УДК 004.4:004.932.2

ОГЛЯД ПЕРСПЕКТИВ ШВИДКОДІЇ АЛГОРИТМІВ ПРОЦЕДУРНОЇ ГЕНЕРАЦІЇ В КОМП'ЮТЕРНИХ ІГРАХ

Іван Лайтарук, Тетяна Гришанович

REVIEW OF THE PROSPECTS OF PROCEDURAL GENERATION ALGORITHMS' PERFORMANCE IN COMPUTER GAMES

Ivan Laitaruk, Tetiana Hryshanovych

Волинський національний університет імені Лесі Українки, просп. Волі, 13, Луцьк, 43025, Україна

***Abstract**. In the modern computer game industry, which is rapidly evolving and demands increased interactivity, realism and scalability of game systems, procedural generation algorithms have become an integral part of development. These algorithms enable the automatic creation of large volumes of content – from landscapes and levels to quests, narratives, and even graphic elements such as textures, materials and 2D/3D models. However, the performance of such algorithms directly depends on their time and space complexity, which becomes a critical factor when generating content in real time or on less powerful devices. Therefore, this analysis considers modern approaches to procedural content generation, particularly algorithms based on noise functions (Perlin noise, fractional Brownian motion), generative grammar (graph rewriting), space partitioning methods (Fortune's algorithm for Voronyi tessellation) and genetic algorithms. In this regard, a detailed review of the prospects for optimization procedural content generation of these algorithms is relevant, which will allow choosing the most suitable solutions for specific game projects and ensuring a balance between content quality, performance, and visual appeal.*

У сучасній індустрії комп'ютерних ігор, яка стрімко розвивається та вимагає все більшої інтерактивності, реалістичності та масштабності ігрових систем, алгоритми процедурної генерації стають невід'ємною частиною розробки. Вони дозволяють автоматизовано створювати великі обсяги контенту — від ландшафтів і рівнів до квестів, сюжетів та навіть графічних елементів, таких як текстур, матеріалів і 2D/3D моделей. Проте ефективність таких алгоритмів напряму залежить від їх часової та просторової складності, що стає критичним чинником при генерації контенту в реальному часі або на слабших пристроях. Задля аналізу розглянуто сучасні підходи до процедурної генерації, зокрема алгоритми на основі шумів (шум Перліна, дробовий броунівський шум), генеративної граматики (переписування графу), розбиття простору (алгоритм Форчуна для теселяція Вороного) та генетичні алгоритми. У зв'язку з цим актуальним є детальний огляд перспектив оптимізації швидкодії цих алгоритмів, що дозволить обрати найбільш придатні рішення для конкретних ігрових проєктів та забезпечити баланс між якістю контенту, продуктивністю й візуальною привабливістю.

Шум Перліна (Perlin noise) [1] є одним із найвідоміших методів процедурної генерації, що використовується для створення текстур, рельєфів, хмар, води та інших складних структур у комп'ютерній графіці та іграх, які виглядають максимально природно. Швидкодія є важливим фактором при його використанні в реальному часі, особливо в ігрових рушіях, таких як Unreal Engine 5, де генерація ландшафтів або ефектів часто виконується під час гри. Алгоритм має часову складність$O(n_p * 2^n)$, де n_p — кількість точок, для яких обчислюється шум, а n — кількість вимірів простору. На практиці під час розробки ігор для забезпечення прийнятної продуктивності часто вдаються до кількох оптимізацій [2]. По-перше, застосовуються попередньо згенеровані шумові карти (noise textures), які зчитуються з пам'яті

замість обчислення шуму в реальному часі. По-друге, використовуються кешування та інтерполяція між уже обчисленими значеннями, що дозволяє значно зменшити кількість операцій. Також можливе обмеження розмірності шуму — замість повноцінного 3D або 4D шуму використовують проєкції або комбіновані карти 2D шуму. Крім того, у деяких випадках розробники переходять на більш швидкі альтернативи, наприклад симплекс-шум, або GPU-реалізації шумових алгоритмів, які дають змогу паралельно обробляти велику кількість точок.

Дробовий броунівський шум (ДБШ) [3] є розвитком ідеї класичного броунівського руху, який описує випадкове переміщення частинки, і широко використовується в процедурній генерації складних природних структур. Швидкодія дробового броунівського шуму безпосередньо залежить від кількості октав, які застосовуються для формування кінцевого результату. Для кожної точки простору необхідно багаторазово обчислювати базовий шум, що в сукупності зі збільшенням частоти кожної наступної октави призводить до суттєвого зростання обчислювального навантаження. Це особливо важливо в задачах генерації в реальному часі або при великих обсягах вхідних точок. У таких рушіях, як Unreal Engine 5, можлива реалізація ДБШ через матеріальні графи або ноди шуму, але навіть там розробникам рекомендується уникати надмірної кількості октав, якщо це не критично для візуальної якості.

Переписування графу [4] — це потужний підхід до процедурної генерації, що дозволяє створювати складні структури на основі чітко заданих правил трансформації. Якщо світ гри представлений у вигляді графу, спочатку формується базовий граф (наприклад, як шаблонна кімната або вулиця), після чого послідовно застосовуються правила, які замінюють підграфи на інші, дотримуючись певної логіки. Найбільш поширеним підходом є алгебраїчне переписування, яке включає методи подвійного та одинарного виштовхування. Метод подвійного виштовхування працює через графи L (вхід), R (вихід) та контекст K, що дозволяє уникнути конфліктів і забезпечує строгість модифікацій. Метод одинарного виштовхування оперує лише графами L і R, що робить його більш гнучким, але потенційно менш стабільним у випадку складних перетворень. Швидкодія переписування графу суттєво залежить від обраної реалізації та розміру графових компонентів. Основною обчислювальною складністю є пошук гомоморфізму — тобто, знайдення у великому графі G підграфу, ізоморфного графу L. Це завдання в загальному випадку є NP-повним, і в найгіршому випадку має експоненційну складність $O\left(n \cdot {V_G}^{V_L}\right)$, де n — кількість правил, а V_G та V_L — кількість вершин у графі G та підграфі L відповідно. У простіших випадках часова складність оцінюється як $O(n \cdot V_G \cdot V_L)$. Хоча застосування самого правила (перетворення підграфу) часто має постійну або лінійну складність, основна затримка виникає саме на етапі пошуку. Практично, для оптимізації переписування графів варто обмежити розмір графів L та G, застосовувати попередню фільтрацію вузлів (наприклад, за мітками або властивостями), а також використовувати кешування результатів частих зіставлень. Окрім цього, доцільно уникати глобальних змін графу й натомість локалізувати правила — це значно пришвидшує обчислення та знижує кількість конфліктів.

Теселяція Вороного [5] є методом розподілення простору, у якому кожна точка області належить до комірки, найближчої до заданої точки розбиття. Такий підхід ідеально підходить для генерації просторово розподілених об'єктів, наприклад, кімнат, островів або біомів у процедурно згенерованих світах. Алгоритм Форчуна [6] є найефективнішим методом побудови діаграми Вороного, що значно оптимізує процес порівняно з наївними геометричними підходами. Замість повного попарного перебору, він використовує пряму «замітання», яка послідовно обробляє події додавання нової точки або зникнення дуги (події кола), створюючи дуги парабол та вершини графу Вороного. Завдяки використанню пріоритетної черги подій і динамічного дерева дуг, алгоритм досягає часової складності $O(nlogn)$ при $O(n)$ пам’яті, де n — кількість точок розбиття. Це робить його ефективним для генерації складних, але детерміновано стабільних структур із великої кількості елементів — особливо коли важлива швидкість генерації на початку гри або при перерахунках під час зміни світу.

Один із підходів до процедурної генерації складних структур ґрунтується на застосуванні еволюційних принципів, зокрема генетичні алгоритми [7]. Швидкодія такого підходу залежить не лише від кількості поколінь та популяції, а й від складності генетичних операцій та обчислення функції допасованості [8]. У спрощеному випадку, коли всі операції мають константну складність, загальна часова складність алгоритму зводиться до $O(GMN)$. Хоча цей алгоритм не завжди є найшвидшим, він забезпечує гнучкість і точність у формуванні складних обмежених структур, що робить його доцільним вибором при розробці генераторів з високим рівнем варіативності та контрольованості. Оптимізація генетичних алгоритмів у рушії передбачає зменшення витрат на обчислення: використання кешування для повторюваних обчислень функції допасованості, паралелізацію обробки популяцій на CPU або GPU, а також обмеження складності операцій схрещення та мутацій (найкраще звести їх до константного часу). Такі покращення дозволяють застосовувати генетичні алгоритми не лише офлайн на етапі розробки, а й у реальному часі — наприклад, для адаптивної генерації контенту в rogue-like іграх або симуляторах.

Підсумовуючи, можна сказати, що швидкодія методів процедурної генерації в комп’ютерних іграх значною мірою залежить від обраного підходу. Оптимізація кожного з підходів має враховувати його особливості: використання кешування, паралелізації, обмеження ітерацій або складності функцій дозволяє досягти прийнятного балансу між якістю згенерованого контенту та продуктивністю. Ефективна реалізація цих методів у рушіях потребує врахування контексту їх застосування — чи це одноразова генерація на старті рівня, чи безперервне створення контенту під час гри. Важливо не перевантажувати алгоритм зайвою складністю — кожен його компонент має бути логічно обґрунтованим, послідовним у виконанні та обов’язково перевіреним на практиці за допомогою тестування, щоб уникнути неочікуваної поведінки під час генерації ігрового контенту.

Перспективи пришвидшення алгоритмів процедурної генерації полягають у використанні апаратних і програмних удосконалень. Одним із ключових напрямів є паралелізація обчислень, зокрема багатопотокове виконання ітерацій клітинних автоматів або обрахунку функцій допасованості в генетичних алгоритмах на CPU та GPU. Значного скорочення часу можна досягти шляхом застосування попередньо обчислених шаблонів, які комбінуються у нові конфігурації замість повної генерації з нуля. Інкрементна генерація лише необхідних у поточний момент частин світу також дає змогу зменшити ресурсне навантаження. Окрім цього, потенціал має інтеграція машинного навчання, коли нейронні мережі швидко приймають рішення на основі попереднього досвіду замість складних евристик. Використання ефективних структур даних, таких як BSP-дерева або Octree [9], може оптимізувати доступ до ігрових об'єктів та їх розміщення в просторі. Поєднання цих підходів забезпечить підвищення продуктивності генерації та зробить її більш гнучкою і адаптивною до умов ігрового середовища. На таблиці 1 наведені узагальнені дані для розглянутих алгоритмів процедурної генерації.

Таблиця 1. Узагальнююча характеристика алгоритмів процедурної генерації

Метод процедурної генерації світу	Можливі оптимізації / перспективи	Оцінка часової складності
Переписування графу	Обмеження розміру графів L і G, локалізація правил, кешування гомоморфних зіставлень, автоматизоване створення правил (наприклад, методом Меррелла [10])	$O(n * V_G * V_L)$ в середньому, $O(n * {V_G}^{V_L})$ в найгіршому: n правил переписування, V_G – кількість вершин G, V_L – кількість вершин L

Продовження таблиці 1

Алгоритм Форчуна для теселяції Вороного	Паралелізація генерації, зменшення кількості точок, використання простіших метрик відстані, попередній фільтр точок, GPU-обчислення	Перебір усіх точок $O(n^2)$, Алгоритм Форчуна $O(nlog(n))$: n – кількість точок діаграми Вороного
Шум Перліна	Зменшення розмірності, обмеження кількості обчислюваних точок, використання попередньо згенерованих карт, кешування результатів, GPU-обчислення	$O(n_p * 2^n)$: n_p – кількість точок, n – кількість вимірів
Генетичний алгоритм	Паралельна обробка популяцій, спрощення функції допасованості, зменшення розміру популяції або кількості поколінь, кешування, GPU-обчислення	$O(GMN)$: G – кількість поколінь, M – кількість генів індивіда, N – розмір популяцій

Бібліографія

1. MacWha R. Generating Digital Worlds Using Perlin Noise. *Medium*. URL: https://medium.com/nerd-for-tech/generating-digital-worlds-using-perlin-noise-5d11237c29e9.
2. Brucks R. Getting the Most Out of Noise in UE4. *Unreal Engine*. URL: https://www.unrealengine.com/en-US/tech-blog/getting-the-most-out-of-noise-in-ue4.
3. Dalefield E. Distributed Architecture for Procedural Terrain Generation in Video Games : Master's Thesis. Wellington, 2024. 75 с. URL: https://openaccess.wgtn.ac.nz/articles/thesis/Distributed_Architecture_for_Procedural_Terrain_Generation_in_Video_Games/25658514?file=45770220.
4. McCollum J. An introduction to graph rewriting for procedural content generation. *The Shaggy Dev*. URL: https://shaggydev.com/2022/11/20/graph-rewriting/.
5. Mount D. CMSC 754: Lecture 11 Voronoi Diagrams and Fortune's Algorithm. URL: https://www.cs.umd.edu/class/spring2020/cmsc754/Lects/lect11-vor.pdf.
6. afuzzyllama. VoronoiDiagramUE4: Voronoi Diagram with Fortune's Algorithm implementation. *GitHub*. URL: https://github.com/afuzzyllama/VoronoiDiagramUE4.
7. Kraner V., Fister I., Brezočnik L. Procedural Content Generation of Custom Tower Defense Game Using Genetic Algorithms. *SpringerLink*. URL: https://doi.org/10.1007/978-3-030-75275-0_54.
8. Пиріг Я. Оцінка обчислювальної складності генетичного алгоритму. Інфокомунікаційні технології та електронна інженерія. 2024. Вип. 4, № 1. С. 52–60. URL: https://doi.org/10.23939/ictee2024.01.052.
9. *Shimer C. BSP trees in 3D worlds. Computer Science | Worcester Polytechnic Institute | CS563, Advanced Topics in Computer Graphics. URL: http://web.cs.wpi.edu/~matt/courses/cs563/talks/bsp/bsp.html.*
10. Merrell P. Example-Based Procedural Modeling Using Graph Grammars. *ACM Transactions on Graphics*. 2023. Т. 42, № 4. С. 1–16. URL: https://doi.org/10.1145/3592119.