DOI: https://doi.org/10.15407/emodel.45.02.095
004.92, 004.05

Д.І. Матвєєв, аспірант, **О.Ф. Лановий**, канд. техн. наук
Харківський національний університет радіоелектроніки
Україна, 61166, Харків, пр-т Науки 14, к. 267
тел. +38 057 702 14 46, e-mail: d_se@nure.ua

Методи спрощення опрацювання систем симуляції незалежних часток на основі середовища Unreal Engine 4

Розглянуто особливості розроблення та практичної реалізації систем, призначених для симуляції поведінки незалежних часток, принципів їх функціонування, особливості реалізації моделі та процесу спрощення опрацювання. Особливу увагу приділено процесу спрощення опрацювання на різних рівнях опрацювання системи симуляції часток. Розглянуто чинники, які мають на неї безпосередній вплив, а також розглянуто спрощення шейдерів, як вагому складову швидкодії системи.

К л ю ч о в і с л о в а: комп'ютерна графіка, системи часток, спрощення, Unreal Engine 4, шейдери.

Активний розвиток технологій опрацювання та інтерпретації візуальної інформації з використанням складних тривимірних моделей набуває широкого застосування в розроблені мультимедійного програмного забезпечення (МПЗ). Поширення інструментальних засобів та технологій розроблення МПЗ знайшло відображення в сфері комп'ютерної ігрової індустрії, інтерактивному кіно, застосунках доповненої та віртуальної реальності.

З розвитком інструментальних засобів розроблення виникає необхідність в динамічному спрощенні опрацювання моделей. Причиною цього слід вважати той факт, що спрощення процесу розроблення достатньо складних моделей зумовило зниження інтересу розробників до багатофакторного спрощення вже існуючих моделей. Такий підхід вимагає окремого аналізу проблеми та розроблення алгоритмів модифікації моделей для спрощення їх опрацювання.

Визначимо основні поняття, які будемо використовувати. Для створення МПЗ використаємо специфічну категорію інструментальних засобів, яка має назву «Game Engine». Позначимо їх терміном «рушій». Наз-

вемо МПЗ, що розглядаються, іграми, оскільки в них використовуються всі технології сучасних рушіїв, спрощення яких розглядається.

Ціль спрощення — покращення швидкодії певного модуля та системи в цілому. Важливо зазначити, що в англомовній літературі в контексті графічних бібліотек та ігрових рушіїв для описання імітації поведінки часток використовується термін «simulation» [1]. Тому далі будемо використовувати термін «симуляція», бо розглядається не сам процес симуляції, а спрощення його опрацювання.

Процес спрощення може бути застосований майже до усіх модулів сучасних рушіїв. Це пов'язано з тим, що рушії забезпечують дотримання вимог у тих сферах застосування, на які ці рушії спрямовані. Для таких сфер застосування, як кіноіндустрія або симуляції (симуляція деяких фізичних процесів, де рушій використовується для візуалізації процесу), де моделі виконуються не в реальному часі, покращення швидкодії не є критичним, бо впливає лише на комфорт роботи з проєктом, але не на фінальний результат, який може проходити через процедуру рендеринга у будь-який час.

Під рендерингом (rendering — візуалізація) розуміємо процес візуалізації тривимірних моделей. Основною сферою застосування, де проведення процедури покращення швидкодії є критичним, залишаються програмні додатки, що виконуються в реальному часі. Для цього класу додатків головною вимогою щодо процесу покращення швидкодії залишається максимізація кількості кадрів у секунду, а також синхронізація функціонування всіх використаних елементів рушія через те, що вони можуть запускатись на різному обладнанні. Це є важливим, бо мобільність активно розвивається і через це спрощення вже існуючих додатків під нові типи мобільних пристроїв стає більш актуальною.

Постановка задачі. Наукова новизна та практичне значення складаються з систематизації вже існуючих методів спрощення опрацювання систем симуляції часток та експериментального дослідження їх ефективності. В першу чергу це стосується саме модуля системи симуляції часток, а також спрощення шейдерів.

Шейдери — це набір інструкцій для графічного процесору, за допомогою якого виконується рендеринг зображення. На рис. 1 наведено схему растеризації зображення — перетворення тривимірної сцени у двовимірне зображення. Кожна частка — це тривимірна модель, яка проходить через весь процес растеризації. Найбільшу складність розрахунку мають шейдерні інструкції та опрацювання прозорості на етапах механізму відображення текстур, тесту прозорості та тесту буферу глибини.

Задача полягає в систематизації методів спрощення опрацювання систем симуляції часток та спрощенні шейдерів цих часток. Систематизація

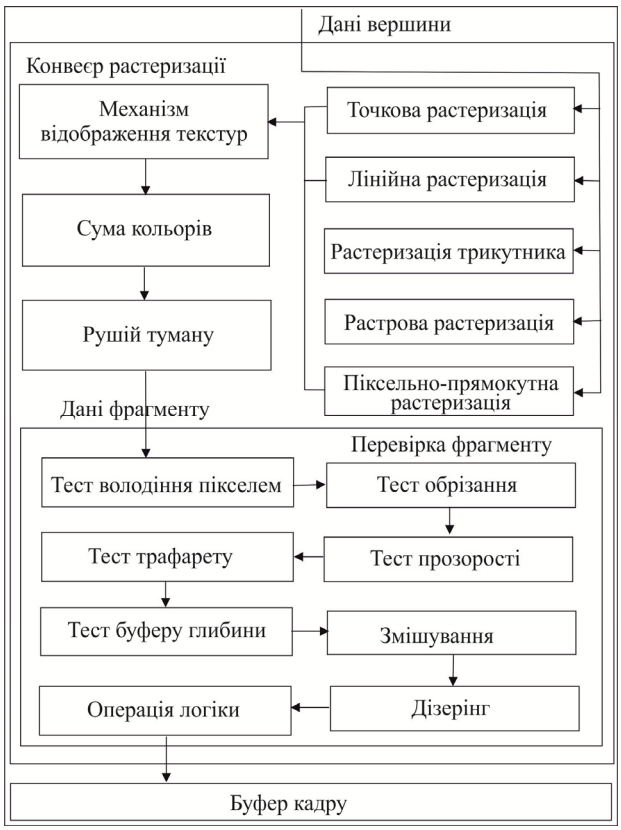


Рис. 1. Схема процесу растерізації зображення в OpenGL

виконується за критеріями ефективності цих методів, що впливає на швидкодію системи симуляції часток

Порівняння систем симуляції часток та типів емітерів. Будемо розглядати спрощення одного з модулів — систем симуляції часток. У рушія Unreal Engine 4 та 5 існує дві системи симуляції часток: Cascade [1] та Niagara [2]. На рис. 2 наведено їх редактори. Фундаментальної різниці між цими системами немає, тому розглянемо особливості спрощення системи Cascade, оскільки вони актуальні і для системи Niagara, основною особливістю якої є можливість наслідувати емітери (елемент системи часток, який описує поведінку групи часток на мета-рівні) та створювати код всередині самої системи.

Система симуляції часток складається з емітерів. Емітер описує весь цикл існування та відображення частки. Він не є детермінованим, оскільки визначає строк циклу існування частки та її зміни у нормалізований період існування. Емітер також задає випадкові значення, визначені верхньою та

нижньою межами, або функціями. Це створює індивідуальну поведінку [3] для кожної частки.

Емітери бувають двох типів: CPU (які обчислюються на центральному процесорі) та GPU (які обчислюються на графічному процесорі), що дозволяє визначити, де буде відбуватись розрахунок поведінки часток. На рис. 3 видно, що при однакових параметрах CPU-часток менше, ніж GPU-часток. Це відбувається через те, що CPU-частки (перші) розраховуються на процесорі, а GPU-частки (другі) — на відеокарті [4].

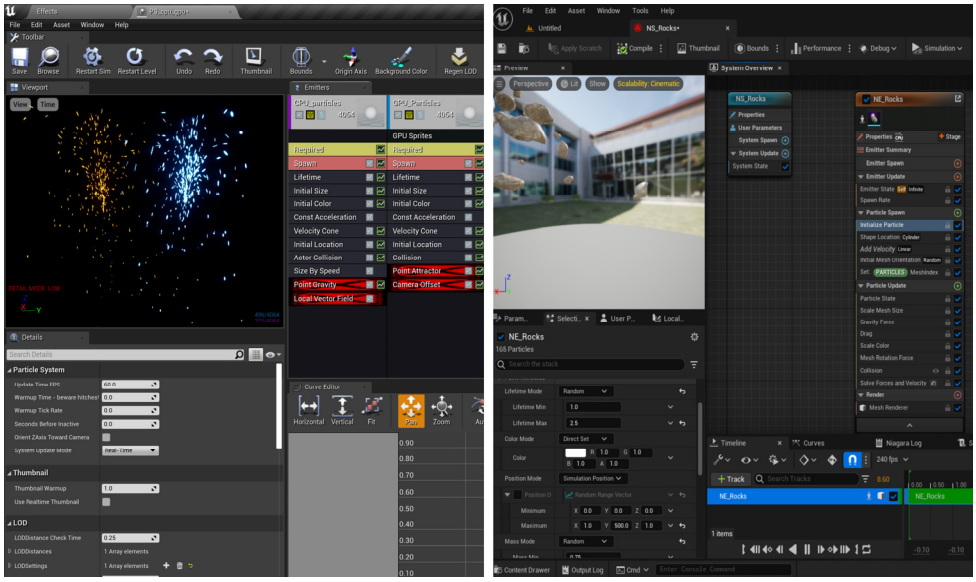
Перші частки мають штучні обмеження кількості, пов'язані з тим, що деякі операції при значній кількості часток складно розраховувати за допомогою процесора. Другі частки не мають обмеження кількості, але мають відмінності, пов'язані з їх розміром. Перші частки краще використовувати для малої кількості часток великого розміру, а другі — для великої кількості часток малого розміру. Така рекомендація зумовлена тим, що частки другого типу важчі, але це компенсується більш швидким опрацюванням великої кількості часток.

Між поведінкою часток першого та другого типу також існують відмінності. Деякі функції для них працюють по різному, або не працюють взагалі. На рис. 2 можна побачити червоні поля у емітерів різних типів — це приклади функцій, що є унікальними для GPU або CPU. На рис. 3 видно, що в нижній частині моделі частки більше розподілені через те, що для них включена фізична реакція на оточення. Вона працює для обох типів часток, але у різний спосіб.

Фізична взаємодія [5] для часток першого типу використовує колізійну геометрію (примітиви, які використовуються для симуляції фізичних взаємодій) усієї сцени. Це потребує складного обчислювання. Ця фізична взаємодія більш точна і дає можливість використання подій [6]. При колізії частки з оточенням чи її смерті можна викликати подію. Вона може використовуватись для створення інших часток з врахуванням координат, параметрів та вектора імпульсу.

Частки другого типу опрацьовуються швидше, але існують як спільність. Через це для них не працюють події, створення джерел світла [7], зсування камери тощо. Фізичні взаємодії для часток другого типу працюють в інший спосіб — візуально. Система симуляції часток отримує з кадру растеризації сцени буфер глибини (тест буфер глибини), нормалізує вектори отриманої «поверхні» та симулює фізичні взаємодії відповідно до них. Якщо підлога розташована нижче області кадру, то частки будуть проходити через неї. Це можна побачити на рис. 4, де стрілочками показані відскоки часток від підлоги, яка не потрапила в кадр.

Інколи необхідно одночасно використовувати фізичні взаємодії для великої кількості часток та створювати події, наприклад генерувати калюжі в місцях падіння крапель. В такому випадку кращим буде варіант



а *б*

Рис. 2. Інтерфейси Cascade (*а*) та Niagara (*б*)

Рис. 3. CPU-частки (ліворуч) та GPU-частки (праворуч)

розподілу системи часток на дві окремі підсистеми: перша буде використовувати CPU для генерації незначної кількість часток з точною коллізією та подіями, а друга, GPU, — генерувати інші частки з використанням швидкої симуляції фізичних взаємодій. В результаті такого розподілу візуальної різниці практично не буде, але це дозволить спростити процес симуляції часток. Цей підхід можна використовувати у багатьох випадках, оскільки при великій кількісті часток складно помітити незначні флуктуації.

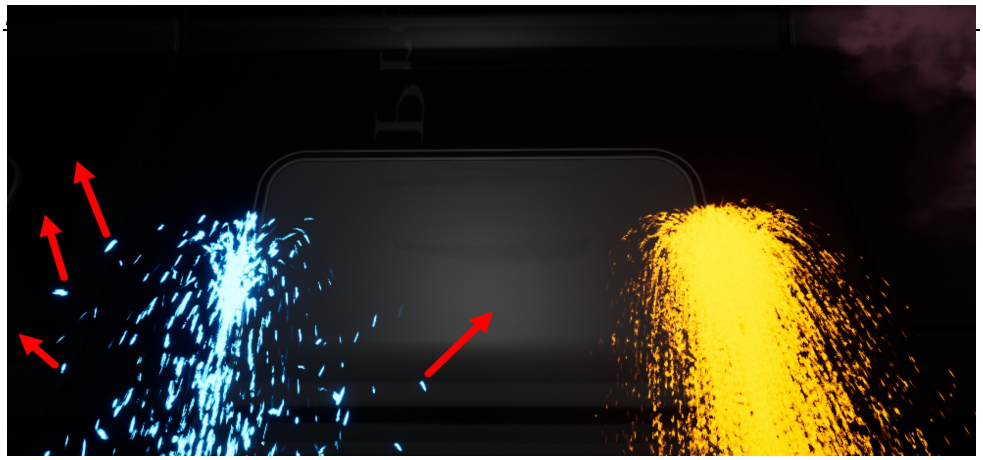


Рис. 4. Фізична взаємодія CPU-часток (ліворуч) та GPU-часток (праворуч)

Рис. 5. Варіанти реалізації диму

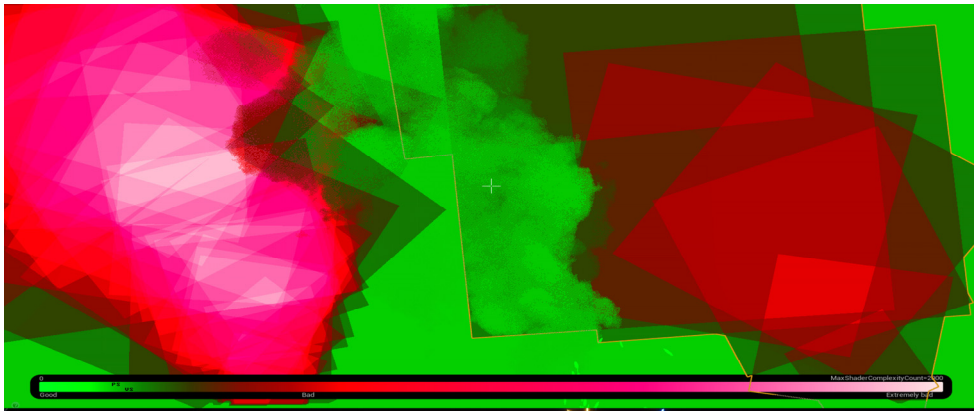


Рис. 6. Складність шейдерів варіантів реалізації диму

Найпростішим варіантом спрощення симуляції часток є зменшення їх кількості [8]. Мозок людини сприймає не кожну частку окремо, а розглядає всю систему в цілому. Отже, враховуючи принципи візуальної композиції (якщо розфокусувати зір, можна зрозуміти як наш мозок сприймає систему часток), систему часток можна уявити як пляму, що має власні характеристики: розмір, колір та прозорість. Зберегти цей ефект можливо, якщо при зменшені кількості часток збільшити їх розмір, але тоді частки будуть виглядати інакше. Тому можна зберегти співвідношення розміру, збільшивши тільки середній розмір часток. Такий саме підхід можна використовувати і для параметрів прозорості та тривалості існування частки, щоб частка, наприклад, існувала не 4 с., а від 1 до 4 с. Це збереже візуальне відображення симуляції, але зробить її оброблення значно простішим.

Спрощення шейдерів для систем симуляції часток. Як було зазначено, спрощення рендерингу часток більш вагоме, ніж їх симуляції. Пам'ятаємо, що шейдер складається з математичних інструкцій [9], які виконуються для кожного фрагменту (пікселю на екрані) [10]. Тобто складність опрацювання шейдеру залежить від кількості інструкцій, їх складності та кількості текселей на екрані, які займає шейдер.

Найбільш складними є напівпрозорі шейдери [11]. Їх складність пов'язана з тим, що необхідно не лише рисувати тексель, але й правильно сортувати та змішувати його з іншими. Для цього створюється буфер прозорих шейдерів (тест прозорості), де відбуваються всі вказані операції. Складність їх прорахунку залежать від глибини каскаду.

На рис. 5 зображено три варіанти диму, зроблені через систему симуляції часток. Перший варіант (зліва направо) — базова система диму. Так його зазвичай створюють, використовуючи високу щільність напівпрозорих спрайтів (тривимірних поверхонь) с Translucent шейдером та атласом анімованого диму [12].

Ці анімації накладаються одна на одну під різними кутами, різних розмірів та пересуваються з різною швидкістю. Спрайти направлені на камеру. Для них використовується значний параметр прозорості (0,9—0,99). Через прозорість і велику глибину каскаду отримується реалістичний ефект диму, пару, тощо. Велика глибина каскаду негативно впливає на швидкість рендеру кадру. Її треба за можливістю зменшувати.

Такий значно зменшений варіант є третім на рис. 5. Він також схожий на дим, але більш рідкий. Щільність диму в цьому варіанті отримується при меншому значенні прозорості (0,2—0,5) та більшому середньому розмірі спрайтів. Це допомагає значно зменшити кількість каскадів при несуттєвому зменшенні якості.

На рис. 6 за шкалою від зеленого до світло-рожевого зображено складність шейдерів диму, показаного на рис. 5. На рис. 6 видно, наскільки ефективним є зменшення глибини каскаду. Можна оцінити, наскільки кращим став новий дим щодо балансу між спрощенням та якістю. Особливу увагу треба приділити диму посередні. Він має яскраво-зелений колір, що свідчить про найменшу кількість інструкцій текселей цього диму. Для нього був використаний так званий «дітер-шейдер», в основі якого лежить маскед-шейдер — шейдер, який малює текселі у двох станах: прозорий та непрозорий. Ця його варіація використовує роботу з гаусовським шумом та особливим буфером TAA (Temporal Anti-aliasing) згладжування.

Отже, він створює щільну сітку варіацій між прозорими-непрозорими пікселями, що зумовлює ефект напівпрозорих пікселів. Цей метод має недоліки. По-перше, він коректно працює тільки з TAA згладжуванням. По-друге, він має ефект «гостінгу» (якщо придивитись до другого диму на рис. 5, то можна побачити шлейф). Це більш помітно в динаміці та впливає на видимість об'єктів за димом. По-третє, він потребує максимальної якості TAA, що використовується для оброблення зображення значної кількості інструкцій. Дітер-шейдер краще за все використовувати для невеликих спрайтів, щоб зменшити помітність гостінгу, бо ефект підвищення швидкодії, який він дає, вагоміший за незначні візуальні артефакти.

Інші вагомі параметри спрощення. У системах симуляції часток можна використовувати джерела світла [7]. Зазвичай динамічне освітлення, в якому не використовуються тіні, несуттєво впливає на швидкодію системи. При цьому воно також може бути вагомим при великій кількості джерел. У даному випадку можна використовувати принцип, описаний для генерації подій. Найважливішим є врахування трьох параметрів цього джерела світла: вплив на прозорість, високоякісне освітлення, відкидання тіней.

Перший параметр надає джерелу можливість впливати на півпрозорі об'єкти (також і на самі частки), що робить його розрахунок складним. Висока якість освітлення збільшує кількість інструкцій для його опрацювання. Найскладнішим за кількістю математичних інструкцій є опрацювання тіней, тому необхідно намагатись зменшити їх кількість.

При створенні системи часток в емітерах вже встановлюється тривалість існування частки [13]. Якщо відключити емітер, частки генеруються нескінченно та починають жити вічно. Вище зазначено, що у CPU-часток є обмеження, пов'язане з їх кількістю. Це впливає на те, яка кількість часток буде відображатись одночасно. Всі інші частки будуть тільки розраховані, що може призвести до переповнення пам'яті. Для запобігання цього необхідно встановлювати максимально допустиму межу тривалості існування часток.

Кількість часток	CPU та GPU, мс	Спрощення фізичних взаємодій CPU та GPU, мс	Спрощення освітлення CPU та GPU, мс	Дим 1, мс	Дим 2, мс	Дим 3, мс
1000	2,73 та 1,27	3,28 та 2,91	9,37 та 4,99	15,39	2,28	6,44
10000	5,12 та 2,32	12,39 та 8,76	19,94 та 12,35	36,94	5,32	12,84
100000	15,37 та 4,79	32,94 та 18,33	42,12 та 24,31	87,27	8,74	23,56

На швидкодію системи суттєво впливає функція попереднього розрахунку часток. Інколи, при побудові моделі, необхідно, щоб частки з'явились не поступово, а моментально і вже з урахуванням симуляції. Цей розрахунок доволі повільний і може вплинути на зменшення кількості кадрів в секунду.

Один з необхідних елементів GPU-часток — так звані «баундінг-бокси» [14], тобто області поширення часток. CPU-частки мають динамічні баундінг-бокси [8]. Це паралелепіпеди, які описують всі частки, що генерує система симуляції часток. Якщо камера не бачить баундінг-бокс, то вона тимчасово відключає систему симуляції часток. Для GPU-часток динамічна генерація баундінг-боксів неможлива, тому необхідно вказувати їх вручну. Слід враховувати, що розрахунок баундінг-боксів — складна операція, складність якої залежить від кількості часток. Тому у тих випадках, коли це можливо, кращим є ручне визначення цих баундінг-боксів для усіх типів часток.

Останнім і не менш важливим елементом спрощення систем часток є лодування [15]. Це механізм, який дозволяє змінювати параметри емітерів в залежності від дистанції. Тобто для системи часток вогняної кулі з димом на визначеній відстані можна спочатку вимкнути іскри, зменшити щільність диму та вогню, а на максимальній відстані — вимкнути дим і навіть вогонь. Треба враховувати те, що хоча б один емітер має бути активним або з активованою функцією обчислювання навіть у вимкненому стані, бо це може призвести до того, що лоди не будуть перемикатись у зворотному напрямку.

Результати порівняльних розрахунків наведено в таблиці. Отримані експериментальні дані свідчать про ефективність використаних методів спрощення. Більшість замірів не потрапили до таблиці через складність подачі зібраної інформації в стислому вигляді.

Експеримент проводився на однаковій сцені з однаковою системою симуляції часток з зафіксованою камерою. Результати замірів подано в мілісекундах. Вони відображають різницю між розрахунком кадру з базовими налаштуваннями системи симуляції часток та отриманим спрощенням. Для диму показано різницю часу між сценою з системою симуляції часток та без неї.

Висновки

На спрощення опрацювання систем часток впливає велика кількість параметрів. Одним з найважливійших параметрів є контроль шейдерів, оскільки сама система часток доволі розумна і проводить симуляцію на низькому рівні, що потребує меншого контролю з боку розробників. Найвпливовійшими є наступні параметри та дії:

1. Правильне використання типу часток (CPU та GPU). Кожен тип має використовувати принципи відповідності цілей та мінімальної необхідності функціоналу.

2. Розумне використання фізичних взаємодій. Якщо можна замінити її менш точною — це необхідно робити, оскільки в динаміці незначні флуктуації складніше помітити. Якщо треба використовувати велику кількість емітерів та події, краще поділити емітери на два типи: емітери GPU-типу використовувати для симуляції великої кількості часток, а емітери CPU-типу — для симуляції меншої кількості часток для подій або джерел світла. Погіршення якості буде незначним, але це допоможе в процесі спрощення.

3. Зменшення кількості. Найпростіший варіант — зменшити кількість часток та збільшити їх середній розмір. Додавши варіацію розміру та тривалості життя, можна візуально компенсувати зменшену кількість часток.

4. Вимірюване використання джерел світла. Нечасто існують ситуації, коли необхідні джерела світла мають максимальну якість. Якщо такі ситуації з'являються, то можна використовувати п. 2.

5. Врахування тривалості існування часток. Вони повинні не лише створюватись, але і вчасно знищуватись.

6. Обмеження максимальної кількості часток у тих випадках, коли це можливо.

7. Обережне використання попереднього розрахунку часток. Великий проміжок часу, помножений на кількість часток та складність операцій, можуть викликати зменшення швидкодії системи.

8. Контроль області розповсюдження часток. Немає необхідності їх опрацьовувати, коли вони не потрапляють до буферу кадра.

9. Налаштування лодування. Це допоможе зменшувати кількість часток в залежності від дистанції та навіть вимикати симуляцію, коли їх майже не видно.

Актуальність проведених досліджень підтверджується збільшенням попиту на портування проектів мультимедійного характеру на більш слабкі пристрої. Автоматизованих процесів спрощення мультимедійних систем недостатньо, тому потрібен більш фундаментальний підхід. Ефективність побудованої моделі підтверджує ефективність саме цього підходу до спрощення систем симуляції незалежних часток.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Cascade Particle Systems // Unreal Engine Documentation. Режим доступу: https://docs. unrealengine.com/4.26/en-US/RenderingAndGraphics/ParticleSystems/ (дата доступу 08 лютого 2023 р.)
2. Niagara Visual Effects // Unreal Engine Documentation. Режим доступу: https://docs. unrealengine.com/4.26/en-US/RenderingAndGraphics/Niagara/ (дата доступу 08 лютого 2023 р.)
3. *Лановой А.Ф., Лановой А.А.* Моделирование поведения толпы на основе дискретно-событийного мультиагентного подхода // Східно-Європейський журнал передових технологій, 2014, №4(70), с. 52—57.
4. GPUSprites Type Data // Unreal Engine Documentation. Режим доступу: https://docs. unrealengine.com/4.26/en-US/RenderingAndGraphics/ParticleSystems/Reference/TypeData/ GPUSprites/ (дата доступу 08 лютого 2023 р.)
5. Collision Modules // Unreal Engine Documentation. Режим доступу: https://docs. unrealengine. com/4.26/en-US/RenderingAndGraphics/ParticleSystems/Reference/Modules/Collision/ (дата доступу 09 лютого 2023 р.)
6. Event Modules // Unreal Engine Documentation. Режим доступу: https://docs.unrealengine. com/4.26/en-US/RenderingAndGraphics/ParticleSystems/Reference/ Modules/Event/ (дата доступу 09 лютого 2023 р.)
7. Particle Lights // Unreal Engine Documentation. Режим доступу: https://docs.unrealengine. com/4.26/en-US/RenderingAndGraphics/ParticleSystems /ParticleLights/ (дата доступу 09 лютого 2023 р.)
8. VFX Optimization Guide // Unreal Engine Documentation. Режим доступу: https:// docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/ParticleSystems /Optimization/ (дата доступу 09 лютого 2023 р.)
9. *Матвєєв, Д.І., Лановий, О.Ф.* Проблеми оптимізації графіки під пристрої віртуальної реальності//Λόγοσ.Online, 2020, №14. http://eoi.citefactor.org/10.11232/2663-4139.14.04
10. Особливості підготовки 3D моделей для використання у VR проєктах // Science, Research, Development. Режим доступу: http://www.xn--e1aajfpcds8ay4h.com.ua/files/ 118_01_xi_2021.pdf#page=35 (дата доступу 09 лютого 2023 р.)
11. Порівняння методів текстурування моделей для мобільних платформ // Science, Research, Development. Режим доступу: http://www.xn--e1aajfpcds8ay4h.com.ua/files/ 118_01_xi_2021.pdf#page=37 (дата доступу 09 лютого 2023 р.)
12. SubUV Modules // Unreal Engine Documentation. Режим доступу: https://docs.unrealengine. com/4.26/en-US/RenderingAndGraphics/ParticleSystems/Reference/Modules/SubUV/ (дата доступу 09 лютого 2023 р.)
13. Lifetime Modules // Unreal Engine Documentation. Режим доступу: https://docs.unrealengine. com/4.26/en-US/RenderingAndGraphics/ParticleSystems/Reference/Modules/Lifetime/ (дата доступу 09 лютого 2023 р.)
14. Дослідження інструментів та засобів оптимізації 3D-графіки в комп'ютерних іграх та їх застосування до ігор у жанрі "First-person Shooter" // Електронний архів Харківського національного університету радіоелектроніки. Режим доступу: https:// open-archive.nure.ua/server/api/core/bitstreams/e8582e45-10b9-44bf-aabc-cb0b120389ee/content (дата доступу 11 лютого 2023 р.)
15. Particle System Level of Detail (LOD) // Unreal Engine Documentation. Режим доступу: https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/ParticleSystems/LODs/ (дата доступу 09 лютого 2023 р.)

Отримано 17.02.2023;
після доопрацювання 03.04.2023

REFERENCES

1. Cascade Particle Systems. *Unreal Engine Documentation*. URL: https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/ParticleSystems/ (date of access: 08.02.2023)
2. Niagara Visual Effects. *Unreal Engine Documentation*. URL: https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/Niagara (date of access: 08.02.2023)
3. Lanovyy O.F., Lanovyy, O.O. (2014), Crowd Behavior Modeling Based on Discrete-Event Multi-Agent Approach. *Skhidno-Ievropeiskyi zhurnal peredovykh tekhnolohii*. Vol. 4, no. 7, pp. 52-57.
4. GPUSprites Type Data. *Unreal Engine Documentation*. URL: https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/ParticleSystems/Reference/TypeData/GPUSprites (date of access: 09.02.2023)
5. Collision Modules. *Unreal Engine Documentation*. URL: https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/ParticleSystems/Reference/Modules/Collision/ (date of access: 09.02.2023)
6. Event Modules. *Unreal Engine Documentation*. URL: https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/ParticleSystems/Reference/Modules/Event/ (date of access: 09.02.2023)
7. Particle Lights. *Unreal Engine Documentation*. URL: https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/ParticleSystems/ParticleLights/ (date of access: 09.02.2023)
8. VFX Optimization Guide. *Unreal Engine Documentation*. URL: https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/ParticleSystems/Optimization/ (date of access: 09.02.2023)
9. Matvieiev D.I., Lanovyy O.F. (2020), Problems of graphics optimization for virtual reality devices. *Λόγοσ.Online*. Vol. 14. URL: http://eoi.citefactor.org/10.11232/2663-4139.14.04 (date of access: 09.02.2023)
10. Matvieiev D.I. (2021), Features of preparing 3D models for use in VR projects. *Science, Research, Development.* Vol. 46, pp. 35–36
11. Matvieiev D.I. (2021), Comparison of model texturing methods for mobile platforms. *Science, Research, Development*. Vol. 46, pp. 37–38
12. SubUV Modules. *Unreal Engine Documentation*. URL: https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/ParticleSystems/Reference/Modules/SubUV/ (date of access: 11.02.2023)
13. Lifetime Modules. *Unreal Engine Documentation*. URL: https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/ParticleSystems/Reference/Modules/Lifetime/ (date of access: 11.02.2023)
14. Hrynevych K.V. Research of tools and means of optimization of 3D graphics in computer games and their application to games in the " First-person Shooter" genre. *Electronic archive of Kharkiv National University of Radio Electronics*. URL: http://openarchive.nure.ua/server/api/core/bitstreams/e8582e45-10b9-44bf-aabccb0b120389ee/content (date of access: 11.02.2023)
15. Particle System Level of Detail (LOD). *Unreal Engine Documentation*. URL: https://docs.unrealengine.com/4.26/en-US/RenderingAndGraphics/ParticleSystems/LODs/ (date of access: 11.02.2023)

Received 17.02.2023;
after revision 03.04.2023

D.I. Matvieiev, O.F. Lanovyy

METHODS OF SIMPLIFYING THE PROCESSING
OF INDEPENDENT PARTICLE SIMULATION SYSTEMS BASED
ON THE UNREAL ENGINE 4 ENVIRONMENT

The peculiarities of the development and practical implementation of systems designed to simulate the behavior of independent particles, the principles of their operation, the peculiarities of the model implementation and the process of facilitating processing are considered. Particular attention is paid to the process of facilitating processing at different processing levels of the particle simulation system. Factors that have a direct influence on it are considered, as well as the facilitation of shaders, as an important component of system speed.

K e y w o r d s: computer graphics, particles system, simplification, Unreal Engine 4, shaders.

МАТВЄЄВ Дмитро Ігорович, аспірант, асистент кафедри програмної інженерії Харківського національного університету радіоелектроніки. В 2018 р. закінчив Харківський національний університет радіоелектроніки. Область наукових досліджень — розробка та оптимізація комп'ютерних ігор та комп'ютерної графіки.

ЛАНОВИЙ Олексій Феліксович, канд. техн. наук, доцент, доцент кафедри програмної інженерії Харківського національного університету радіоелектроніки. В 1988 р. Закінчив Харківський інститут радіоелектроніки. Область наукових досліджень — імітаційне моделювання складних об'єктів та систем, оптимізаційне моделювання, застосування алгоритмів ML в моделюванні поведінки соціальних об'єктів.