УДК 004.77
DOI: 10.32626/2308-5916.2024-25.97-106

Д. В. Скоробогатський,
І. М. Кузьменко, канд. техн. наук

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського», м. Київ

МОДЕЛЮВАННЯ СИСТЕМ РЕАЛЬНОГО ЧАСУ З ВИКОРИСТАННЯМ RUDP ПРОТОКОЛУ ПЕРЕДАЧІ ДАНИХ

Ефективність систем реального часу забезпечує їх використання в різноманітних сферах за рахунок використання протоколів для передачі даних. Відомими протоколами транспортного рівня є Transmission Control Protocol (TCP) і User Datagram Protocol (UDP), що забезпечують таку передачу. Однак існують проблеми використання цих протоколів [1] в системах реального часу, де яких дані швидко змінюються на етапі виконання застосунку. А використання виключно одного з зазначених протоколів негативно впливає на стабільну роботу застосунку та збільшує ризик втрати актуальності даних.

Метою роботи є розробка та моделювання підсистеми передачі даних для систем реального часу, яка при сихронізації даних використовує Reliable UDP (RUDP) протокол [2]. Реалізація мережевого протоколу RUDP в підсистемі вирішує мережеві проблеми з затримкою, втратою та дуплікацією пакетів найбільш оптимальним шляхом під час real-time синхронізації даних датчиків і серверу і зменшує навантаження на пропускну здатність.

Задачами розробленої підсистеми, яка складається з сервера та клієнта є: зчитування даних, заданих користувачем системи; надійна передача повідомлень між сервером і клієнтом з використанням сокетів для роботи за протоколом RUDP; забезпечення найбільш оптимальної доставки повідомлень у випадку втрати пакетів, моделювання роботи підсистеми, порівняно з відомими UDP, TCP протоколами.

В результаті моделювання встановлено, що максимальне значення витраченого підсистемою часу становлять 2.03 секунди – для протоколу RUDP та 6.22 секунди – для протоколу TCP і тому в подальшому для передачі даних рекомендується протокол RUDP. У випадку втрати повідомлення, відбувається повторна відправка та експоненційно, за запропонованою формулою, обраховується час відправки. Якщо на першому кроці відправка відбувається відразу, то далі час час зростає експоненційно та сумарно є меншим, ніж час відправки за TCP протоколом.

Ключові слова: *протокол передачі, RUDP, TCP, втрата пакетів.*

Вступ. Для вирішення завдань в умовах часових обмежень використовуються системи реального часу. Критично важливі додатки, зокрема – управління транспортними засобами, медичне обладнання, авіаційні системи, телекомунікації та ін., базуються на використанні саме таких систем. І тому підвищення ефективності їх роботи є актуальним.

Огляд літератури та постановка задачі роботи. Системи реального часу ґрунтуються на контролі даних з високим рівнем надійності технічних характеристик процесів [3]. В літературі існують загальні підходи до побудови таких систем з використанням ряду протоколів для передачі даних [4]. Однак, на відміну від систем з потоковим відео, що вже досліджені [5], в літературі відсутній опис систем реального часу з використанням RUDP протоколу.

Тому **метою роботи** є розробка підсистеми що забезпечить в системах реального часу ефективність, надійність та базується на RUDP протоколі.

Задачами розробленої підсистеми, яка складається з сервера та клієнта є: зчитування даних, заданих користувачем системи; надійна передача повідомлень між сервером і клієнтом з використанням сокетів для роботи за протоколом RUDP; забезпечення найбільш оптимальної доставки повідомлень у випадку втрати пакетів, моделювання роботи підсистеми, порівняно з відомими UDP, TCP протоколами.

Архітектурна основа [6] підсистеми – клієнт-сервер. Ця архітектура охоплює розподілені системи, яким притаманна взаємодія клієнтів та серверу. Простіша форма клієнт-серверної системи – дворівнева архітектура, зазвичай, представляє собою серверний додаток, до якого підключаються клієнти за допомогою запитів. Сервер має авторитарний доступ до інформації і надає її у вигляді відповіді на запити.

Дана архітектура означає застосунок з графічним інтерфейсом для обміну даними з бізнес-логікою (базою даних або виділеним сервером).

У розробленій системі комунікація між сервером і клієнтом налаштована за рахунок підсистеми повідомлень. Ключова відмінність між клієнтом і сервером тут полягає в обробці різних типів повідомлень та різного підходу в підготовці до роботи.

Розроблювана система виконувалася з використанням ESP 8266 Nodemcu v3. Для передачі даних з плати ESP (або Arduino) було використано Wi-Fi бібліотеки. Модуль ESP8266 може працювати в режимі точки доступу і в режимі робочої станції, а також в обох режимах водночас. Зазвичай точка доступу має підключення до мережі і надає пристрою доступ до інтернету. Декілька робочих станцій у локальній мережі спілкуються через точку доступу. Кожен девайс має власну унікальну 48-бітову MAC-адресу в мережі.

Для розрізнення одної точки доступу від іншої, вони мають мережевий ідентифікатор, званий SSID (Service Set Identifier) – це ім'я мережі, що має довжину до 32 символів.

Для роботи було обрано датчик фірми Bosch Sensortec BME280, призначений для вимірювання атмосферного тиску, температури і вологості. Наприклад, діапазон вимірювань температури датчиком складає -40 – +85 °C з точністю – 0.01 °C.

Датчик підтримує два інтерфейси – I2C і SPI [7], зокрема SPI – послідовний периферійний інтерфейс – це повнодуплексний синхронний послідовний протокол послідовного зв'язку, який використовується для зв'язку на короткі відстані. Оскільки надійність відправки даних є пріоритетною у контексті розроблюваної системи, то вибір зупиняється на використанні протоколу I2C («inter-Integrated Circuit bus») – це двопровідний послідовний протокол для підключення низькошвидкісних пристроїв [8].

Для роботи з датчиком необхідно встановлено бібліотеки – Adafruit BME280 Library і Adafruit Sensor. Для роботи по протоколу I2C за допомогою скетчу I2C-сканер визначаємо адресу. Підключаємо датчик до плати Arduino згідно зі схемою з'єднань, завантажуємо скетч і запускаємо монітор послідовного порту.

Після підключення датчика BME280 до плати Arduino, було завантажено на плату приклад BME280test з бібліотеки Adafruit BME280 Library і використано його як основу взаємодії. Підсистема розроблювалася за допомогою мови програмування C++ [9] у зв’язці з Qt Framework. Qt Creator – крос-платформний засіб для підготовки прототипу на етапі розробки і фінальний продукт для користувача. Qt Framework має механізм сигналів і слотів [10], які і є обробниками, які з’єднуються з сигналами. Сигнали кидаються з певного класу, який містить оголошення цього сигналу.

Робота з сокетами відбувається через бібліотеку Winsock2.h. Найпоширенішим способом використання цього класу є прив'язка до адреси та порту за допомогою bind(), виклику recvfrom() і sendto() для передачі даних.

Архітектура та програмна реалізація. Архітектуру створеної підсистеми показано на рис. 1. Головне вікно застосунку (MainWindow) є водночас елементом компонування системи і графічного інтерфейсу. За мережеву роботу, маніпуляції з сокетами, передачу, отримання пакетів і синхронізацію даних відповідає модуль Мережа (Network layer). Модуль Network layer об’єднує відповідні для клієнта і сервера реалізації протоколів TCP, UDP, RUDP. Модуль Network layer взаємодіє з підсистемою повідомлень (Message Subsystem), яка інкапсулює серіалізацію\ десеріалізацію даних. Network layer відправляє дані у вигляді масиву байт через мережу. Agent встановлює з’єднання з сервером і синхронізує зареєстровані дані. Server відповідає за отримання даних від клієнта і їх передачу модулю графічного інтерфейсу. Це виконується через вбудований Адап-

тер, який є посередником між UI (user interface) і серверною логікою, забезпечуючи надійність мультипоточній системі (сервер винесений в окремий потік виконання).

Підсистема працює наступним чином: після підготовки клієнта, наступним є налаштування серверу, зокрема, – порта, який сервер буде відкривати для прослуховування, отримання даних та мережевого протоколу. На стороні агента підключається розроблена бібліотека, користувач налаштовує назву точки і пароль до Wifi-точки, яка буде використовуватися для отримання доступу.

Далі визначається код, який буде використаний системою для отримання даних за датчиків і синхронізований з сервером. Обирається надійність повідомлення для синхронізації, інтервал відправки, а також, чи потрібно синхронізувати дані датчика після підтвердження сервером даних. Клієнт відправляє зареєстровані дані у вигляді надійного повідомлення серверу. Після цього агент вмикається, клієнт відправляє запит на підключення до сервера і отримує відповідь, зв'язок встановлено. Після чого починається відправка даних у заданому інтервалі (за замовчуванням – 2 секунди). Сервер отримує ці повідомлення, відсилає оновлення до графічного інтерфейсу, щоб відобразити їх і скорегувати за потреби. Сервер має можливість конфігурувати наступні параметри синхронізації динамічно без перепрошивки плати агента: заданий час інтервалу, надійність(тип повідомлення) і «ввімкненість» певного датчика для синхронізації. Уся логіка серіалізації є схованою від користувача, однак користувач у межах серіалізації, може описати відповідні методи для своїх типів повідомлень.

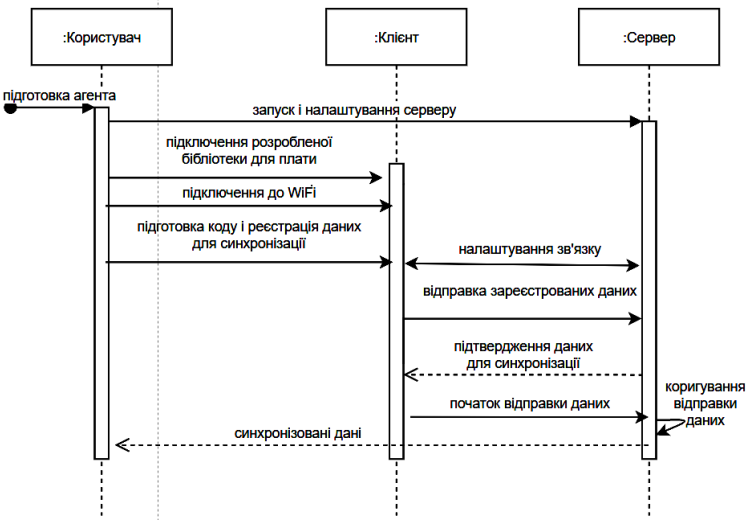

Рис. 1. *Архітектура створеної підсистеми*

Повідомлення у системі між клієнтом і сервером використовують об’єкти патерну Фабрика.

Серіалізація повідомлень включає перевірку існування об’єкту і виклик методу serialize для певного типу повідомлення, що повертає значення записаних байт. Після десеріалізації повідомлення опрацьовується системою. Для цього створено клас на основі патерна Singleton. Сервер і клієнт реєструють лише певні повідомлення для обробки. Наприклад, немає потреби клієнту реєструвати повідомлення на запит підключення до серверу, оскільки клієнт відправляє дані.

Опис роботи підсистеми. Для роботи з сокетами на Windows платформі було використано бібліотеку WinSock2.h. Для відправки даних використовуються методи send() і sendto(). Для отримання recv() і recvfrom() [8]. Дані відправляються з клієнта чи сервера, проте механізм отримання даних є блокуючим за замовчуванням, що створює проблему навіть для багатопоточної системи. Тобто, серверна та клієнтська частина, які відповідають за мережеву взаємодію винесені в окремий потік виконання. Наприклад, логіки сервера призупиняється якщо він намагається отримати дані від клієнта.

Виклики recvfrom() використовуються для отримання повідомлень із сокета (незалежно від того, чи підтримує він фактичне з'єднання). Виклик recv () зазвичай використовується лише на сокеті, що встановлює з’єднання (приклад, TCP).

Якщо сокет не отримує поточних пакетів, програмні виклики отримання даних recvfrom чекають надходження повідомлення і блокують виконання процесу, якщо сокет є блокучим.

Застосування select або poll підходів можуть бути використано для визначення часу надходження більшої кількості даних.

На кожному кроці оновлення серверної або клієнтської частини, система викликає функцію recvfrom() і перевіряє, чи містить вона якісь дані. Якщо даних для обробки немає, виконується інструкція переходу в сон для поточного потоку (на стороні сервера) на 5 мс. Якщо дані є, перевіряємо, чи отримано усі дані для поточного пакету і передаємо бінарні дані далі для подальшої конвертації у відповідний вид.

Функція recvfrom повертає будь-які доступні дані, не очікуючи отримання визначеної кількості байт, яку вимагає користувач окремим аргументом функції. Це призводить до того, що під час реалізації логіки стріму для серіалізації повідомлень у розроблюваній системі було отримано помилку з вирівнюванням на стороні плати ESP8266. Для виправлення помилки LoadStoreAlignmentCause потрібно зчитування або записувати дані за вирівняною адресою [9]. Тобто, розміщувати дані в оперативній пам'яті таким чином, щоб прискорити отримання доступу до них. Тому структура з char, int і float значення займатиме 12 байт, а не 9 очікуваних.

Однак, ESP8266 має обмеження на доступ до даних через вказівник. А саме, 32-бітовий доступ потрібно вирівняти за 32-бітовою межею а для 16-бітовий доступ – за16-бітовою межею.

У запропонованій підсистемі це вирішено з використанням функції memcpy, яка виконує низькорівневе копіювання даних без приведення даних до певного типу. Функція має наступну сигнатуру: *void * memcpy (void * destination, const void * source, size_t num).* Тобто, функція приймає вказівники, які вказують на void – функції не потрібно знати тип, з яким працює користувач, а важливо лише знати адресу пам'яті *destination*, яку необхідно заповнити *num* байтами з пам'яті *source.*

Додатково реалізовано шаблонну потоконадійну чергу запитів для сервера – призупити\завершити роботу, відправити повідомлення. Ця черга перевіряється сервером на кожному оновлені і при наявності запиту, оброблює його перед отриманням даних.

RUDP, доставляє пакети з організацією двох черг (для «надійних» і «ненадійних» повідомлень) і окремих порядкових номерів відповідно. Оскільки RUDP додає логіку, яка контролює надійну доставку\отримання, то, в першу чергу, йде обробка інформації про повідомлення, вже після самого повідомлення.

Відправка «надійного» повідомлення відбуватиметься до тих пір, доки ми не отримаємо підтвердження про його отримання або не буде перевищено ліміт доступних спроб, а тому їх необхідно деякий час зберігати. Також, повідомлення з порушеним порядком необхідно буферизувати перед обробкою програмою.

Якщо повідомлення «ненадійне», то встановлюємо індекс повідомлення на стороні клієнта та сервера.

Усі файли підсистеми – виконуваний файл, залежності і бібліотеки, запаковані в архів. Також в архів розміщено бібліотеку Ардуїно з прикладом використання датчика BME280 для плати ESP8266. Оскільки система має два основні потоки виконання логіки виконання з графічним інтерфейсом і серверу окремо, то рекомендується процесор з щонайменше двома ядрами. Об'єм оперативної пам'яті – до 15 мегабайт.

Інтервал відправки даних датчиків агента на стороні клієнта оновлюється в полі Sensor Interval протягом синхронізації на стороні сервера.

При роботі клієнта для кожного повідомлення, яке клієнт відправляє повторно, обраховується час відправки. Якщо на першому кроці T рівний RTT, тобто повторна відправка відбувається відразу, то далі час обраховується за запропонованою формулою

$$T = RTT \cdot \left(2^n - 1\right), \tag{1}$$

де n – номер поточної спроби повторно відправити пакет даних, RTT – Round-trip-time. Тобто клієнт збільшує експоненційно час відправки і цим зменшує навантаження на мережу.

Моделювання роботи розробленої підсистеми. Моделювання роботи розробленої підсистеми проведено за допомогою програмного застосунку Clumsy, що симулює такі мережеві проблеми, як затримка при відправленні у мілісекундах, відсоток втрати пакетів, які відсилаються/приймаються, відсоток дуплікації пакетів і втрата порядку пакетів при отриманні та відправленні.

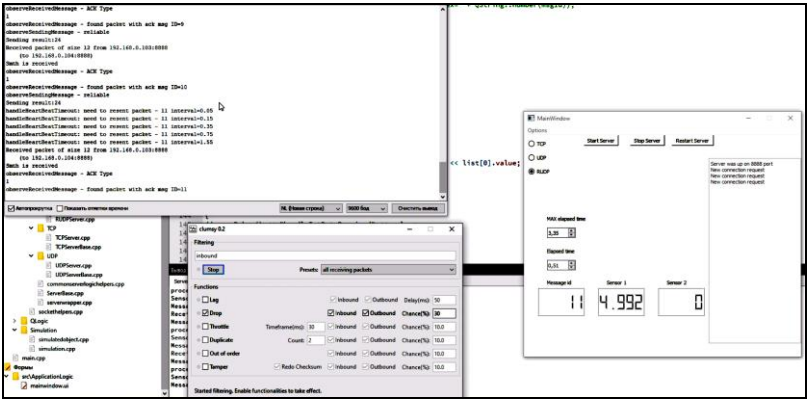

Рис. 2. *Моделювання часу роботи з використанням RUDP протоколу*

В ході моделювання проведено порівняння роботи підсистеми з використанням запропонованого протоколу RUDP та відомого протоколу TCP. При моделюванні в програмному застосунку Clumsy встановлено дуплікацію пакетів – 5% і симуляцію втрати пакетів в межах 15-30%.

Результати моделювання для протоколу TCP показали максимальне значення витраченого при моделюванні часу 6.22 сек за симуляції втрати пакетів 15%.

Отримані результати показують, що за роботи даної підсистеми та параметрів мережі, змодельованих застосунком Clumsy, повідомлення доходять. Для порівняння роботи, підсистема також моделює роботу з використанням протоколу RUDP за тих же параметрів мережі, змодельованих застосунком Clumsy. Модель підсистеми з використанням протоколу RUDP отримала максимальне значення витраченого часу 2.03 сек за симуляції втрати пакетів 15% і 3.35 сек за симуляції втрати пакетів 30%.

Тобто, на основі проведеного моделювання встановлено, що максимальне значення витраченого підсистемою часу з використанням протоколу RUDP в три рази менше, ніж з використанням протоколу TCP. Зокрема, за вищевказаних умов моделювання значення витраченого підсистемою часу становлять 2.03 сек – для протоколу RUDP та 6.22 сек – для протоколу TCP і тому в подальшому для передачі даних використано лише протокол RUDP.

Також в описаній підсистемі, за формулою (1) експоненційно встановлено час повторної відправки для кожного повідомлення якщо підси-

стема втрачає повідомлення. Зокрема повідомлення повторно буде надіслано через 0.15 сек. Після того, як це повідомлення не отримане, воно повторно надсилається через 0.35, 0.75, 1.55, 3.15 секунд, що відповідає часу, розрахованому відповідно до формули (1), де $RTT = 0{,}05$.

Слід зазначити, що змінюється час очікування, проте не про фактичний час доставки. Після останньої спроби перевідправити пакет, отримано ACK Type (підтвердження).

Обрахуємо: 2 + 0,05 + 0,15 + 0,35 + 0,75 = 3,3, що приблизно дорівнює Max elapsed time (3.35) на рисунку 2.

Порівняно з значенням, отриманим при роботі з TCP (6.22), значення при RUDP є майже вдвічі меншим, що дає змогу зробити висновок, RUDP зменшує час перевідправки повідомлень. Тобто, загальна кількість повторних спроб для повідомлення складає 6, а в цілому кількість спроб доставки встановлювалася в діапазоні 5-8. Після вичерпання усіх спроб, повідомлення вважається втраченим, а інша сторона підсистеми – недоступною.

Висновки. Моделювання підсистеми проведено за допомогою застосунку, що симулює втрати пакетів під час відправки, їх дуплікацію і порушення порядку. В результаті моделювання встановлено, що максимальне значення витраченого підсистемою часу становлять 2.03 секунди – для протоколу RUDP та 6.22 секунди – для протоколу TCP і тому в подальшому для передачі даних використано лише протокол RUDP.

У разі втрати повідомлень за умови не стабільної роботи мережі підсистема експоненційно збільшує час повторної відправки повідомлень. В цілому, кількість спроб повторної доставки повідомлень не перевищувала 8.

На основі аналізу літератури встановлено, що для реалізації систем реального часу використовують протокол TCP. Розроблена та реалізована підсистема для реалізації систем реального часу використовує RUDP протокол.

Реалізована підсистема складається з бібліотеки Arduino для плати агента і серверного застосунку з графічним інтерфейсом. Підсистеми інкапсулює логіку серіалізації даних і їх відправку з урахуванням вирівнювання адрес даних, що полегшує роботу користувача і розробника при подальшій підтримці коду. Робота описаної клієнт-серверної підсистеми полягає у синхронізації даних датчиків і сервера. Таким чином, розроблена та реалізована підсистема забезпечує стабільну та надійну роботу систем реального часу з використанням протоколу RUDP.

Список використаних джерел:

1. Reliable UDP Algorithms. URL: https://io7m.com/documents/udp-reliable.
2. Partridge C., Hinden R. Specification RFC 1151 Experimental – Version 2 of the Reliable Data Protocol (RDP). 1990.

3. Kopetz Hermann Real-Time Systems. Design Principles for Distributed Embedded Applications Springer. 2011. URL: https://www.springer.com/-gp/book/9781441982360.
4. Saurabh Tripathi, Shaurya Gupta, Tushar Babbar, Vishal Gupta. TCP-over-UDP for Real Time Applications. *International Journal of Scientific & Engineering Research.* 2013. Vol. 4. Is. 7. P. 357-360.
5. Weinrank F., Tuxen M., Rathgeb E. P. Integration of RTMFP in the OMNeT++ Simulation Environment, Proceedings of the OMNeT++ Community Summit. 2015. P. 1-4.
6. Архитектурные шаблоны и стили. URL: https://studfiles.net/preview/-6413815/page:11.
7. Difference between I2C and SPI (I2C VS SPI). URL: https://medium.com/-@rjrajbir24/difference-between-i2c-and-spi-i2c-vs-spi-c6a68d7242c4.
8. ESP8266 with BME280 using Arduino IDE. URL: https://randomnerdtutorials.com/esp8266-bme280-arduino-ide.
9. Страуструп Б. Язык программирования C++. Специальное издание = The C++ programming language. Special edition. Москва: Бином-Пресс, 2007. 1104 с. ISBN 5-7989-0223-4.
10. Qt documentation. Signals & Slots. URL: https://doc.qt.io/qt-5/signalsandslots.html
11. Recvfrom(2) – Linux man page. URL: https://linux.die.net/man/2/recvfrom.
12. Exception 9. Arduino Board. URL: https://readmodifywrite.github.io/blog/-html/2016/12/15/2016_12_15_memory_alignment_esp8266.html.
13. Липпман С. Б., Лажойе Ж. Язык программирования C++. Базовый курс. Москва: Диалектика-Вильямс, 2019. 1120 с.
14. Мейерс С. Эффективный и современный C++. 42 рекомендации по использованию C++11 и C++14. Москва: Вильямс, 2016. 304 с.

A MODELING OF REAL-TIME SYSTEM WITH RUDP DATA TRANSMISSION PROTOCOL

The efficiency of real-time systems makes them applicable in various fields by utilizing protocols for data transmission. Well-known transport layer protocols include Transmission Control Protocol (TCP) and User Datagram Protocol (UDP), which facilitate such transmission. However, there are challenges in using these protocols [1] in real-time systems where data rapidly changes during application execution. Exclusively using one of these protocols adversely affects the stable operation of the application and increases the risk of data obsolescence.

The objective of this work is to develop and model a data transmission subsystem for real-time systems that utilizes the Reliable UDP (RUDP) protocol [2] for data synchronization. Implementing the RUDP network protocol in the subsystem addresses network issues such as delay, packet loss, and duplication in the most optimal way during real-time data synchronization between sensors and the server, reducing the load on bandwidth.

The tasks of the developed subsystem, consisting of a server and a client, include: reading user-specified system data, reliable message transmission between the server and the client using sockets for RUDP protocol operation, ensuring the most optimal message delivery in case of packet loss, subsystem operation modeling compared to well-known UDP and TCP protocols.

The simulation results reveal that the maximum time spent by the subsystem is 2.03 seconds for the RUDP protocol and 6.22 seconds for the TCP protocol. Therefore, RUDP is recommended for data transmission in the future. In the event of message loss, retransmission occurs exponentially, calculated using the proposed formula. If the initial transmission occurs immediately, subsequent times exponentially increase but remain collectively less than the transmission time for the TCP protocol.

Key words: *transmission protocol, RUDP, TCP, packet loss.*

Отримано: 04.01.2024

УДК 004.94:53.04:004.925
DOI: 10.32626/2308-5916.2024-25.106-113

О. С. Станіславів,
О. О. Жолтовський,
О. А. Смалько, канд. пед. наук

Кам'янець-Подільський національний університет
імені Івана Огієнка, м. Кам'янець-Подільський

КОМП'ЮТЕРНЕ МОДЕЛЮВАННЯ ДЕЯКИХ ПРИРОДНИХ ПРОЦЕСІВ ДЛЯ ГЕНЕРАЦІЇ ЛАНДШАФТІВ

У статті розглядаються різні підходи до формування рельєфних структур з натуралістичними формами, що є корисним для подальшого їх використання в ігровій індустрії, у середовищах розширеної реальності, для створення якісного, правдоподібного візуального контенту.

Дослідивши значну частину математичного інструментарію ландшафтоутворення, автори виділяють серед багатьох методів, заснованих на фізиці, опис таких природних процесів, як ерозія, седиментація, повзучість матеріалів, завдяки чому можна синтезувати реалістичний рельєф місцевості.

Деякі методи розв'язання задач чисельної гідроаеромеханіки зі спрощеними умовами є ефективними для моделювання різних особливостей ландшафту. Для синтезування крупних структур рельєфу, взявши за основу нестисливу нев'язку рідину, можна скористатися, наприклад, рівнянням Ейлера. Дрібніші ландшафтні компоненти можна формувати за допомогою рівнянь мілководдя. Їх же можна використовувати для моделювання ерозійних процесів, спричинених руйнуванням ґрунту або гірських порід водним потоком. За потреби симуляції ерозії русла річки варто застосовувати напівемпіричне сімейство рівнянь закону потужності потоку. Природних форм до рельєфу додасть також рівняння Бейтмана-Бюргерса, це допоможе змоделювати різні аспекти руху рідини, такі як течія в річках, морях, океанах та