УДК 004.421.2
DOI https://doi.org/10.32782/tnv-tech.2024.4.9

ОГЛЯД І АНАЛІЗ АЛГОРИТМІВ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ ІГРОВИХ СВІТІВ

Марчук Г. В. *– старший викладач кафедри комп’ютерних наук Державного університету «Житомирська політехніка»*
ORCID ID: 0000-0003-2954-1057

Левківський В. Л. *– доктор філософії з інженерії програмного забезпечення, доцент кафедри комп’ютерних наук Державного університету «Житомирська політехніка»*
ORCID ID: 0000-0002-1643-0895

Харченко А. В. – *магістрант кафедри комп’ютерних наук Державного університету «Житомирська політехніка»*
ORCID ID: 0009-0007-7995-2352

Марчук Д. К. *– старший викладач кафедри комп’ютерних наук Державного університету «Житомирська політехніка»*
ORCID ID: 0000-0001-8675-8047

Комп'ютерні ігри в сучасному світі стали важливою частиною культури та розваг. Одним з основних елементів, що впливає на сприйняття гри, є ігровий світ, де розгортаються події. З розвитком ігрової індустрії та зростанням популярності відеоігор виникає потреба у швидкому створенні нових ігрових світів. У цьому контексті застосування алгоритмів процедурної генерації стає корисним, оскільки вони дозволяють автоматично створювати ігрові середовища із заданими характеристиками. Процедурна генерація ігрових світів – це метод, який дозволяє створювати різноманітні ігрові середовища за допомогою алгоритмів і випадкових процесів. На сьогодні існують різні алгоритми процедурної генерації, такі як алгоритми генерації шумів, фрактальної геометрії, клітинні автомати. Метою цього дослідження є аналіз алгоритмів, методів та підходів до процедурної генерації тривимірних ігрових світів, опис їх роботи та визначення особливостей. В роботі були проаналізовані алгоритми Perlin Noise, Improved Noise, Simplex Noise, Diamond-Square, Fractal Noise. В якості показників було обрано швидкість роботи, якість генерації та раціональність використання пам’яті. Якість генерації оцінювалась щодо реалістичності та різноманітності отриманих локацій. Для ігор, де важливою є швидкість генерації ландшафтів більше підходять Simplex Noise та Diamond-Square. Diamond-Square, хоча і простий у реалізації, менш гнучкий для великих світів. Якщо ж потрібна максимально деталізована карта, варто звернути увагу на Fractal Noise, але він потребує значних обчислюваних ресурсів. Для отримання цікавих та різноманітних ігрових світів краще використовувати комбінації декількох алгоритмів, наприклад один для створення базового ландшафту а інший для деталізації. Алгоритми процедурної генерації дозволяють створювати великі, деталізовані світи з мінімальним ручним втручанням, що значно економить час і ресурси та робить гру більш інтерактивною та непередбачуваною.

Ключові слова: *Perlin Noise, Improved Noise, Simplex Noise, Diamond-Square, Fractal Noise, процедурна генерація, розробка ігор, створення локацій.*

Marchuk G. V., Levkivskyi V. L., Kharchenko A. V., Marchuk D. K. Review and analysis of procedural game world generation algorithms

Computer games have become an essential part of modern culture and entertainment. One of the key elements influencing the player's perception of a game is the game world where the events unfold. With the growth of the gaming industry and the increasing popularity of video games, there is a need for rapid creation of new game worlds. In this context, the application

of procedural generation algorithms proves useful, as they allow for the automatic creation of game environments with specific characteristics. Procedural generation of game worlds is a method that enables the creation of diverse game environments using algorithms and random processes. Today, various procedural generation algorithms exist, such as noise generation algorithms, fractal geometry, and cellular automata. The purpose of this research is to analyze algorithms, methods, and approaches to procedural generation of 3D game worlds, describe their operation, and identify their unique features. The study examined algorithms such as Perlin Noise, Improved Noise, Simplex Noise, Diamond-Square, and Fractal Noise. The chosen evaluation criteria included performance speed, generation quality, and memory efficiency. The quality of generation was assessed in terms of the realism and variety of the generated locations. For games where landscape generation speed is crucial, Simplex Noise and Diamond-Square are more suitable. While Diamond-Square is easy to implement, it is less flexible for large worlds. If a highly detailed map is required, Fractal Noise is the best choice, though it demands significant computational resources. For creating interesting and diverse game worlds, it is better to combine several algorithms, for instance, one for creating the base landscape and another for adding detail. Procedural generation algorithms allow the creation of vast, detailed worlds with minimal manual intervention, significantly saving time and resources while making the game more interactive and unpredictable.

Key words: *Perlin Noise, Improved Noise, Simplex Noise, Diamond-Square, Fractal Noise, procedural generation, game development, location creation.*

Постановка проблеми. У зв'язку з розвитком ігрової індустрії та загальної популярності відеоігор виникає проблема зі швидким створенням ігрових світів. Корисним в даній ситуації може бути використання алгоритмів процедурної генерації, за допомогою яких з'являється можливість автоматично створювати ігрові середовища із зазначеними параметрами.

Процедурна генерація є важливим та актуальним елементом у розробці сучасних ігор та віртуальних світів. Цей підхід дозволяє створювати дуже різноманітні та нескінченні ігрові середовища, забезпечуючи унікальний досвід для гравця кожного разу, коли він запускає гру. Важливість процедурної генерації полягає в її здатності створювати величезні, яскраві світи з куточками, які можна досліджувати, зберігаючи при цьому зацікавленість гравця. Крім того, цей підхід оптимізує використання ресурсів і дозволяє розробляти ігрові проекти з меншим бюджетом і в коротші терміни. Процедурна генерація продовжує впливати на індустрію розробки ігор і стала важливим інструментом для досягнення варіативності, реалістичності та глибини ігрових світів.

Аналіз цих алгоритмів дасть змогу визначити їх переваги та недоліки, їх складність та загальну зручність, як в реалізації, так і в їх використанні у ігрових застосунках.

Аналіз відомих результатів досліджень. На сьогоднішній день вже існує безліч алгоритмів та методів для автоматичного створення ігрового середовища. У роботі [1] представлено новий підхід до процедурної генерації наративних світів, що включає два етапи: модель наративного світу генерується лише один раз для певної історії; форма наративного світу використовується для створення одного (або кількох) можливих наративних світів для цієї історії. У роботі [2] описано створений фреймворк ProcTHOR – структуру для процедурної генерації середовищ. ProcTHOR дозволяє відбирати великі набори даних інтерактивних, настроюваних і продуктивних віртуальних середовищ, щоб навчати й оцінювати втілені агенти для завдань навігації, взаємодії та маніпулювання. Авторами роботи [3] представлено підхід до генерації рівнів у стилі підземелля. Використано гібридну техніку, яка поєднує контекстно-вільні граматики для створення опису рівнів і клітинні автомати, для створення фізичного простору. Доведено, що гібридні підходи до автоматизованого проектування рівнів мають значну цінність. На сьогоднішній

день не так багато досліджень щодо процедурного 3D-моделювання печер і подібних замкнених природних просторів. У статті [4] представлено модульний конвеєр для процедурного створення підземних печер у реальному часі, які будуть використовуватися як частина більших ландшафтів у ігрових світах. Авторами запропоновано підхід, який можна повністю реалізувати за допомогою технології GPGPU (General-Purpose Computing on Graphics Processing).

Незважаючи на те, що розробники ігор і академічні дослідники широко досліджували процедурне створення контенту, не вистачає методів обробки процедурного створення квестів. У статті [5] представлено новий метод генерації квестів на основі генетичних алгоритмів і автоматизованого планування. Поєднуючи планування з еволюційною стратегією пошуку, керуючись сюжетними дугами, запропонований метод може генерувати узгоджені квести на основі певної структури оповіді. Попередні результати показують, що квести, створені за допомогою запропонованого методу, майже не поступаються квестам, створеним професіоналами з розробки ігор. У роботі [6] представлено нову систему AutoBiomes, яка здатна ефективно створювати величезні території з правдоподібним розподілом біомів і різними просторовими характеристиками. Основна ідея полягає в поєднанні кількох синтетичних процедурних методів генерації рельєфу з цифровими моделями рельєфу і спрощеним кліматичним моделюванням. Розроблена система дозволяє швидко створювати реалістичні ландшафти.

Процедурна генерація – це потужний інструмент, який дозволяє створювати великі обсяги різноманітних даних на основі набору правил та обмежень. Цей підхід знаходитиме широке застосування в різних галузях, від розробки ігор до наукових досліджень включаючи комп'ютерне зір та обробку природної мови. На сьогодні є актуальним створення синтетичних даних, які дозволяють моделювати рідкісні чи небезпечні ситуації, які важко чи неможливо відтворити у реальному світі. Створення фотореалістичних зображень людей, об'єктів та сцен [7]. Генерація синтетичних даних для навчання моделей автономного водіння у різних дорожніх умовах [8-9]. Отже, важливо забезпечити, щоб синтетичні дані були реалістичними та містили всі необхідні для навчання моделі характеристики.

Мета роботи. Метою даного дослідження є аналіз алгоритмів процедурної генерації тривимірного ігрового світу.

Виклад основного матеріалу. Процедурна генерація широко використовується при розробці ігор. В деяких випадках, даний функціонал є повноцінною ігровою механікою, де гравцю надається доступ до налаштувань ігрового середовища перед початком основної ігрової сесії. Цими параметрами можуть бути: розмір, наявність певних локацій, так званих «біомів», їх тип, кількість ресурсів, ворогів тощо. Насправді, їх кількість та різноманітність обмежуються лише фантазією та ресурсами розробників. Генерація ігрового світу складається з різних елементів, що формують фізичну основу світу, таких як місцевість і ландшафт. До них належать рельєф, рівнини, гори, річки, озера та інші природні форми. Також генерація включає в себе розміщення різних об'єктів у просторі, таких як будівлі, рослини і тварини. Крім того, створення ігрового світу може включати елементи штучного інтелекту для створення поведінки персонажів і взаємодії з гравцями. У сукупності ці елементи створюють унікальний і захопливий ігровий досвід.

Рівень якості результату процедурних алгоритмів генерації світів може значно варіюватися залежно від конкретних потреб проекту, його аудиторії та цілей розробки. Наприклад, один проект може бути націлений на створення світу з великою кількістю деталей, що включає в себе реалістичні текстурами та складні

архітектурні елементи, в той час як інший може спробувати створити більш абстрактний або мінімалістичний стиль.

У будь-якому разі, рівень якості має бути достатнім, щоб забезпечити приємний ігровий досвід для користувача і відповідати стандартам, які він очікує від ігрового продукту. Результат генерації має приймати складні та цікаві форми, але, в той же час, бути зрозумілим, тобто вміщувати в себе впізнавані патерни, візерунки.

Одним із найбільш відомих псевдовипадкових шумів є Perlin Noise – це алгоритм для створення процедурних мап шуму, який широко використовується у процедурній генерації. Він був розроблений Кеном Перліном в 1982 році спеціально для фільму «Трон», за що Перлін отримав Академічну нагороду за технічні досягнення [10]. У 2002 році алгоритм був удосконалений до Покращеного Шуму (Improved Noise) самим Кеном Перліном. Проблема звичайного шуму полягає в тому, що він повністю складається з випадкових значень, тому загальний результат буде доволі грубим, адже ймовірність того, що сусідні пікселі будуть мати схожі значення – доволі низька. Perlin Noise та Improved Noise створює доволі приємне гладке зображення і робить він це, якщо коротко, завдяки генерації випадкових чисел на більшій сітці та інтерполюючи потім значення між ними на меншій, що в кінці видає набагато приємніший результат.

Хоча точний математичний опис Perlin Noise може бути досить складним, загальна ідея полягає в наступному: нехай P(x, y, z) – значення шуму в точці (x, y, z), тоді:

$$P(x,y,z) = \sum \Big(grad\big(G(i,j,k)\big) \cdot (x-i, y-j, z-k) * fade(x-i, y-j, z-k) \Big), \quad (1)$$

де G(i, j, k) – градієнт у точці гратки з індексами (i, j, k), градієнт показує напрямок найбільшого зростання функції. У контексті Perlin Noise градієнт визначає це буде яма чи висота; fade(x) – функція плавного згасання, яка забезпечує плавний перехід між значенням шуму в різних точках; · – скалярний добуток двох векторів. Він використовується для обчислення проекції вектора (x-i, y-j, z-k) на напрямок градієнта.

Сума обчислюється по всіх точках сітки, що знаходяться поблизу точки (x, y, z). Для отримання значення шуму у довільній точці простору використовується інтерполяція між найближчими точками сітки. Класичний метод інтерполяції, що використовується в Perlin Noise (рис. 1), називається бікубічною інтерполяцією. Вона дозволяє отримати плавні переходи між різними точками гратки.

Для обчислення значення шуму в довільній точці простору, обчислюється скалярний добуток вектора градієнта найближчої точки сітки на вектор, що з'єднує цю точку з обчислюваною точкою. Цей скалярний добуток масштабується за допомогою функції, яка залежить від відстані до найближчої точки сітки.

Шум Перліна є потужним інструментом для створення різноманітних візуальних ефектів. Його гнучкість та ефективність зробили його незамінним у багатьох областях комп'ютерної графіки.

Алгоритм Diamond-Square дозволяє створювати двовимірні мапи висот. Алгоритм був розроблений Альоном Фурньє, Доном Фасселлом та Лореном Карпентером і вперше був продемонстрований на конференції SIGGRAPH у 1982 році [12]. Diamond-Square є покращенням алгоритму Midpoint Displacement. Обидва алгоритми використовуються для генерації фрактальних ландшафтів, але Diamond-Square вносити кілька важливих удосконалень. Diamond-Square розбиває квадрат на чотири рівні частини, а потім обчислює висоту центральної точки (діамантовий крок) та чотирьох кутових точок (квадратний крок). Це дозволяє обчислити висоту

кожної точки лише один раз, на відміну від Midpoint Displacement, де деякі точки можуть обчислюватися кілька разів. Структура Diamond-Square дозволяє легко розпаралелити обчислення, що прискорює генерацію великих ландшафтів. Алгоритм Diamond-Square може генерувати ландшафти довільних розмірів, тоді як Midpoint Displacement може бути обмеженим певними співвідношеннями сторін [13].

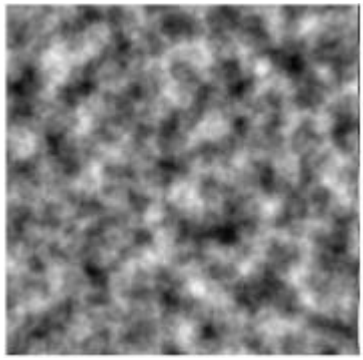

Рис. 1. Результат роботи алгоритму Perlin Noise [11]

Алгоритм Diamond-Square іноді називають Plasma Fractal через його здатність генерувати зображення, що нагадують плазму (рис. 2). Це порівняння базується на візуальній схожості результатів роботи алгоритму з природними плазмовими утвореннями.

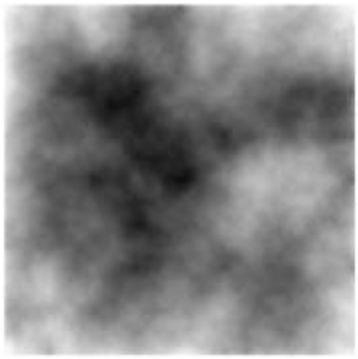

Рис. 2. Приклад зображення карти висот, алгоритм Diamond-Square [11]

Diamond-Square є доволі простим для розуміння та відносно легким в реалізації. Для коректної роботи він вимагає, щоб розмір ширини та висоти полотна був рівним 2^N+1, де N – ціле число, що більше або дорівнює 0. Базові кроки алгоритму (рис. 3):

1. Створюємо квадратну сітку та задаємо висоту для чотирьох кутових точок.
2. Обчислюємо висоту центральної точки квадрата як середню арифметичну висот чотирьох кутових точок плюс випадкове значення (шум).
3. Обчислюємо висоту кожної середньої точки сторони квадрата як середню арифметичну висот двох сусідніх кутових точок та центральної точки плюс випадкове значення (шум).

4. Ділимо кожен отриманий квадрат на чотири частини та повторюємо кроки 2 та 3 до досягнення бажаної деталізації.

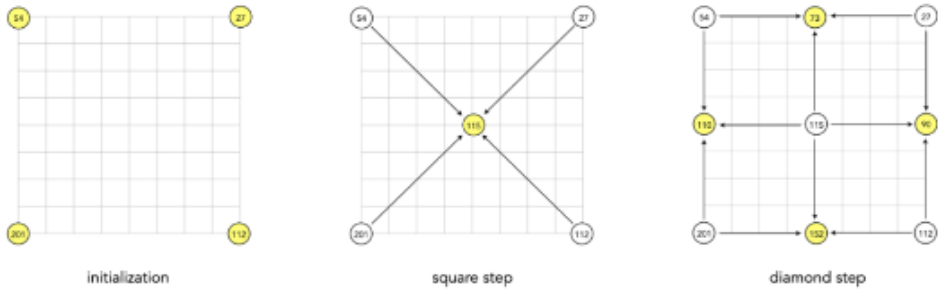

Рис. 3. Базові кроки алгоритму Diamond-Square [12-14]

Формально алгоритм можна описати наступним чином:

1. H(x, y) – висота точки з координатами (x, y) на сітці.
2. Задаємо початковий розмір квадрата N.
3. Вибираємо випадкові значення для чотирьох кутових точок H(0, 0), H(0, N-1), H(N-1, 0) та H(N-1, N-1).
4. Diamond step. Для кожного квадрата зі стороною довжини s обчислюємо:

$$H\left(x+\frac{s}{2},y+\frac{s}{2}\right)=\frac{H(x,y)+H(x+s,y)+H(x,y+s)}{4}+random(-r,r), \qquad (2)$$

де (x, y) нижній лівий кут квадрата, r амплітуда шуму, яка зазвичай зменшується з кожною ітерацією для створення більш гладких деталей.

5. Square step. Для кожної сторони квадрата зі стороною довжини s обчислюємо:

$$H\left(x+\frac{s}{2},y\right)=\frac{H(x,y)+H(x+s,y)+H\left(x+\frac{s}{2},y+\frac{s}{2}\right)}{3}+random(-r,r), \qquad (3)$$

$$H\left(x,y+\frac{s}{2}\right)=\frac{H(x,y)+H(x,y+s)+H\left(x+\frac{s}{2},y+\frac{s}{2}\right)}{3}+random(-r,r), \qquad (4)$$

де s – розмір поточного квадрата, random() – випадкове число із заданим розподілом (наприклад, нормальний розподіл).

Формули 3 та 4 використовуються і для інших сторін квадратів.

6. Ділимо кожен квадрат на чотири рівні частини. Повторюємо кроки Diamond та Square для кожного нового квадрата зі зменшеним значенням s та r.

Алгоритм Diamond-Square є популярним методом генерації фрактальних ландшафтів, але має свої обмеження та недоліки, які варто враховувати при його застосуванні. Незважаючи на те, що алгоритм створює досить реалістичні ландшафти, у них часто можна помітити певну "прямолінійність" рельєфу, особливо на великих масштабах. Результат роботи алгоритму залежить від значень, заданих у чотирьох кутах початкового квадрата. Це може призвести до того, що отримані

ландшафти будуть виглядати надто одноманітними або, навпаки, надто хаотичними. Алгоритм Diamond-Square найкраще підходить для генерації "класичних" гірських ландшафтів. Для створення більш складних та різноманітних ландшафтів (наприклад каньйонів, печер чи підводних рельєфів) можуть знадобитися додаткові модифікації чи комбінації з іншими алгоритмами.

Однією з проблем алгоритму при використанні його для генерації тривимірних ландшафтів є наявність піків, що виглядають недоречно. Це вирішується додаванням додаткових етапів згладжування.

Fractal Noise – унікальний спосіб створення шуму використовуючи фрактальний підхід. Одна з найбільших сильних сторін фрактального шуму полягає в тому, що ландшафт, згенерований таким чином, по суті є віртуально нескінченним. Тобто, є можливість продовжувати генерацію в будь-якому напрямку, використовуючи попередньо визначену точку відліку [15]. Фрактальний шум будується з урахуванням рекурсивних математичних формул, які дозволяють створювати нескінченно деталізовані зображення.

Fractal Noise будується з урахуванням базової функції шуму, яка повторюється у різних масштабах і змішується. Цей процес ітеративно повторюється, створюючи детальніші і складніші структури.

$$fractalNoice(\mathrm{x,y}) = sum\left(amplitude[i] * noice\left(x * frequency[i], y * frequency[i]\right)\right) \quad (5)$$

де, x, y – координати точки; amplitude[i] – амплітуда; frequency[i] – частота і-ої октави; noise– функція базового шуму.

Simplex Noise є більш ефективним та менш гучним варіантом Perlin Noise, яка вирішує проблему спрямованості та забезпечує більш плавні градієнти. Алгоритм краще працює з великою кількістю вимірів. Фрактал в даному випадку використовується як опис об'єкта, що створений з потенційно нескінчених частин самого себе.

На рисунку 4 зображений результат роботи алгоритму Simplex Noise, візуально він виглядає краще за Perlin Noise. Мапа висот, хоч і достатньо деталізована для створення базової 3D-моделі, все ж потребує додаткової обробки. Ландшафт, згенерований за її допомогою, матиме одноманітний вигляд та недостатньо виражені перепади висот для реалістичної візуалізації.

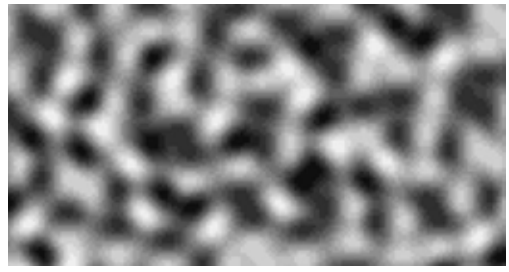

Рис. 4. Результат алгоритму Simplex Noise

Щоб зробити зображення більш детальним та текстурованим можна використати фрактальний метод. Цей метод дозволяє додати додаткові деталі до зображення шляхом накладання на нього кількох шарів шуму. На рисунку 5 представлено результат цієї операції, яка надає зображенню більш грубого вигляду та різкіших переходів між висотами.

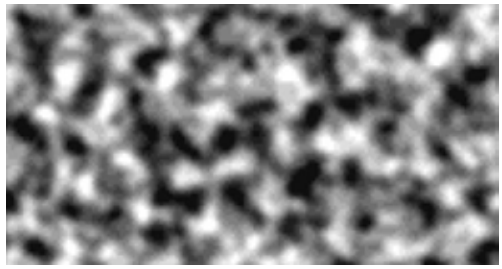

Рис. 5. Результат Fractal Noise з октавою 2

Слід зазначити, що при кожній ітерації накладання шумів, необхідно змінювати значення наступних параметрів:

1. Амплітуда, яка визначає різкість переходів значень шуму. Маленька амплітуда створюватиме доволі плоскі ландшафти, а висока – гори та океани.
2. Частота, яка визначає кількість деталей, зображених на одиниці простору.

Для описаних алгоритмів було проведено дослідження щодо швидкості та якості роботи і раціональності використання пам'яті. Результати проведеного аналізу представлено у таблиці 1.

Таблиця 1

Порівняння алгоритмів генерації ігрових світів

Назва алгоритму	Швидкість	Якість	Використання пам'яті
Perlin Noise	Середня	Помірна	Низьке
Improved Noise	Висока	Добра	Середнє
Simplex Noise	Висока	Висока	Низьке
Diamond-Square	Дуже висока	Добра	Високе
Fractal Noise	Залежить від кількості октав	Дуже висока	Високе

Аналіз алгоритмів генерації шумів показує, що Simplex Noise є оптимальним за співвідношенням швидкості та якості, що робить його привабливим для ігор з великою кількістю генерованого контенту. Fractal Noise, незважаючи на вищу якість, має обмеження у використанні через високу обчислювальну складність при збільшенні кількості октав. Improved Noise є компромісним варіантом, а Diamond-Square – є простим у реалізації, але менш гнучким. Fractal Noise забезпечує найвищу якість, але за рахунок значних витрат ресурсів.

Висновки. В даному дослідженні було проведено аналіз алгоритмів процедурної генерації для тривимірного ігрового світу. В роботі були проаналізовані алгоритми Perlin Noise, Improved Noise, Simplex Noise, Diamond-Square, Fractal Noise.

Алгоритми шумів створюють масив псевдовипадкових значень, що зручно використовувати як мапи висот для створення початкового тривимірного ландшафту. Вибір доцільного алгоритму залежить загалом від потреб розробників та вимог клієнтів. При потребі досягнення високої швидкодії створення шуму чудовим варіантом буде використання алгоритмів Simplex Noise або Diamond-Square, проте останній не дуже зручний для генерації нескінченних ландшафтів, потребує більше пам'яті та особливого розміру полотна. Для досягнення найкращої якості

мапи висот слід використовувати принцип фракталів, що полягає в накладанні шумів один на одне. Слід зазначити, що збільшення кількості октав зменшує швидкість генерації шуму, через збільшення об'єму роботи.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ:

1. Balint J. T., Bidarra R. Procedural Generation of Narrative Worlds. *IEEE Transactions on Games.* Vol. 15, no. 2. Pp. 262-272. June 2023. DOI: https://doi.org/10.1109/TG.2022.3216582.

2. Deitke M., VanderBilt E., Herrasti A. et all. ProcTHOR: Large-Scale Embodied AI Using Procedural Generation. *Advances in Neural Information Processing Systems.* Vol. 35. Pp. 5982-5994. 2022.

3. Gellel A., Sweetser P. A Hybrid Approach to Procedural Generation of Roguelike Video Game Levels. *In Proceedings of the 15th International Conference on the Foundations of Digital Games (FDG '20). Association for Computing Machinery, New York, NY, USA,* Article 3. Pp. 1–10. 2020. DOI: https://doi.org/10.1145/3402942.3402945.

4. Mark B., Berechet T., Mahlmann T., Togelius J. Procedural Generation of 3D Caves for Games on the GPU. *Paper presented at Foundations of Digital Games, United States.* 2015. URL: https://lucris.lub.lu.se/ws/portalfiles/portal/6067634/5464988.pdf.

5. Soares de Lima E., Feijó B., Furtado A. L. Procedural Generation of Quests for Games Using Genetic Algorithms and Automated Planning. *18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames), Rio de Janeiro, Brazil.* Pp. 144-153. 2019. DOI: https://doi.org/10.1109/SBGames.2019.00028.

6. Fischer R., Dittmann P., Weller R. et al. AutoBiomes: procedural generation of multi-biome landscapes. *The Visual Computer.* Vol. 36. Pp. 2263–2272. 2020. DOI: https://doi.org/10.1007/s00371-020-01920-7.

7. Марчук Д.К., Левківський В.Л., Марчук Г.В., Голенко М.Ю. Система розпізнавання дактильної мови української абетки. *Вчені записки Таврійського національного університету імені В.І. Вернадського. Серія: Технічні науки.* Т. 33 (72), № 6. С. 109–114. 2022. DOI: https://doi.org/10.32782/2663-5941/2022.6/19

8. Levkivskyi V., Marchuk D., Lobanchykova N., Pilkevych I., Salamatov D. Available parking places recognition system. *CEUR Workshop Proceedings 4th Workshop for Young Scientists in Computer Science & Software Engineering.* Vol. 3077, Pp. 123–134. 2022. URL: http://ceur-ws.org/Vol-3077/paper07.pdf

9. Марчук Д.К. Аналіз сучасних алгоритмів виявлення і розпізнавання об'єктів з відеопотоку для систем управління паркуванням в реальному часі. *Вісник Хмельницького національного університету. Серія: «Технічні науки».* № 3. С. 339-347. 2023. DOI: https://www.doi.org/10.31891/2307-5732-2023-321-3-17-23

10. Perlin K. Improving noise. *ACM Trans.* Graph. 21, 3. Pp. 681–682. 2002. DOI: https://doi.org/10.1145/566654.566636.

11. Ryan J. S., Cowling P., Alfred Walker J. Procedural generation using spatial GANs for region-specific learning of elevation data. *IEEE Conference on Games (CoG).* IEEE, 2019.

12. Fournier A., Fussell D., Carpenter L. Computer rendering of stochastic models. *Commun.* ACM 25, 6. Pp. 371–384. 1982. DOI: https://doi.org/10.1145/358523.358553.

13. Lautakoski J. Procedurell generering av terräng Perlin noise eller Diamond-Square: med fokus på exekveringstid och framkomlighet. *Dissertation.* 2016. URL: https://urn.kb.se/resolve?urn=urn:nbn:se:his:diva-12854.

14. Gavin S P Miller. The definition and rendering of terrain maps. *SIGGRAPH Comput.* Graph. 20, 4. Pp. 39–48. 1986. DOI: https://doi.org/10.1145/15886.15890.

15. Adrian C., Liarokapis F. Fractal nature-generating realistic terrains for games. *7th International Conference on Games and Virtual Worlds for Serious Applications (VS-Games).* IEEE, 2015.

REFERENCES:

1. Balint, J. T., Bidarra, R. (2023). Procedural Generation of Narrative Worlds. *IEEE Transactions on Games,* 15, 2. Pp. 262-272. DOI: https://doi.org/10.1109/TG.2022.3216582. [in English]

2. Deitke, M., VanderBilt, E., Herrasti, A. et all. (2022). ProcTHOR: Large-Scale Embodied AI Using Procedural Generation. *Advances in Neural Information Processing Systems,* 35, 5982-5994. [in English]

3. Gellel, A., Sweetser, P (2020). A Hybrid Approach to Procedural Generation of Roguelike Video Game Levels. *In Proceedings of the 15th International Conference on the Foundations of Digital Games (FDG '20). Association for Computing Machinery, New York, NY, USA,* Article 3, 1–10. DOI: https://doi.org/10.1145/3402942.3402945. [in English]

4. Mark, B., Berechet, T., Mahlmann, T., & Togelius, J. (2015). Procedural Generation of 3D Caves for Games on the GPU. *Paper presented at Foundations of Digital Games, United States.* URL: https://lucris.lub.lu.se/ws/portalfiles/portal/6067634/5464988.pdf. [in English]

5. Soares de Lima, E., Feijó, B., and Furtado, A.L. (2019). Procedural Generation of Quests for Games Using Genetic Algorithms and Automated Planning. *18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames), Rio de Janeiro, Brazil,* 144-153, DOI: https://doi.org/10.1109/SBGames.2019.00028. [in English]

6. Fischer, R., Dittmann, P., Weller, R. et al. (2020). AutoBiomes: procedural generation of multi-biome landscapes. *Vis Comput* 36, 2263–2272 DOI: https://doi.org/10.1007/s00371-020-01920-7. [in English]

7. Marchuk, D.K., Levkivskyi, V.L., Marchuk, G.V. & Holenko, M.Yu. (2022). Systema rozpiznavannia daktylnoi movy ukrainskoi abetky [Dactylic Language Recognition System of the Ukrainian Alphabet]. *Vcheni zapysky Tavriiskoho natsionalnoho universytetu imeni V.I. Vernadskoho. – Academic notes of the V.I. Vernadsky Tavriya National University* 33 (72), 6, 109–114, DOI: https://doi.org/10.32782/2663-5941/2022.6/19. [in Ukrainian]

8. Levkivskyi, V., Marchuk, D., Lobanchykova, N., Pilkevych, I. & Salamatov, D. (2022). Available parking places recognition system. *CEUR Workshop Proceedings 4th Workshop for Young Scientists in Computer Science & Software Engineering,* Vol. 3077, 123–134, URL: http://ceur-ws.org/Vol-3077/paper07.pdf. [in English]

9. Marchuk, D. K. (2023). Analiz suchasnykh alhorytmiv vyiavlennia i rozpiznavannia obiektiv z videopotoku dlia system upravlinnia parkuvanniam v realnomu chasi [Analysis of Modern Algorithms for Detecting and Recognizing Objects from a Video Stream for Real-Time Parking Management Systems]. *Visnyk Khmelnytskoho natsionalnoho universytetu. – Herald of Khmelnytskyi National University.* № 3. 339-347. DOI: https://www.doi.org/10.31891/2307-5732-2023-321-3-17-23. [in Ukrainian]

10. Perlin, K. (2002). Improving noise. *ACM Trans.* Graph. 21, 3, 681–682. DOI: https://doi.org/10.1145/566654.566636. [in English]

11. Ryan, J. S., Cowling, P., Alfred Walker, J. (2019). Procedural generation using spatial GANs for region-specific learning of elevation data. *IEEE Conference on Games (CoG).* IEEE. [in English]

12. Fournier, A., Fussell, D., Carpenter, L. (1982). Computer rendering of stochastic models. *Commun.* ACM 25, 6, 371–384. DOI: https://doi.org/10.1145/358523.358553. [in English]

13. Lautakoski, J. (2016). Procedurell generering av terräng Perlin noise eller Diamond-Square : med fokus på exekveringstid och framkomlighet *(Dissertation).* URL: https://urn.kb.se/resolve?urn=urn:nbn:se:his:diva-12854. [in English]

14. Gavin S P Miller. (1986). The definition and rendering of terrain maps. *SIGGRAPH Comput.* Graph. 20, 4, 39–48. DOI: https://doi.org/10.1145/15886.15890. [in English]

15. Adrian, C., Liarokapis, F. (2015). Fractal nature-generating realistic terrains for games. *7th International Conference on Games and Virtual Worlds for Serious Applications (VS-Games).* IEEE. [in English]