

계층적 임의 보행 기반 대규모 타일맵 생성 알고리즘과 Lock-free 병렬·IPC 구현

유현수, 고정국
동명대학교 컴퓨터공학과
e-mail : lioie6478@gmail.com

A Hierarchical Random-Walk Algorithm for Large-Scale Tilemap Generation and Its Lock-Free Parallel, IPC-Based Implementation

Hyun-su Yu, Jeong-guk Koh
Dept. of Computer Engineering, Tongmyong University

요 약

본 논문은 맵을 Abstract-Room-Tile 계층으로 나누어 확률적 임의 보행과 flood-fill로 주 통로를, 수직 관통로(gate)로 층간 연결을 구성하는 대규모 타일맵 생성 알고리즘을 제안한다(보조 방은 Spelunky 식 탐색 요소로 의도적 산재). 이를 수억 개 타일 규모에서 게임 엔진 정지 없이 실행하기 위해, 생성을 Unity와 분리된 독립 에이전트 프로세스로 옮기고 실행 인자·Named Pipe·공유 PNG의 세 IPC 채널로 연결하며, 겹치지 않는 영역 분할과 스레드 지역 난수기로 Lock 없이 병렬화한다. 약 2.7억 타일(3000×3000×30) 실험에서 제안 구현은 메인 스레드 직접 생성의 약 232초 프레임 동결·약 20GiB 할당을 피하고 생성 중에도 호스트 실패 프레임률을 안정적으로 유지하여, 제안 알고리즘이 대규모로 실현 가능함을 보였다.

1. 서론

절차적 콘텐츠 생성은 방대한 게임 월드를 자동으로 확보할 수 있게 해 주지만, 타일맵 규모가 커지면 연산량이 급증한다. 한 번이 수천 타일인 계층이 수십 개인 3차원 타일맵은 타일 수가 수억 개에 달하며, 이를 게임 엔진 메인 스레드에서 생성하면 완료까지 화면이 멈추고 대량의 임시 객체가 가비지 컬렉션(GC) 부하를 유발한다.

이를 완화하려면 생성을 엔진과 분리된 외부 프로세스로 옮겨 런타임·GC 제약과 장애 전파에서 자유롭게 하되, 두 프로세스를 잇는 프로세스 간 통신(IPC, Inter-Process Communication)이 필요하다. 또한, 생성 연산 자체도 병렬화해야 하는데, 여러 스레드가 하나의 맵 자료구조를 동시에 수정하면 Lock이 경쟁과 대기를 유발하므로 Lock에 의존하지 않는 병렬화가 요구된다.

본 논문의 기여는 다음과 같다. 첫째, 맵을 Abstract-Room-Tile 계층으로 나누고 확률적 임의 보행과 flood-fill·BFS로 층 내부 통로와 층간 관통로를 구성하는 대규모 타일맵 생성 알고리즘을 제안한다. 둘째, 이 알고리즘을 맵의 겹치지 않는 영역 분할과 작업별 스레드 지

역 난수기로 Lock 없이 병렬화한다. 셋째, 이를 Unity 호스트와 분리된 에이전트 프로세스에 세 IPC 채널(실행 인자·Named Pipe·공유 파일)로 연결해 약 2.7억 타일(3000×3000×30) 규모에서 호스트 정지 없이 실현하고, 메인 스레드 직접 생성 및 엔진 내부 Job System과의 비교로 그 실행 특성과 절충을 실측으로 제시한다.

2. 관련연구

절차적 맵 생성에는 다양한 접근이 연구되어 왔다. 프랙탈·노이즈 기반 기법은 구현이 간단하지만 시드 외에 형태를 직접 제어하기 어렵고, 셀룰러 오토마타는 동굴형 레벨을 실시간으로 생성할 수 있으나 생성 규칙에 따라 결과가 크게 달라진다[4]. 격자 기반 던전 생성은 방과 통로를 floor/wall 타일로 배치하는 방식으로 폭넓게 쓰이며[3], 관련 연구들은 생성 속도보다 결과에 대한 풍부한 제어가 더 중요한 과제임을 지적한다[2]. 본 논문이 제안하는 생성 알고리즘은 이러한 기존 기법과 달리, 맵을 Abstract→Room→Tile의 계층으로 분할하고 확률적 임의 보행과 flood-fill·BFS로 층 내부 통로와 층간 관통로를 직접 구성하도록 설계하였으며, 그 맵 형태는 Spelunky 2[1]의 계층형 구조에서 영감을 받았다.

한편 무거운 생성 연산을 게임 엔진과 분리하는 접근에서는 두 프로세스 사이의 통신이 필요하다. 프로세스 간 통신(IPC)은 파이프·공유 메모리 등으로 구현되며, 작은 제어 메시지에는 파이프가, 수억 개 타일과 같은 대량 데이터 전송에는 파일·공유 메모리가 유리한 것으로 알려져 있다[5]. 또한, 다중 스레드 환경에서 Lock에 의존하지 않는 비차단 병렬화는 경쟁과 대기 줄이는 방법으로 연구되어 왔다[8]. 본 논문은 공유 자원 자체를 제거하는 데이터 분할과, 작업마다 상태를 공유하지 않는 xorshift 기반 스레드 지역 난수기[6]를 결합하고, .NET ThreadPool[7]로 작업을 배치 실행함으로써 Lock 없이 병렬 생성하도록 설계한다.

3. 연구 방법

제안하는 프레임워크는 게임 엔진인 Unity 호스트와, 맵 생성만 전담하는 독립 실행 파일인 에이전트의 두 프로세스로 구성된다(그림 1). 에이전트는 Unity의 StreamingAssets에 동봉되어, 맵이 필요할 때 Unity가 자식 프로세스로 실행한다. 무거운 생성을 별도 프로세스로 분리하므로 메인 스레드 정지와 GC 부하를 피하고, 에이전트가 비정상 종료해도 호스트가 감지해 재실행할 수 있다.



그림 1. 프레임워크 구조 및 IPC 채널 개요

두 프로세스는 세 채널로 데이터를 나누어 주고받는다. 입력 파라미터는 JSON 직렬화 후 Base64로 인코딩해 실행 인자로 넘기고, 진행 상황은 Named Pipe로 스트리밍하여 에이전트가 레이어 생성·저장 완료마다 메시지를 보내고 마지막 AgentFin으로 완료를 알린다. 완성 맵은 직렬화 시 수백 MB에 달해 레이어별 PNG 파일로 공유한다. 호스트는 필요한 레이어만 능동적으로 읽고, 타임아웃 내 완료 신호가 없으면 프로세스를 정리·재실행해 통신 실패에 대응한다.

맵은 Abstract→Room→Tile의 계층으로 정의한다. Abstract는 Room을 한 칸으로 보는 격자이고 각 칸이 Single Room Size의 Tile 격자(Room)에 대응하여, 최종 해상도는 Map Size × Single Chunk Size × Single Room Size로 결정된다. 이 계층은 병렬화의 분할 단위가 된다. 생성은 통로 채움 비율, 브러시 크기, 관통로 개수·부피 비율, 경로 탐색 재시도 상한, 난수 시드 등으로 제어한다.

한 층의 통로 생성은 세 단계다(그림 2). ① Abstract: 하단 중앙에서 상단까지 도착 거리를 가중치로 삼는 확률적 임의 보행으로 방 사이 경로를 그린다. 보행은 아직 비어 있는 칸으로만 이동하므로 한 번 채워진 방은 다른 보행이 밟지 못한다. flood-fill로 연결 영역을 확인해 좌표쌍을 저장함으로써 시작에서 목표까지 이어지는 주 통로의 연결성을 확보한다. 이어 주 통로에 인접한 빈 칸에 보조 방을 덧붙이는데, 이 보조 방들은 주 통로와 반드시 이어지는 않으며 Spelunky 식 맵처럼 탐색 요소로 의도적으로 산재시킨 것이다. ② Room: 좌표쌍을 따라 각 Room 내부에서 진입·진출 변까지 같은 임의 보행으로 방향표를 만든다. ③ 브러시: 그 경로를 따라 랜덤 크기 정사각 브러시로 통과 가능 타일을 두께감 있게 채운다.

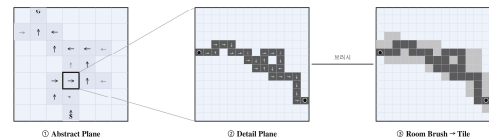


그림 2. 한 층 내부 통로 생성

층 내부 통로가 놓이면 층을 잇는 관통로를 만든다(그림 3). 인접 두 층에서 양쪽 모두 통과 가능한 (x, z) 칸을 후보로 모아 하나를 선점하고, 상하좌우 BFS로 수직 연결 영역(gate)을 확장하며, 부피 상한 안에서 층당 정해진 횟수만큼 반복한다.

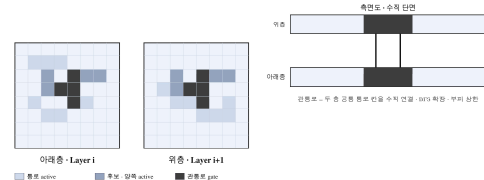


그림 3. 인접한 두 층의 수직 gate

생성·저장은 ① 층 단위 생성, ② 좌표쌍 단위 Room 계산, ③ 방 블록 단위 관통로, ④ 레이어 PNG 저장의 네 지점에서 병렬화한다. 작업을 ThreadPool[7]에 넣어 최대 N개씩 배치 실행하고 ManualResetEvent/WaitHandle 배리어로 완료를 기다린다(그림 4).

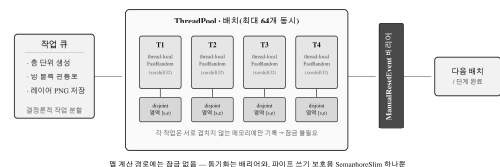


그림 4. Lock-Free 병렬 파이프라인

Lock 없이 병렬화할 수 있는 근거는 두 가지다. 첫째, 데이터 분할로, 각 층은 독립 배열을 쓰고 상세 통로 계산도 앞서 설명했듯 추상 셀이 좌표쌍별로 분할되어 각 방을 한 작업만 기록하며, 관통로는 겹치지 않는 블록에 기록한다. 둘째, 스레드 지역 난수기로, 각 작업이 공용 난수기 대신 좌표 기반 시드의 지역 xorshift[6] 인스턴스를 쓴다. 따라서 공유 쓰기가 구조적으로 없어 계산 경로에 Lock이 불필요하며, Lock은 배치 배리어와 파이프 쓰기 보호 세마포어에만 쓰인다. 각 작업이 서로 다른 영역에 결정론적 시드로 기록하므로 스케줄 순서와 무관하게 동일 결과가 생성되도록 설계하였다.

각 타일 상태는 Unity 텍스처용으로 정규화된 부동소수점 값으로 변환되어 R8G8B8A8 PNG에 저장된다. 8비트 한 채널에 담으면 소수점 약 3자리 정밀도만 확보되어 온전한 전달이 어려우므로, 값을 상위·하위로 나눠 R·G 두 채널에 분배하는 2채널 멀티비트 인코딩을 사용하고 읽을 때 합산해 복원한다. 이 방식은 항후 지형 높이·확률값 등 소수 단위 속성 확장에도 유효하다.

4. 연구 결과

연구 결과는 제안 알고리즘의 구조적 품질(표 3)을 핵심 근거로, 그 알고리즘을 대규모로 실현하는 비용(표 1·2)을 보조 근거로 제시한다. 실험은 Intel Core i9-13900HX, RAM 약 32GB, RTX 4070 Laptop, Unity 6000.3.6f1 Mono 호스트에서 수행하였다. 3000×3000×30(약 2.7억 타일)의 동일한 맵 생성을 (A) Unity 내부 Job System + Burst, (B) 메인 스레드 직접 생성, (제안) 외부 에이전트의 세 방식으로 각각 위밍업 2회 후 10회 측정하였다. 생성 시간은 계산 구간만을 가리키며 PNG 저장·프로세스 시작·IPC 준비는 제외하였으며, 발열은 통제하지 않았다.

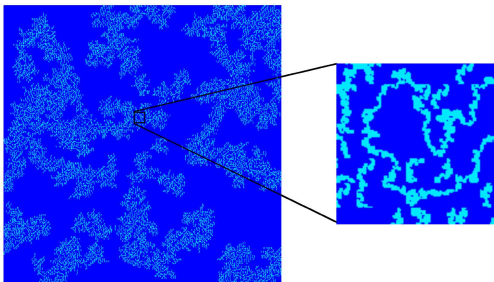


그림 5. 생성 결과 예시(3000×3000 한 층)

세 실행 방식의 측정 결과는 표 1과 같다.

표 1. 실행 방식별 생성·준비 시간과 호스트 영향(2.7억 타일, 본 10회)

지표	A: Job System+Burst	B: Main Thread	제안: IPC+Agent
생성 평균±SD(초)	3.565 ± 0.126	239.358 ± 11.657	22.467 ± 3.291
Total Turnaround 평균±SD(초)	6.559 ± 0.128	242.192 ± 11.701	37.206 ± 3.374
실효 FPS 평균	29.31	0.257	51.26
전체 최악 프레임(ms)	2750.3	252161.8	230.3
호스트 누적 관리형 할당	10.5 KB	약 21.9 GB	118 KB

여기서 Total Turnaround는 생성 요청부터 사전 지정한 20×20 플레이 영역(활성 타일 165개)의 렌더·충돌·격자 인접 준비 완료까지이며, 전체 2.7억 타일의 게임 설치 완료가 아니다. 생성 계산은 Job System + Burst(A)가 평균 3.565초로 가장 빠르나, A조차 최악 프레임이 약 2.75초에 달하고 메인 스레드(B)는 약 239초 동결·약 21.9GiB 할당으로 생성 중 호스트가 멈춘다. 반면, 제안 방식은 생성이 별도 프로세스에서 수행되어 생성 시간은 22.467초로 더 길어도 생성 중 호스트 실효 프레임률이 51.3FPS로 세 방식 중 가장 높고 최악 프레임도 약 230ms로 가장 낮았다. 영역 준비 기준 A 대비 제안은 약 5.67배로, 저장·IPC·반입·디코딩을 포함해도 격차가 크게 벌어지지 않는다. 즉, 제안 방식의 강점은 속도가 아니라 생성 중 호스트 반응성이며, 호스트 누적 할당 118KB가 보이듯 대량 관리형 할당은 제거가 아니라 별도 프로세스로 분리된 것이다. 단, 시간·프레임 지표는 2차 보완 단일 런, 누적 관리형 할당은 선행 계측 런 기준이다.

저장 포맷별 용량과 저장·복원 비용은 표 2와 같다.

표 2. 저장 방식별 용량·I/O 비용(2.7억 타일, 본 10회)

방식	용량(MB)	저장(s)	복원(s)
2채널 PNG(제안)	14.6	2.142	0.758
Raw 파일 I/O(무압축)	270	0.123	0.021
Raw의 MMF 접근(copy)	270	0.091	0.053
Raw의 MMF 접근(direct)	270	0.104	0.200

제안 방식의 2채널 PNG는 30장 합쳐 14.603MB로 타일 원시 자료(270MB)의 약 5.41%에 그쳐, 완성 맵을 공유 파일로 전달할 때 용량 면에서 유리하다. 다만 저장

2.142초·복원 0.758초로 무압축 Raw(0.123초·0.021초)보다 느려, 용량을 I/O 시간으로 치르는 절충을 보인다. 대량 맵 전달의 또 다른 선택지로, 무압축 데이터를 메모리 맵 파일(MMF)로 주고받는 방식도 함께 측정하였다. MMF는 매핑 후 배열로 복사하는 copy와, 매핑을 유지한 채 접근하는 direct 두 경로다. 세 방식은 각각 압축 이미지, 무압축 파일, 공유 메모리라는 서로 다른 전달 전략으로 용량과 속도의 절충이 다르며, 전 본 시행에서 복원 맵의 SHA-256이 입력과 일치했다.

생성 맵의 구조적 품질은 표 3과 같다.

표 3. 생성 맵 구조 품질

지표	값
활성 타일 비율(%)	12.70(12.53~12.98)
격자상 막다른 끝 타일 비율(%)	0.28(0.27~0.29)
시드 간 형태 다양성(Jaccard 거리)	0.926(0.917~0.938)
3D 최대 연결 성분 / 전체 활성(%)	63.5

생성 맵은 한 층만 보면 주 통로와 산재한 보조 방으로 나뉘어 2차원 평면상 다수의 조각으로 나타난다. 보조 방은 Spelunky 식 탐색 요소로 의도적으로 분리한 것이며, 층을 잇는 수직 관통로(gate)가 이 조각들을 3차원에서 통합한다. 약 2.7억 타일 규모에서 전체 통행 가능 타일의 63.5%가 하나의 3차원 연결 성분을 이루고, 최하층에서 최상층까지 격자 인접으로 도달 가능하다(그림 6). 막다른 끝 타일 비율은 약 0.28%로 낮고, 서로 다른 시드 간 형태 다양성(Jaccard 거리 0.926)은 시드에 따라 형태가 뚜렷이 달라짐을 보인다.

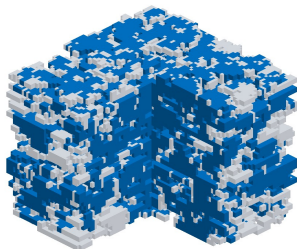


그림 6. 3D 연결성

제안 방식의 ThreadPool 배치 상한 N을 4~64로 바꾼 결과, N=16~32에서 가장 짧았고(최저 13.9초, N=4 대비 2.45배) 이후로는 둔화 및 비단조였다. 실험 CPU(i9-13900HX, 24코어·32논리 스레드)의 가용 스레드 수에 도달하면 병렬도가 포화되며, N=64에서는 스레

드 파구독과 P/E 코어 이질성으로 오히려 느려진다. 실제로 N을 늘려도 벽시계 시간은 정체되나 누적 CPU 시간은 증가하여(N=4→32 약 84%↑), 추가 자원이 순수 생성보다 스케줄링·동기화·캐시 접근 비용에 소모됨을 보인다.

5. 결론

본 논문은 맵을 Abstract-Room-Tile 계층으로 나누어 임의 보행과 flood-fill·BFS로 통로와 층간 관통로를 구성하는 대규모 타일맵 생성 알고리즘을 제안하고, 겹치지 않는 영역 분할과 스레드 지역 난수기로 Lock 없이 병렬화하였다. 이를 게임 엔진 정지 없이 실현하고자 생성을 Unity와 분리된 에이전트 프로세스로 옮기고 세 IPC 채널로 연결하였으며, 2.7억 타일 실험에서 메인 스레드 직접 생성의 약 232초 동결·약 20GiB 할당을 피하고 생성 중에도 호스트 실패 프레임률을 안정적으로 유지하였다. 즉 프레임워크의 강점은 생성 속도가 아니라 제안 알고리즘을 호스트 반응성을 유지한 채 대규모로 실행하는 데 있으며, 향후 Job System 프레임 스파이크 프로파일링, 하드웨어 수준 병목 분석, Release/IL2CPP·실제 게임 일반화를 과제로 남긴다.

참고문헌

- [1] Mossmouth, Spelunky 2, Mossmouth LLC, 2020.
- [2] R. van der Linden, R. Lopes, and R. Bidarra, "Procedural Generation of Dungeons," IEEE Transactions on Computational Intelligence and AI in Games, Vol. 6, No. 1, pp. 78-89, 2014.
- [3] A. Hilliard, C. Salis, and H. Elaaraag, "Algorithms for Procedural Dungeon Generation," Journal of Computing Sciences in Colleges, Vol. 33, No. 1, pp. 166-174, 2017.
- [4] L. Johnson, G. N. Yannakakis, and J. Togelius, "Cellular Automata for Real-time Generation of Infinite Cave Levels," Proc. of the 2010 Workshop on Procedural Content Generation in Games (PCGames '10), Article No. 10, 2010.
- [5] W. R. Stevens and S. A. Rago, Advanced Programming in the UNIX Environment, 3rd ed., Addison-Wesley, 2013.
- [6] G. Marsaglia, "Xorshift RNGs," Journal of Statistical Software, Vol. 8, No. 14, pp. 1-6, 2003.
- [7] D. Leijen, W. Schulte, and S. Burckhardt, "The Design of a Task Parallel Library," Proc. of the ACM SIGPLAN Conf. on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), pp. 227-242, 2009.
- [8] M. Herlihy and N. Shavit, The Art of Multiprocessor Programming, Morgan Kaufmann, 2008.