.com – Epic Games. Режим доступу: https://dev.epicgames.com/documentation/en-us/unreal-engine/physics-in-unreal-engine

3. Epic Games. Огляд мережевої фізики [Електронний ресурс]. dev.epicgames.com – Epic Games. Режим доступу: https://dev.epicgames.com/documentation/en-us/unreal-engine/networked-physics-overview

4. daily.dev. Виклики мережевої фізики: питання й відповіді [Електронний ресурс]. daily.dev. Режим доступу: https://daily.dev/blog/networked-physics-challenges-qanda#:~:text=,engines%20aren%27t%20built%20for%20this

5. Epic Games. Огляд Chaos Physics [Електронний ресурс]. dev.epicgames.com – Epic Games. Режим доступу: https://dev.epicgames.com/documentation/en-us/unreal-engine/chaos-physics-overview?application_version=4.27

6. SDLCCorp. Дослідження системи Chaos Physics в Unreal Engine для руйнувань у грі [Електронний ресурс]. sdlccorp.com – SDLCCorp. Режим доступу: https://sdlccorp.com/post/exploring-unreal-engines-chaos-physics-system-for-game-destruction/#:~:text=The%20Chaos%20Physics%20system%20is,tools%2C%20especially%20for%20destructible%20environments

7. Reddit. Обговорення продуктивності фізики в UE5.2 - порівняння Chaos [Електронний ресурс]. Режим доступу: https://www.reddit.com/r/unrealengine/comments/15nawjv/horrific_physics_performance_ue52_chaos_compared/#:~:text=Image%3A%20Profile%20Badge%20for%20the,Commenter

8. Лем А., Початок розробки ігор: компенсація затримки та передбачення [Електронний ресурс]. Режим доступу: https://medium.com/@lemapp09/beginning-game-development-lag-compensation-and-prediction-f8bc028c20b6

9. MAS-Bandwidth. Вибір правильної мережевої моделі для вашої мультиплеєрної гри [Електронний ресурс]. Режим доступу: https://mas-bandwidth.com/choosing-the-right-network-model-for-your-multiplayer-game/#:~:text=To%20keep%20the%20server%20authoritative%2C,development%2C%20optimistic%20execution%20with%20rollback

10. GameDev.net. Відкати та продуктивність відтворення симуляції [Електронний ресурс].Режим доступу:https://www.gamedev.net/forums/topic/713082-rollbacks-and-simulation-replay-performance/#:~:text=It%27s%20also%20interesting%20to%20me,only%20assume%20the%20rest%20of

11. Unreal Engine Forums. Як правильно реплікувати позицію м’яча з фізикою [Електронний ресурс]. Режим доступу: https://forums.unrealengine.com/t/how-to-properly-replicate-ball-location-with-physics/407862

12. Photon Engine. Photon Quantum - детермінована мережна симуляція [Електронний ресурс]. Режим доступу: https://www.photonengine.com/quantum#:~:text=Photon%20Quantum%20is%20the%20only,in%20sync

13. Vorixo / DevTricks. Використання передбачення фізики [Електронний ресурс]. Режим доступу: https://vorixo.github.io/devtricks/phys-prediction-use/#:~:text=The%20way%20I%E2%80%99ve%20found%20other,make%20use%20of%20these%20features

References:

1. Networking and multiplayer in Unreal Engine. dev.epicgames.com. Retrieved from https://dev.epicgames.com/documentation/en-us/unreal-engine/networking-and-multiplayer-in-unreal-engine [in English].

2. Physics in Unreal Engine. dev.epicgames.com. Retrieved from https://dev.epicgames.com/documentation/en-us/unreal-engine/physics-in-unreal-engine [in English].

3. Networked physics overview. dev.epicgames.com. Retrieved from https://dev.epicgames.com/documentation/en-us/unreal-engine/networked-physics-overview [in English].

4. Networked physics challenges: Q&A. daily.dev. Retrieved from https://daily.dev/blog/networked-physics-challenges-qanda#:~:text=,engines%20aren%27t%20built%20for%20this [in English].

5. Chaos physics overview. dev.epicgames.com. Retrieved from https://dev.epicgames.com/documentation/en-us/unreal-engine/chaos-physics-overview?application_version=4.27[in Eng]

6. Exploring Unreal Engine's Chaos physics system for game destruction. sdlccorp.com. Retrieved from https://sdlccorp.com/post/exploring-unreal-engines-chaos-physics-system-for-game-destruction/#:~:text=The%20Chaos%20Physics%20system%20is,tools%2C%20especially%20for%20destructible%20environments [in English].

7. Horrific physics performance UE5.2 - Chaos compared. reddit.com. Retrieved from https://www.reddit.com/r/unrealengine/comments/15nawjv/horrific_physics_performance_ue52_chaos_compared/#:~:text=Image%3A%20Profile%20Badge%20for%20the,Commenter [in English].

8. Lem Apperson, Beginning game development - lag compensation and prediction. medium.com. Retrieved from https://medium.com/@lemapp09/beginning-game-development-lag-compensation-and-prediction-f8bc028c20b6 [in English].

9. Choosing the right network model for your multiplayer game. mas-bandwidth.com. Retrieved from https://mas-bandwidth.com/choosing-the-right-network-model-for-your-multiplayer-game/#:~:text=To%20keep%20the%20server%20authoritative%2C,development%2C%20optimistic%20execution%20with%20rollback [in English].

10. Rollbacks and simulation replay performance. gamedev.net. Retrieved from https://www.gamedev.net/forums/topic/713082-rollbacks-and-simulation-replay-performance/#:~:text=It%27s%20also%20interesting%20to%20me,only%20assume%20the%20rest%20of [in English].

11. How to properly replicate ball location with physics. forums.unrealengine.com. Retrieved from https://forums.unrealengine.com/t/how-to-properly-replicate-ball-location-with-physics/407862 [in English].

12. Photon Quantum. photonengine.com. Retrieved from https://www.photonengine.com/quantum#:~:text=Photon%20Quantum%20is%20the%20only,in%20sync [in English].

13. Phys prediction: use. vorixo.github.io. Retrieved from https://vorixo.github.io/devtricks/phys-prediction-use/#:~:text=The%20way%20I%E2%80%99ve%20found%20other,make%20use%20of%20these%20features [in English].