



(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2025-0063784
(43) 공개일자 2025년05월08일

- | | |
|--|---|
| <p>(51) 국제특허분류(Int. Cl.)
 <i>G06F 30/27</i> (2020.01) <i>G06F 111/10</i> (2020.01)
 <i>G06F 123/02</i> (2023.01) <i>G06F 9/38</i> (2024.01)
 <i>G06N 3/092</i> (2023.01) <i>G06T 1/20</i> (2018.01)</p> <p>(52) CPC특허분류
 <i>G06F 30/27</i> (2020.01)
 <i>G06F 9/3885</i> (2013.01)</p> <p>(21) 출원번호 10-2025-0051279(분할)
 (22) 출원일자 2025년04월21일
 심사청구일자 2025년04월21일</p> <p>(62) 원출원 특허 10-2025-0038962
 원출원일자 2025년03월26일
 심사청구일자 2025년03월26일</p> | <p>(71) 출원인
 강성운
 부산광역시 해운대구</p> <p>(72) 발명자
 강성운
 부산광역시 해운대구</p> |
|--|---|

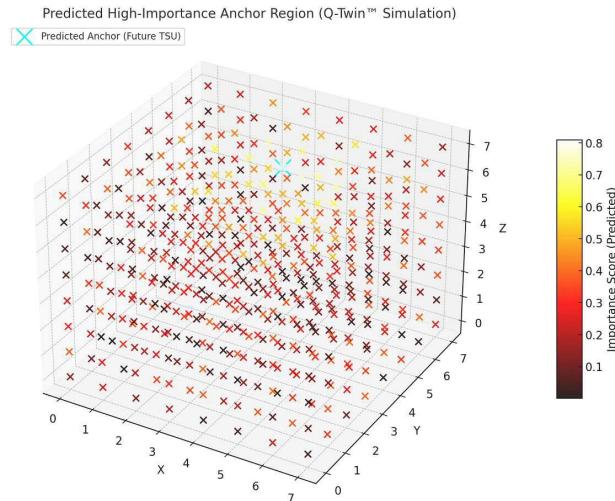
전체 청구항 수 : 총 10 항

(54) 발명의 명칭 Q-Twin™: Q-Block 기반 디지털 트윈 생성 및 시계열 시뮬레이션 시스템

(57) 요약

본 발명은 Q-Block 기반의 디지털 트윈 생성 및 시뮬레이션 시스템(Q-Twin™)에 관한 것으로, 각 단위큐브(Unit Cube)의 메타데이터(정보 밀도, 에너지, 중첩 확률 등)를 기반으로 연산 중요도를 산출하고, 중심 단위큐브(anchor cube)를 기점으로 병렬 연산 및 시계열 예측을 수행한다. 특히 강화학습 기반 AI가 시공간 흐름을 학습하여 미래 TSU(Time Singularity Unit)에서의 연산 집중 영역을 예측하고, 이를 시각화 및 시뮬레이션 인터페이스를 통해 실시간으로 표현함으로써, 고도화된 예측형 디지털 트윈 시스템을 구현한다.

대표도 - 도16



(52) CPC특허분류

G06N 3/092 (2023.01)

G06T 1/20 (2013.01)

G06F 2111/10 (2020.01)

G06F 2123/02 (2023.01)

명세서

청구범위

청구항 1

Q-Twin™: Q-Block 기반 디지털 트윈 생성 및 시계열 시뮬레이션 시스템에 있어서,
다수의 단위큐브(unit cube)로 구성된 3차원 Q-Block과,
상기 Q-Block 내 좌표상 중심 위치((4,4,4))에 위치하는 중심 단위큐브(anchor cube)를 기준으로,
각 단위큐브의 메타데이터를 수집하고,
수집된 메타데이터에 사전 설정된 가중치를 적용하여 중요도 점수를 계산하며,
상기 중요도 점수를 기준으로 단위큐브를 정렬하고,
정렬된 단위큐브를 연산 우선순위 큐로 구성한 후,
상기 연산 우선순위 큐를 CUDA 기반 GPU 환경에서 병렬로 연산 수행하고,
그 연산 결과를 기반으로 시계열 시뮬레이션 또는 강화학습 기반 연산 정책(policy)을 적용하는 중첩 정보 기반
디지털 트윈 생성 및 시뮬레이션 시스템.

청구항 2

제1항에 있어서,
상기 중요도 점수는 다음 수식을 기반으로 계산되는 것을 특징으로 하는 시스템;
$$\text{importance}_i = w_d \cdot \text{density}_i + w_e \cdot \text{energy}_i + w_o \cdot \text{overlap}_i + w_t \cdot \text{tsu}_i + w_v \cdot \text{vector}_i$$

(단, w_d , w_e , w_o , w_t , w_v 는 사전에 설정된 가중치 값이다.)

청구항 3

제1항에 있어서,
상기 연산 우선순위 큐는 중심 단위큐브의 메타데이터 상태가 미리 정의된 트리거 조건을 만족할 경우,
anchor-to-anchor 기반 연산 파장이 외곽 큐브로 확산되도록 구성되며,
CUDA 환경 내에서 동기화된 병렬 연산 흐름을 실시간으로 제어하는 것을 특징으로 하는 시스템.

청구항 4

제1항에 있어서,
상기 중심 단위큐브의 메타데이터는 강화학습 기반 정책 네트워크(policy network)의 초기 입력값으로
활용되며,
다중 Q-Block 간 시계열 흐름을 학습하여 경로 예측 또는 시뮬레이션 흐름을 최적화하는 것을 특징으로 하는 시
스템.

청구항 5

제1항에 있어서,
각 Q-Block의 중심 단위큐브로부터 추출된 중요 메타데이터는
후속 Q-Block의 연산 초기값(seed)으로 전달되어 anchor-to-anchor 연산 흐름을 시계열적으로 연계하는 것을 특
징으로 하는 시스템.

청구항 6

Q-Twin™: Q-Block 기반 디지털 트윈을 생성하고 시계열 시뮬레이션을 수행하는 방법에 있어서,
 (가) 단위큐브들로 구성된 Q-Block 내 중심 단위큐브를 기준으로 반경 내 단위큐브들을 선별하고,
 (나) 각 단위큐브로부터 메타데이터를 수집하며,
 (다) 메타데이터에 사전 정의된 가중치를 적용하여 중요도 점수를 산출하고,
 (라) 산출된 중요도 점수를 기준으로 단위큐브들을 정렬하며,
 (마) 정렬된 큐브 정보를 기반으로 CUDA GPU 환경에서 병렬 연산을 수행하거나,
 (바) 강화학습 기반 연산 정책(policy)의 초기 입력값으로 제공하는
 중첩 정보 기반 디지털 트윈 운용 방법.

청구항 7

제6항에 있어서,
 중심 단위큐브 간 anchor-to-anchor 기반 시계열 연산 흐름이
 강화학습 프레임워크의 상태(state), 행동(action), 보상(reward) 기반 구조로 연계되어,
 AI 연산 최적화 또는 경로 탐색에 활용되는 것을 특징으로 하는 방법.

청구항 8

제6항에 있어서,
 상기 정렬은 GPU 디바이스 상에서 Thrust 라이브러리를 활용하여 직접 수행되며,
 CPU 메모리로의 복사 없이 GPU 상에서 병렬 정렬 및 연산 스케줄링이 실시간으로 진행되는 것을 특징으로 하는 방법.

청구항 9

제6항에 있어서,
 연산 흐름 또는 경로 탐색 초기화 시,
 이전 Q-Block의 중심 단위큐브로부터 전달된 시드 벡터를 기반으로
 후속 Q-Block의 연산 방향성 또는 탐색 정책을 초기화하는 것을 특징으로 하는 방법.

청구항 10

제6항에 있어서,
 정렬된 단위큐브의 중요도 값 또는 연산 상태가
 WebGL, Unity, 또는 Unreal 기반의 시뮬레이션 인터페이스에 연동되어
 사용자 인터랙션 기반의 실시간 시각적 분석 및 피드백 기반 파라미터 조정이 수행되는 것을 특징으로 하는 방법.

발명의 설명

기술 분야

[0001] 본 발명은 시공간 기반의 디지털 트윈 생성 및 시뮬레이션 기술에 관한 것으로, 보다 구체적으로는 시간 특이점 (Time Singularity)을 기준으로 일정 시간 단위(TSU: Time Singularity Unit)마다 생성되는 디지털 시공간 단위인 단위큐브(Unit Cube)를 3차원 격자 형태로 적층하여 구성한 Q-Block을 기반으로, 현실 공간의 센서 데이터를 단위큐브에 실시간으로 매핑하고, 이를 누적 및 시뮬레이션 가능한 시계열 디지털 트윈 구조로 구현하는 시

시스템에 관한 것이다.

[0002] 여기서 Q-Block은 $8 \times 8 \times 8$ 구조로 적층된 총 512개의 단위큐브로 구성되며, 이는 GPU 기반 병렬 연산 환경에서 메모리 정렬, 쓰레드 병렬화, 동기화 효율 측면에서 최적의 연산 단위로 설정된 것이다. 8이라는 수치는 2의 거듭제곱 구조로서 CUDA 기반의 병렬 컴퓨팅 아키텍처에 이상적으로 대응되며, 향후 연산 성능의 발전에 따라 $16 \times 16 \times 16$ (4,096개) 등으로 유연하게 확장 가능한 구조적 장점을 지닌다.

[0003] Q-Block은 각 단위큐브의 고유 식별자, 생성 시점(TSU), 위치 좌표, 센서 기반 상태정보 등 메타데이터를 포함하고, 이러한 Q-Block이 시간의 흐름에 따라 적층되거나 병렬 배치되어 시공간 구조를 형성한다. 본 발명은 이러한 Q-Block을 현실 공간의 정보 흐름에 따라 실시간으로 갱신하며, 시뮬레이션 입력 또는 예측 연산에 따라 Q-Block 기반의 디지털 트윈 구조를 동적으로 구성·확장하는 시스템에 관한 것이다.

[0004] 본 발명은 2022년 2월 8일 최초 출원된 단위큐브 기반 기술사상을 바탕으로 도출된 Q-Block 구조를 기반으로 하며, 해당 Q-Block에 현실 데이터를 구조화된 시공간 단위로 매핑하고 시계열적으로 적층함으로써 디지털 트윈의 실시간성, 누적성 및 시뮬레이션 연계 기능을 구현하는 기술적 독립성과 응용 확장성을 갖춘 분할 발명이다.

배경 기술

[0005] 기존의 디지털 트윈 기술은 주로 현실 공간의 위치 좌표 정보를 가상 공간에 대응시키는 방식으로 구현되며, 정적인 3D 모델이나 센서 기반 모니터링 시스템에 집중되어 왔다. 이러한 시스템은 개별 위치의 상태를 시각화하거나 실시간 모니터링하는 데 적합하지만, 시간 흐름에 따라 누적되는 시공간 변화를 구조적으로 표현하거나, 센서 정보의 시계열 데이터가 축적된 연산 구조로 발전시키는 데에는 한계가 존재하였다. 예를 들어, 기존의 스마트 시티 모델이나 BIM 기반 시뮬레이션 시스템은 고정된 건축 구조 또는 설비 상태를 반영하는 데에는 유용하나, 지속적으로 변화하는 센서 정보(예: 기온, 밀도, 진동, 유동량 등)를 시간 단위로 구조화하여 적층적 디지털 트윈으로 구성하고, 이를 통해 동적 시뮬레이션을 수행하는 데에는 구조적 제약이 많았다. 또한 이러한 시스템은 대부분 CPU 기반 직렬 연산 구조로 구성되어 있어, 수많은 센서 노드로부터의 데이터 흐름을 실시간으로 병렬 분석하거나, 시계열 흐름을 공간적으로 재현하는 데에는 연산 병목이 발생한다.

[0006] 한편, 기존의 GADGET-2와 같은 시뮬레이션 도구는 천체 물리학적 연산에 특화되어 있으나, 정보 단위의 구조적 시공간 표현 및 현실 세계 데이터와의 실시간 대응 구조를 제공하지 않으며, 특히 경과 시간에 따라 데이터가 누적되는 트윈 적층 구조로서의 활용에는 부적합하다. 이에 따라, 본 발명은 시간 특이점(Time Singularity)을 기준으로 일정 시간 단위(TSU)마다 생성되는 단위큐브(Unit Cube)를 $8 \times 8 \times 8$ 로 적층한 Q-Block을 병렬 연산 단위로 삼고, 현실 공간의 센서 데이터를 이 Q-Block 단위의 메타데이터로 매핑함으로써, 현실 세계의 상태 변화가 시계열적으로 누적·적층된 디지털 트윈으로 변환되는 새로운 구조를 제안한다. Q-Block 간의 적층 순서, 메타데이터 유사도, 중첩 확률 등의 시공간 특성을 고려하여, 단순 모니터링을 넘어 예측, 변화 분석, 시뮬레이션 수행이 가능한 고차원 트윈 연산 시스템이 요구된다.

선행기술문헌

특허문헌

- [0007] (특허문헌 0001) KR1020220016546
(특허문헌 0002) KR1020240174073
(특허문헌 0003) KR1020250038962

비특허문헌

- [0008] (비특허문헌 0001) Springel, V. (2005). The cosmological simulation code GADGET-2. Monthly Notices of the Royal Astronomical Society, 364(4), 1105-1134.

발명의 내용

해결하려는 과제

- [0009] 기존의 디지털 트윈 구현 기술은 주로 정적인 좌표 기반 매핑이나 제한적인 센서 모니터링 방식에 의존하므로, 시간의 흐름에 따라 누적되는 시공간 정보의 동적 변화를 구조적으로 표현하지 못하며, 현실 공간의 상태 변화 이력을 연산 가능한 형태로 축적하고 분석하는 데 한계가 있다. 특히, 현실에서 수집되는 연속적 센서 데이터를 시계열적으로 구조화하여 누적 저장하고, 이후 시뮬레이션 및 예측 연산에 활용할 수 있는 구조적 기반이 부족하다.
- [0010] 첫째, 본 발명은 시간의 흐름을 기준으로 일정한 단위 시간(Time Singularity Unit; TSU)이 경과할 때마다 생성되는 디지털 정육면체 구조의 단위큐브(Unit Cube)를 3차원 격자 형태로 $8 \times 8 \times 8$ 로 적층하여 하나의 Q-Block으로 구성하고, 현실 공간의 센서 또는 외부 시스템으로부터 수집된 위치 및 상태 데이터를 이 단위큐브의 메타데이터로 매핑함으로써, 시공간 누적 구조를 디지털 트윈으로 구현할 수 있는 기반을 마련한다. 이 구조는 현실 상태의 지속적인 적층 기록 및 변화 분석을 가능하게 하며, 디지털 트윈을 시공간 흐름을 반영하는 정보 구조체로 확장한다.
- [0011] 둘째, 본 발명은 센서 데이터 기반 Q-Block 생성 시, 각 단위큐브에 포함된 메타데이터(예: 온도, 진동, 유동량, 에너지 상태 등)를 기준으로 시간 축에 따른 연산 가능한 트윈 구조를 형성하며, 이로부터 시뮬레이션, 예측 연산, 경향 분석 등의 후속 처리가 가능하도록 설계된다. 이러한 트윈 구조는 단순 복사형 모델이 아니라, 현실과 가상의 시계열 동기화가 가능한 구조로 기능하며, 시뮬레이션 중에 발생하는 변수 조건을 현실 구조에 피드백하거나 반대로 현실 정보를 가상 환경에 실시간 반영할 수 있도록 지원한다.
- [0012] 셋째, 본 발명은 Q-Block 단위의 정보 구조가 GPU 기반 병렬 연산 환경에 최적화되도록 설계되어 있으며, 각 단위큐브는 개별 CUDA 스레드에 매핑될 수 있도록 정의된다. 이를 통해 실시간으로 수집되는 대규모 센서 데이터를 병렬 연산으로 처리하고, 트윈 구조 내 시공간 상태의 변화와 흐름을 시각화하며, 변화 로그를 자동 축적한다. 이러한 병렬 연산 기반 구조는 대규모 인프라 모니터링, 스마트 시티, 재난 대응 등 고빈도 데이터 환경에서 고속성과 확장성을 동시에 만족시킨다.
- [0013] 넷째, 본 발명은 기존의 디지털 트윈이 단순히 ‘좌표 매핑’ 또는 ‘센서 시각화’에 그쳤던 구조를 넘어서, 시간 기반 적층 구조, 시뮬레이션 연계성, 상태 변화 기록의 누적성 등을 갖춘 고차원적 트윈 시스템을 제공하는 것을 기술적 목표로 한다. 특히, 본 발명은 현실 세계의 센서 데이터가 Q-Block이라는 병렬 연산 단위로 구조화되고, TSU 단위로 누적되며, 시뮬레이션 결과와의 비교 또는 예측 모델의 피드백이 가능한 구조를 통해, 미래형 디지털 트윈 환경에서의 핵심 인프라로 작동할 수 있다.

과제의 해결 수단

- [0014] 상기와 같은 기술적 과제를 해결하기 위하여, 본 발명은 시간 특이점(Time Singularity)을 기준으로 일정 시간 단위(TSU: Time Singularity Unit)마다 생성되는 디지털 단위큐브(Unit Cube)를 3차원 격자 구조로 $8 \times 8 \times 8$ 적층하여 구성된 Q-Block을 시공간 정보의 기본 단위로 정의하고, 현실 공간으로부터 수집되는 센서 데이터를 단위큐브의 메타데이터로 실시간 매핑함으로써, 시간 기반으로 누적되는 시계열 디지털 트윈 구조를 구현하는 시스템을 제공한다.
- [0015] 본 시스템은 현실 공간의 다양한 상태 정보(예: 위치, 온도, 밀도, 진동, 유동량 등)를 TSU 단위로 분할된 시공간 블록에 구조화하여 집적하며, 이러한 정보가 포함된 Q-Block을 병렬 연산 기반의 디지털 트윈 적층 구조로 형성한다. Q-Block 내부는 각 단위큐브의 생성 시점, 위치 좌표, 고유 식별자 및 센서 기반 메타데이터를 포함하고 있으며, 해당 정보는 시간의 흐름에 따라 연속적으로 적층·확장되어 트윈 구조의 시계열성을 유지한다.
- [0016] 각 Q-Block은 GPU 기반 병렬 연산 환경에서 개별 CUDA 스레드에 할당되어 연산되며, 센서 데이터의 실시간 처리, 변화 탐지, 예측 모델 적용 등을 포함하는 병렬 연산 태스크가 큐 단위로 배정된다. 시스템은 복수의 Q-Block에 대한 병렬 처리 결과를 공유 메모리 상에서 통합·집계하여 시공간상의 상태 변화를 시뮬레이션하거나, 예측 분석 결과를 시각화 출력할 수 있다.
- [0017] 특히 본 발명에서는 Q-Block 간의 연결 관계를 시간 순서(TSU 차이), 메타데이터 유사도, 정보 밀도 등의 요소에 따라 그래프 구조로 해석하며, 이를 기반으로 현실 공간에서의 트윈 구조 확산, 정보 흐름 예측, 변화 패턴 탐지 등의 연산을 가능하게 한다. 이러한 구조는 고속성 및 확장성을 동시에 만족시켜, 대규모 인프라 모니터링, 재난 대응 시뮬레이션, 도시 운영 디지털 트윈 등 다양한 응용 분야에 적용될 수 있다. 아울러, 본 발명은 트윈 구조의 상태 변화를 시각화 인터페이스를 통해 실시간으로 사용자에게 제공하며, 각 Q-Block의 연

산 상태, 메타데이터 변화, 적층 구조 등을 3차원 공간 상에 시물레이션 형태로 표현함으로써, 변화 흐름의 이해도와 예측 가능성을 향상시킨다.

1. 중심 단위큐브(anchor cube)에 기반한 시계열 정렬 및 정보 흐름 구조

Q-Block은 총 512개의 단위큐브로 구성되며, 이들은 현실 공간으로부터 수집된 센서 데이터를 기반으로 TSU(Time Singularity Unit) 단위로 생성되며, 그 생성 시점과 공간 좌표가 상이하다. 이에 따라, 본 발명은 Q-Block 내 연산 흐름을 정렬하고 정보 흐름의 기준점을 제공하기 위해, 격자 구조상 중심 좌표인 (4,4,4)에 위치한 단위큐브를 중심 단위큐브(anchor cube)로 정의한다. 중심 단위큐브는 Q-Block 내 연산 정렬의 기준점이자 시물레이션 및 시계열 트윈 적층의 중심축으로 작용한다. 연산 프레임이 시작되면 중심 단위큐브를 기준으로, 나머지 511개의 단위큐브에 대해 유클리드 거리 기반 상대 우선순위를 계산하고, 여기에 메타데이터 유사도, 정보 밀도, 상태 플래그 등을 가중치로 반영하여 병렬 연산의 실행 순서를 정렬한다. 이 흐름은 GPU 환경에서는 CUDA 스레드 할당 기준이 되며, 정보 흐름의 확산 방향을 중심 단위큐브에서 외곽 방향으로 설정하는 파장형 병렬 연산 모델을 구성한다.

또한 중심 단위큐브는 이전 TSU에서의 중심 단위큐브와 연계되어 트윈 적층의 시계열 흐름을 형성한다. 예를 들어, 이전 시점 Q-Block의 anchor에 기록된 메타데이터는 후속 Q-Block의 초기 anchor 상태 결정 또는 예측 시드 값으로 활용될 수 있다. 이 구조는 시계열 디지털 트윈의 누적 적층 구조를 효율적으로 제어하며, 정보 흐름의 방향성과 변화 트렌드를 파악할 수 있는 기반을 제공한다. 이와 같은 중심 단위큐브 중심 정렬은 연산 정렬 뿐만 아니라, 트윈 구조 내 정보 중심성 평가, 우선순위 기반 시물레이션 연산, 시각화 프레임 정렬, AI 기반 학습 모델의 입력 순서 등 다양한 기능에서 핵심 기준점으로 활용된다. 특히, 연산 흐름 내에서 메타데이터 기반 중심성이 변화하는 경우, 시스템은 중심 단위큐브를 재설정하거나 동적으로 보완 anchor를 설정함으로써, 정보 흐름의 적응성과 연산 최적화를 동시에 만족시킬 수 있다.

표 1

Q-Twin™ 내 중심 단위큐브 기반 정보 정렬 및 연산 구조 요약

항목	설명
중심 단위큐브 위치	Q-Block 내 좌표 (4,4,4) 기준 중심 단위큐브
정렬 기준	유클리드 거리, 메타데이터 유사도, 상태 플래그, 정보 밀도 등
연산 순서 구성	anchor → 인접 큐브 → 외곽 큐브 순의 파장형 병렬 연산 흐름
GPU 연산 매핑	각 큐브를 CUDA threadIdx 기준으로 자동 병렬 스레드에 할당
시계열 흐름 형성	이전 anchor의 메타데이터 → 후속 Q-Block anchor 상태 결정에 활용
확장성	anchor 재지정 또는 보조 anchor 설정 가능, AI 학습 기반 동적 구조 대응 가능

2. 중첩 정보 집중 분석 및 정보 중심성 평가 구조

본 발명에서 정의하는 중심 단위큐브(anchor cube)는 단순한 연산 시작점을 넘어서, Q-Block 내부 정보의 집중도 및 연산 흐름 중심성을 평가하는 기능적 허브로 작용한다. Q-Block은 시간 단위(TSU)마다 누적된 512개의 단위큐브로 구성되며, 각 큐브는 정보 밀도, 에너지 상태, 중첩 확률, 생성 시점, 상태 벡터 등 다양한 메타데이터를 포함한다. 이러한 정보는 실시간 센서로부터 수집된 데이터 또는 외부 시스템과의 인터페이스를 통해 주입되며, Q-Block 내에 시계열적으로 구조화된다.

중심 단위큐브는 일정한 연산 주기마다 주변 단위큐브들로부터 메타데이터를 수집하고, 이를 거리 기반 계층 구조(예: 유클리드 거리 또는 격자 레벨)에 따라 정렬하며, 가중치 기반의 평가를 수행한다. 이때 각 메타데이터 항목은 사전에 정의된 중요도(weight set)에 따라 상대적 중요도를 산출하며, 이를 통해 현재 Q-Block 내부에서 가장 정보가 집중된 방향성 또는 연산 최적화가 가능한 핵심 큐브들을 식별한다. 이와 같은 구조는 단위큐브 간의 연산 우선순위를 재설정하거나, 중심 단위큐브를 기준으로 연산 흐름을 재조정하는 데 활용되며, 특히 디지털 트윈 시물레이션의 초기 연산 조건 설정, 변화 감지 기반 시나리오 실행 시 중심성 이동 판단, 정보 중심성 기반 시각화 흐름 제어, 강화학습 기반 예측 모델의 학습 입력 구성과 같은 고차원 연산에 핵심 기준을 제공한다. 이 구조를 실행하는 핵심 알고리즘은 가중치 필터링 기반 메타데이터 선별 알고리즘(Weighted Filtering-Based Metadata Selection Algorithm)으로, Q-Block 내 각 단위큐브가 보유한 메타데이터 집합을 기반으로 연산에 유의미한 정보를 중심 단위큐브로 집계한다.

표 2

[0025]

중심 단위큐브 기반 메타데이터 집중 분석 흐름 요약

항목	설명
데이터 종류	밀도, 에너지, 중첩 확률, TSU 차이, 상태 벡터 등
집계 방식	거리 기반 계층화 + 가중치 기반 중요도 계산
계산 기준	중심 단위큐브에서 반경 거리 내 단위큐브를 우선 선별
연산적 활용	병렬 연산 큐 정렬, 경로 탐색 초기화, 정보 흐름 방향 판단
실행 알고리즘	Weighted Metadata Filtering Algorithm

표 3

[0026]

알고리즘 요약(의사코드)

```
def compute_importance(cube, weights):
    return (
        weights["density"] * cube.density +
        weights["energy"] * cube.energy +
        weights["overlap"] * cube.overlap +
        weights["tsu"] * cube.tsu_delta +
        weights["vector"] * cube.vector_magnitude
    )

def filter_top_cubes(cubes, anchor_coord, radius, weights, top_n):
    filtered = []
    for cube in cubes:
        if euclidean_distance(cube.coord, anchor_coord) <= radius:
            score = compute_importance(cube, weights)
            filtered.append((score, cube))
    filtered.sort(reverse=True, key=lambda x: x[0])
    return [c[1] for c in filtered[:top_n]]
```

[0027]

상기 알고리즘은 중심 단위큐브를 기준으로 일정 반경 내 단위큐브 중에서 연산 우선순위가 높은 큐브를 선별하고, 후속 시뮬레이션이나 예측 연산의 초기 조건으로 활용한다. 이때 정렬된 중요도 순위는 병렬 연산 큐 구성, AI 학습 입력 데이터 생성, 텔레포테이션 경로 평가 등의 흐름에 적용 가능하다.

표 4

[0028]

중심 단위큐브 기반 중요도 평가의 적용 예

적용 영역	기대 효과
경로 탐색 초기화	정보가 집중된 방향으로 연산 경로를 설정하여 효율성 증가
연산 우선순위 정렬	중요도가 높은 큐브를 먼저 병렬 큐에 배치하여 전체 흐름 최적화
AI 학습 데이터 전처리	불필요한 정보 제거 후 핵심 정보만 추출하여 고정밀 예측 모델 훈련에 활용
연산 밀도 예측 및 분산	특정 지역의 연산 집중도 증가 감지 후 사전 리소스 재배분 및 충돌 방지
시뮬레이션 초기 조건 설정	실시간 상태 중심 단위 기준으로 가상 모델 초기화 및 정책 판단에 활용 가능

[0029]

3. GPU 연산 스레드의 중심 좌표 매핑 및 연산 흐름 동기화

[0030]

본 발명에서 정의하는 Q-Block은 CUDA 기반의 GPU 병렬 연산 환경에서 하나의 연산 블록(CUDA block)으로 매핑되며, 내부에 포함된 512개의 단위큐브는 각각 독립된 스레드(threadIdx)에 배정되어 병렬 연산된다. 이 구조는 실시간 디지털 트윈 연산의 고속성과 병렬성 확보를 위한 기반이 되며, 특히 Q-Block의 중심 좌표인 (4,4,4)에 위치한 중심 단위큐브(anchor cube)는 전체 연산 흐름의 논리적 중심축이자, GPU 스레드 간 동기화 기준점으로

기능한다.

- [0031] 중심 단위큐브는 연산 초기화 시점에 다른 단위큐브들과 비교 연산을 수행할 때 기준값(reference value) 또는 초기 조건(initial condition)을 제공하며, 이 기준은 연산 스코어(score), 상태 벡터 비교, 에너지 상태 기반 동작 전환 등의 다양한 연산 흐름에서 공통 참조점으로 사용된다. 예를 들어, 중심 단위큐브의 메타데이터 중 중첩 확률(overlap probability), 에너지 상태, 정보 밀도는 인접 단위큐브들의 연산 가중치 산출 시 핵심 기준값으로 작용한다.
- [0032] 또한 중심 단위큐브는 단순 기준값 이상의 역할로서, 연산 큐(task queue) 내 연산 트리거(trigger) 또는 병렬 연산 개시점으로 작동할 수 있다. 예컨대 중심 단위큐브를 중심으로 한 일정 반경 내의 단위큐브들에서 일정 이상의 연산 스코어가 집계될 경우, 시스템은 이를 병렬 연산 시작 조건으로 판단하고 전체 Q-Block 연산을 자동 개시하도록 설계할 수 있다. 이와 같은 구조는 다음과 같은 GPU 병렬 연산 기반 제어 흐름에 적용된다;
- [0033] (1) __syncthreads()를 기준으로 중심 단위큐브가 연산 개시 조건에 도달했을 때, 전체 CUDA thread 블록을 동기화
- [0034] (2) __syncthreads_or(trigger_flag)와 같이 조건부 동기화 시, 중심 단위큐브의 연산 상태를 기준으로 병렬 연산 타이밍을 조율
- [0035] (3) 연산 흐름의 시작점 및 종료 조건을 anchor 중심으로 지정하여, 파동처럼 확산되는 연산 흐름 구조를 생성

표 5

[0036] Q-Twin™ 내 중심 단위큐브의 GPU 연산 제어 기능 요약

역할	설명
기준값 전달	연산 가중치 계산, 상태 비교 등에서 메타데이터 기반 reference 역할 수행
연산 트리거	중심 반경 내 상태 스코어 임계 도달 시 병렬 연산 시작 조건으로 작용
동기화 허브	__syncthreads 기준의 thread 간 연산 타이밍 조율 핵심
논리적 중심	CUDA threadIdx 기준 연산 중심축으로서 정보 흐름 제어의 앵커 역할
연산 최적화	거리 및 유사도 기반 연산 우선순위 부여 및 흐름 방향 결정

- [0037] Q-Twin™에서 이와 같은 GPU 중심 연산 제어 구조는 단위큐브 간 메타데이터 유사도, TSU 시간 거리, 중심성 평가 결과 등을 기반으로 연산 흐름의 속도 및 정확성을 동적으로 최적화할 수 있다. 특히, 현실 공간으로부터 실시간으로 유입되는 센서 데이터가 Q-Block 단위로 구조화될 때, 중심 단위큐브를 기준으로 연산 순서 및 흐름을 정렬함으로써, 시계열 디지털 트윈 구조의 정보 흐름 통제 및 시뮬레이션 정확도 향상에 기여한다. 이러한 연산 흐름의 정밀 제어는 향후 Q-Block 간 연결 구조(anchor-to-anchor 기반)에서도 연산 시드(seed value)의 기점으로 확장되며, 복수 Q-Block 간의 시공간 정보 전이 및 예측 모델 실행 시 중요한 기준점으로 활용된다.
- [0038] 4. Q-Block 간 anchor-to-anchor 연결 구조 및 트윈 적층 흐름
- [0039] 본 발명에서 정의하는 Q-Block 간 anchor-to-anchor 연결 구조는, 시간 특이점(TSU: Time Singularity Unit)을 기준으로 생성되는 복수 Q-Block들이 시계열적으로 적층되며 형성하는 디지털 트윈 구조에서, 각 Q-Block의 중심 단위큐브(anchor cube)를 연결축으로 삼아 정보 흐름을 유지하고 연산 구조의 연속성을 보장하는 핵심 메커니즘이다. 각 Q-Block은 $8 \times 8 \times 8$ 구조의 단위큐브들로 구성되며, 시간 흐름에 따라 순차적으로 생성된다. 이때 각 시점(TSU_n)에 생성된 Q-Block의 중심 단위큐브(anchor cube)는, 직전 시점(TSU_{n-1}) Q-Block의 중심 단위큐브와 연결되어, 시공간상의 트윈 적층 흐름을 형성한다. 이러한 구조는 다음과 같은 3단계 계층 구조를 따른다;
- [0040] (1) 물리적 연속성 : 각 중심 단위큐브의 위치는 동일 좌표계 또는 보정된 상대 좌표계 상에서 연속적인 시공간 흐름을 형성하며, 트윈 구조 상 위치 매핑의 기준점이 된다.
- [0041] (2) 정보 연속성 : 직전 중심 단위큐브가 보유한 메타데이터(예: 정보 밀도, 상태벡터, 에너지 상태)는 후속 중심 단위큐브의 초기 조건 설정에 직접 반영되며, anchor-to-anchor 간 정보 전이가 수행된다.
- [0042] (3) 연산 흐름 연속성 : 각 중심 단위큐브는 이전 시점에서 연산된 결과를 기반으로 후속 anchor의 연산 방향 및 병렬 연산 흐름 초기화를 유도하며, 시계열 연산의 파장적 확산 구조를 유지한다.
- [0043] 이러한 구조 하에서, anchor-to-anchor 연결은 단순한 포인터 연결이 아닌, 메타데이터 기반의 시공간 연산 그

래프(Spatiotemporal Execution Graph)로 구성된다. 각 연결은 TSU 간 시간 거리(ΔTSU), 중심 단위큐브 간 위치 차이($\Delta X, \Delta Y, \Delta Z$), 메타데이터 유사도(cosine similarity, overlap rate), 연산 영향력(propagation weight)을 포함할 수 있다. 이러한 연산 기반 연결 구조는 시계열 디지털 트윈의 누적 적층을 단순 시각화가 아닌 예측 가능한 시뮬레이션 실행 구조로 전환시키며, 연산 트리거, 시뮬레이션 재현, 고밀도 예측 등에 모두 응용 가능하다.

표 6

[0044]

Q-Block 간 anchor-to-anchor 연결 흐름 및 기능 요약

요소	설명
anchor-to-anchor 연결 기준	TSU 차이 + 좌표차 + 메타데이터 유사도 기반
연결 목적	트윈 구조의 시간축 적층 및 정보 연산 흐름의 연속성 유지
정보 전이 항목	중심 단위큐브의 메타데이터, 연산 결과값, 상태 벡터 등
연산 흐름 전이 방식	후속 anchor에 이전 anchor의 연산 결과 반영하여 초기 연산 흐름 설정
시계열 연산 흐름 구성	$TSU_0 \rightarrow TSU_1 \rightarrow \dots \rightarrow TSU_n$ 구조로 누적 적층

[0045]

본 발명의 anchor-to-anchor 연결 구조는 다양한 디지털 트윈 연산 시나리오에 유연하게 적용될 수 있으며, 예를 들어, TSU_n 시점에서 실시간으로 수집된 데이터를 기반으로 트윈 연산이 실행될 경우, 직전 시점인 TSU_{n-1} 의 중심 단위큐브(anchor cube)에서 전달된 상태 정보를 바탕으로 변화 패턴을 정밀하게 예측할 수 있다. 또한, 이와 같은 anchor 간 연결성은 단순한 시계열 적층을 넘어, 각 시점의 anchor 메타데이터를 추적하고 호출함으로써 디지털 트윈 시뮬레이션을 되감거나 특정 시점으로 리셋하는 기능을 가능하게 하며, 과거 상태의 복원 또는 변화 이력의 시각적 재현에 활용될 수 있다. 나아가, 연속된 anchor-to-anchor 경로 상에 AI 예측 모델을 적용함으로써, 향후 TSU에서의 고밀도 정보 발생 가능성을 사전에 예측하고, 그에 따른 경로 분기 또는 자원 분산 정책을 수립할 수 있는 예측 기반 경로 시뮬레이션 또한 구현 가능하다.

발명의 효과

[0046]

본 발명은 Q-Block 기반의 병렬 시공간 연산 구조와 중심 단위큐브(anchor cube)를 활용한 정보 집중 및 우선순위 정렬 로직을 통해, 기존 단위큐브 기반의 단일 처리 방식이 가진 연산 성능 및 유연성의 한계를 극복하고, 시간 축 상에서 적층되는 디지털 트윈의 시계열 연산 구조를 정교하게 구현할 수 있는 기술적 기반을 제공한다. 구체적으로, 본 발명은 다음과 같은 기술적·산업적 효과를 갖는다;

[0047]

1. 병렬 연산 효율의 극대화

[0048]

$8 \times 8 \times 8$ 구조의 Q-Block은 CUDA 기반 GPU 병렬 연산 환경에 최적화되어 있으며, 각 단위큐브를 독립된 GPU 스레드에 자동 할당함으로써 연산 병목 없이 고속 병렬 처리가 가능하다. 중심 단위큐브를 기준으로 연산 우선순위를 정렬하는 구조는 thread 동기화의 기준점으로 활용되어 전체 연산 흐름의 일관성과 효율성을 높이며, 디지털 트윈의 실시간 적층 처리에 있어 연산 안정성을 확보할 수 있다.

[0049]

2. 시공간 트윈 경로 탐색의 정밀도 향상

[0050]

Q-Block 간의 연산 경로는 단순한 거리 기반이 아닌, 중첩 확률, 메타데이터 유사도, TSU 시간 거리 등을 통합한 가중치 기반 알고리즘으로 탐색된다. 이에 따라 시계열 디지털 트윈 구조 내에서의 이동 경로, 변화 추세, 정보 집중 영역 등을 고정밀로 예측 가능하며, 이는 자율비행체(UAM, 드론) 항법 시스템은 물론, 다차원 시뮬레이션 및 텔레포테이션 흐름 재현 등에도 활용될 수 있다.

[0051]

3. 정보 흐름 기반 트윈 연산 집중성 구현

[0052]

중심 단위큐브는 단순 연산 시작점이 아닌, 메타데이터 기반 중요도 분석 및 필터링을 통해 연산 자원의 집중 방향과 흐름을 제어하는 중앙 허브로 작동한다. 이를 통해 Q-Block 내에서 연산 우선 대상이 명확히 정렬되고, 전체 디지털 트윈 적층의 흐름상에서 정보 전파의 안정성과 방향성을 구조적으로 제어할 수 있다.

[0053]

4. AI 학습 연계 및 자율 최적화 연산 구조 제공

[0054]

본 발명의 구조는 Q-Block 내에서 발생하는 연산 패턴, anchor 간 전이 이력, 중요도 기반 정렬 결과 등을 AI 학습 입력으로 제공할 수 있어, 강화학습 기반의 자율 트윈 경로 예측, 변화 탐지, 혼잡 회피 등에 응용 가능하다.

다. 또한, anchor-to-anchor 기반의 연산 흐름은 AI 정책의 시계열 최적화에도 적용되어 다층 시공간 시뮬레이션에서 연산 시드(seed)로 활용 가능하다.

[0055] 5. 산업 전반으로의 적용 확장성

[0056] 본 발명은 메타버스 기반 도시 트윈 구현, 재난 대응 훈련 시뮬레이션, 고밀도 교통 예측, 자산화 가능한 디지털 공간 토큰 관리, AI 기반 공간 변화 예측, 양자 상태 시뮬레이션 등 다양한 응용 시나리오에 적용 가능하다. 각 산업 분야에서는 Q-Block의 시계열 적층 구조, 중심 단위큐브 기반 병렬 연산 흐름, anchor-to-anchor 정보 전이 구조가 강력한 기술적 기반으로 작용하며, 디지털 트윈의 미래형 구현에 필수적인 연산 프레임워크를 제공한다.

도면의 간단한 설명

[0057] 도 1은 본 발명에 관한 신규 메타버스 행성 생성을 위한 척도를 이루는 단위큐브와, 가상의 우주가 탄생하는 시작점을 이루는 특이점을 기준으로 한 공간 구성 방식의 개념을 도시한 것이다.※ 본 도면은 원출원(출원번호 KR1020220016546) 공통 개념도로, 본 발명을 포함하는 Q-Block 기반 기술 계열 전체 출원에서 동일하게 사용된다.

도 2는 본 발명에 관한 상기 특이점에 의한 무한 확장 가능성이 자연법칙에 준하는 기술적 사상의 당위성에 의거한 것임을 보이기 위한 예시이다. ※ 본 도면 역시 Q-Block 기반 기술 계열 전체 출원에서 개념 공유가 이루어지며, 발명의 철학적 배경 및 기술 기초 구조를 설명한다.

도 3은 Q-Block 내부에 포함된 중심 단위큐브(anchor cube)를 기준으로, 병렬 연산 흐름이 중심에서 인접 큐브로, 다시 외곽 큐브로 확산되는 구조를 나타낸 개념도이다. 중심 기준의 정보 파장이 연산 큐 구성에 영향을 미치는 구조를 도시한다.

도 4는 중심 단위큐브(anchor cube)의 중요도 계산 및 병렬 연산 큐 배정 과정을 도시한 흐름도이다. Q-Block 내 512개의 단위큐브 각각은 기준 단위큐브(좌표 (4,4,4))로부터의 거리 및 메타데이터(밀도, 에너지, 중첩 확률, TSU 차이, 상태 벡터 등)에 따라 상대 중요도(importance score)를 산출받는다. 이때, 각 메타데이터 항목은 사전에 정의된 가중치 테이블에 의해 가중화되며, 해당 중요도 값은 연산 우선순위를 결정하는 기준으로 사용된다. 계산된 중요도 값을 기준으로 단위큐브들을 내림차순 정렬한 후, 중요도 순으로 병렬 연산 태스크 큐(priority queue)에 배정되며, CUDA 기반 GPU 연산 환경에서는 각 단위큐브가 threadIdx 구조에 따라 자동으로 병렬 쓰레드에 할당된다. 이와 같은 흐름은 중심 큐브로부터 연산 파장이 외곽으로 확산되도록 구성함으로써, 병렬 연산 구조의 공간적·시간적 정렬성과 연산 효율성을 동시에 확보한다.

도 5는 메타데이터 기반 가중치 필터링 알고리즘의 흐름도를 도시한 도면이다. 중심 단위큐브(anchor cube)를 중심으로 일정 반경 내에 존재하는 단위큐브들을 선별한 후, 각 큐브의 메타데이터 항목(밀도, 에너지 상태, 중첩 확률, TSU 이력, 상태 벡터 등)에 사전 정의된 가중치를 적용하여 상대 중요도를 계산한다. 중요도 값에 따라 단위큐브들을 내림차순 정렬하고, 상위 중요도 큐브들만을 선별하여 후속 연산에 반영함으로써 연산 집중도, 탐색 방향, 정보 흐름의 효율성을 극대화한다.

도 6은 Q-Block 간 중심-to-중심(anchor-to-anchor) 연결 구조를 도시한 개념도이다. 각 Q-Block의 중심 단위큐브는 시간 흐름에 따라 위쪽으로 적층되며, 이들 간의 메타데이터 유사도, 중첩 확률, 정보 전파 이력 등을 기반으로 연결 강도가 계산된다. 연결 강도에 따라 Q-Block 간의 정보 전이 경로가 설정되며, 해당 경로는 A* 알고리즘 또는 다익스트라 알고리즘 기반의 시공간 경로 탐색에서 탐색 시드(seed)로 활용된다. 이와 같은 anchor-to-anchor 연결 구조는 연산 파장의 방향성과 전이 확률에 대한 예측 정밀도를 높이는 데 기여한다.

도 7은 중심 단위큐브가 CUDA 기반 GPU 연산 내에서 수행하는 기준값 전달, 연산 트리거, 동기화 허브 등의 기술적 역할을 도시한 개념도이다. Q-Block 내 중심 단위큐브는 병렬 연산이 시작될 때 초기 reference value 제공, 병렬 태스크의 우선순위 트리거, thread 간 연산 동기화 기준점 등 다양한 연산 제어 기능을 수행한다. 특히 GPU의 threadIdx 구조 내에서는 연산 흐름의 논리적 중심으로서 작용하며, anchor 중심의 파장 확산 연산 구조를 가능하게 하는 핵심 노드로 기능한다.

도 8은 본 발명의 Q-Twin™ 시스템에서 GPU 병렬 연산(CUDA)에 의해 처리되는 Q-Block 단면(z=4 평면)의 시뮬레이션 결과를 도시한 예시 도면이다. 좌측은 초기 상태에서의 단위큐브들의 상태 분포를 나타내며, 우측은 중심 단위큐브(anchor cube; 좌표 (4,4,4))를 기준으로 연산 가중치가 적용된 후 각 단위큐브의 상태가 변화된 모습을 시각적으로 보여준다. 각 단위큐브는 anchor와의 메타데이터 유사도 및 TSU 기반 거리 등에 따라 연산 우선

순위 및 가중치가 달리 부여되며, 이로 인해 연산 결과가 파장처럼 확산되는 구조로 표현된다.

도 9는 본 발명의 Q-Twin™ 시스템이 Unreal Engine 또는 Unity와 연동된 상태에서 수행된 시각화 예시를 도시한 도면이다. 좌측은 $z=4$ 평면 상의 Q-Block 단위큐브 각각의 정보 밀도(density) 분포를 색상으로 시각화한 결과로, 실시간 디지털 트윈 시뮬레이션 중 단위큐브 상태를 직관적으로 확인할 수 있도록 한다. 우측은 중심 단위큐브(anchor cube)를 기준으로 anchor-to-anchor 경로 상에서 예측된 미래 위험도 점수를 시각화한 화면으로, 이는 시뮬레이션 상 고밀도 연산 영역, 경로 충돌 가능성, 미래 TSU에서의 집중 연산 흐름 등을 사전에 식별할 수 있도록 한다. 이러한 시각화 구조는 Q-Twin™ 시스템이 제공하는 디지털 트윈의 시간 누적 기반 예측 기능 및 현실 공간 반영 구조의 기술적 효과를 명확히 표현한다.

도 10은 Q-Twin™ 시스템에서 시계열로 축적된 중심 단위큐브(anchor cube)의 메타데이터 흐름을 시각화한 예시로서, TSU(Time Singularity Unit) 단위로 생성된 각 Q-Block의 중심에서 수집된 정보 밀도, 에너지 상태, 중첩 확률의 변화 추이를 시간 축을 따라 나타낸다. 본 도면은 anchor-to-anchor 기반 연산 구조에서 중심 단위큐브가 시공간 연산의 시드(seed)로 기능하는 흐름을 직관적으로 표현하며, 실시간 디지털 트윈 시뮬레이션에서의 예측 기반 연산, 경로 탐색 초기화, 정보 집중 영역의 시각적 파악 등에 활용될 수 있는 핵심 구조를 설명한다.

도 11은 Q-Twin™ 구조에서 각 TSU별 중심 단위큐브의 메타데이터를 시계열로 시각화한 것으로, anchor-to-anchor 연산 흐름의 예측 및 경로 최적화 흐름을 시뮬레이션한 예를 나타낸다.

도 12는 Q-Twin™ 시스템에서 중심 단위큐브(anchor cube)를 기준으로 병렬 연산이 확산되는 흐름을 시각적으로 표현한 예시이다. 중심 단위큐브는 Q-Block의 중앙 좌표((4,4,4))에 위치하며, 해당 도면은 연산 트리거가 중심에서 발생하여 외곽 단위큐브로 파동처럼 전파되는 구조를 나타낸다. 각 단위큐브는 연산 우선순위에 따라 색상 또는 투명도로 구분되며, 이 구조는 기존의 grid-stride loop 방식과 달리 anchor-to-anchor 기반 연산 흐름 정렬 및 타이밍 조율 기능이 적용된 Q-Twin™ 고유의 병렬 연산 제어 방식을 시각화한 것이다.

도 13은 Q-Twin™ 시스템 내에서 anchor-to-anchor 기반 시공간 그래프 탐색 및 AI 정책 학습 흐름을 시각적으로 도시한 것으로, Q-Block 내 연산 구조로부터 시공간 그래프를 구성하고, 이를 기반으로 강화학습 에이전트가 경로 탐색 및 연산 정책을 반복 학습하는 전체 과정을 단계적으로 표현한다. 도면은 중심 단위큐브(anchor cube) 단위로 구성된 TSU 기반 Q-Block들이 시간 순으로 적층되며, 각 anchor 간 연결 강도(edge weight)를 통해 spatiotemporal graph가 형성되는 구조를 보여준다. 이어서, 이 그래프 상에서 A* 또는 Dijkstra 알고리즘에 의해 경로가 탐색되고, 탐색 결과는 AI 정책 네트워크의 초기 시드(seed policy)로 전달되어, 강화학습 기반의 정책 최적화가 수행되는 일련의 흐름이 시각화된다. 이를 통해, Q-Twin™이 단순한 병렬 연산 엔진을 넘어 시공간 예측과 자율 연산 최적화를 동시에 실현하는 기술임을 직관적으로 이해할 수 있도록 설계된 도면이다.

도 14는 Q-Twin™ 시스템에서 병렬 연산 흐름, anchor-to-anchor 경로, 연산 중요도 분포 등을 통합적으로 시각화하는 사용자 인터페이스 구조를 나타낸 것이다. 연산 파장 확산 애니메이션, 히트맵 기반 연산 집중 영역 표현, 시뮬레이션 제어 슬라이더 및 AI 학습 시각화 인터페이스 등으로 구성된다.

도 15는 Q-Twin™ 시스템을 기반으로 구성된 재난 대응 훈련 시뮬레이션 환경에서, 단위큐브(Unit Cube)별 위험도(importance score)를 시각화한 3차원 결과를 도시한 것이다. 본 도면에서는 건물 내 센서(온도, 연기, 진동 등)로부터 실시간 수집된 메타데이터를 바탕으로, 중심 단위큐브(anchor cube)를 기준으로 병렬 연산이 수행되며, 연산 결과가 위험 구역의 공간적 확산 형태로 표현된다. 각 큐브는 위험 수준에 따라 색상으로 구분되며, 붉은 계열은 고위험 영역, 푸른 계열은 저위험 영역을 나타낸다. 시각화된 구조는 시간 특이점(TSU)을 기준으로 적층된 Q-Block 내에서 연산 확산 흐름과 anchor-to-anchor 전이 방향성을 직관적으로 이해할 수 있도록 구성되어 있다.

도 16은 Q-Twin™ 시스템에서 강화학습 기반 AI가 미래 TSU(Time Singularity Unit)에서 고중요도(anchor) 영역을 예측한 결과를 시각화한 것이다. 예측된 중심 단위큐브(anchor cube)는 Glow 또는 Particle 효과를 통해 시각적으로 강조되며, 예측된 중요도 점수는 색상 스펙트럼을 통해 표현된다. 특히, 예측된 TSU 시점은 큐브 상단 또는 인터페이스 패널에 명확히 표기되어 사용자가 향후 연산 집중 흐름을 직관적으로 파악할 수 있도록 구성되어 있다. 이 도면은 Q-Twin™이 단순한 현재 상태 시뮬레이션을 넘어서, AI 기반의 미래 연산 흐름 예측 기능을 갖춘 자율형 디지털 트윈 시스템으로 작동함을 보여주는 핵심 시각 자료이다.

도 17은 본 발명의 Q-Twin™ 시스템이 Unreal Engine 또는 Unity와 연동되어, 도심 내 드론 또는 UAM과 같은 공간 비행체의 이동 경로를 기준으로 단위큐브 기반 정보 밀도 및 위험도 예측 결과를 3D로 시각화한 예시를 도시한 도면이다.

발명을 실시하기 위한 구체적인 내용

- [0058] 본 발명은 현실 공간의 정보를 디지털화하여 시계열적으로 누적·확장 가능한 디지털 트윈 구조를 실현하기 위한 시스템으로서, Q-Block 구조를 기반으로 GPU 병렬 연산 환경에서의 실시간 정보 갱신, 중심 단위큐브(anchor cube)를 기준으로 한 정보 집중 분석, 가중치 기반 큐브 정렬, AI 기반 변화 예측, anchor-to-anchor 기반 상태 이력 전이 및 시물레이션 복원 등 다양한 핵심 기술 모듈을 포함한다. 이러한 각 기술 모듈은 Q-Block 내부 연산뿐만 아니라, Q-Block 간 시간 축상의 트윈 적층 흐름에서도 유기적으로 작동하며, 시공간 기반 디지털 트윈의 정확성, 예측 가능성, 재현성 및 연산 최적화를 동시에 실현할 수 있도록 구성된다. 이하에서는 본 발명의 실시예 및 구현 방식을 중심으로 각 구성 요소의 구조적 특징, 작동 메커니즘, 시각화 연계 방식 등을 단계적으로 설명한다.
- [0059] 1. Q-Block 및 단위큐브 기본 구조
- [0060] Q-Twin™ 시스템에서 사용되는 Q-Block은 $8 \times 8 \times 8$ 구조의 단위큐브로 구성된 병렬 연산 블록으로서, 총 512개의 단위큐브(Unit Cube)를 포함한다. 이 Q-Block 구조는 GPU 기반 병렬 연산 환경에서의 최적화를 목적으로 설계되었으며, 특히 CUDA 아키텍처에 적합하도록 2의 거듭제곱 단위로 정렬된다. $8 \times 8 \times 8$ 이라는 격자 구조는 메모리 정렬, 스레드 간 동기화, 연산 큐 배정의 효율성을 극대화하며, 연산 단위로서 시공간 정밀도와 성능 간의 균형을 이루는 최적 구조이다. Q-Block은 디지털 트윈 구현을 위한 시공간 연산 단위로 사용되며, 시간 특이점(Time Singularity)을 기준으로 일정한 시간 단위(TSU: Time Singularity Unit)마다 생성되는 단위큐브들을 적층함으로써 구성된다. TSU는 디지털 시공간의 최소 단위로 정의되며, 각 TSU 경과 시마다 하나의 단위큐브가 생성되어 현실 세계의 상태 변화를 시간적으로 추적·디지털화하는 기준이 된다.
- [0061] 예를 들어, TSU의 시간 단위는 빛이 진공 내에서 0.1나노초 동안 이동하는 거리인 약 3cm를 기준으로 할 수 있으며, 이는 현실 세계의 물리 단위와 디지털 트윈 시스템 간의 정밀한 동기화 기준을 제공한다. 이러한 정의는 고정된 것이 아니며, 시물레이션 해상도 및 예측 정확도에 따라 가변적으로 설정될 수 있다. 각 단위큐브는 다음과 같은 정보를 포함한다;
- [0062] (1) 고유 식별자(ID) : 단위큐브를 구별하기 위한 UUID 또는 해시 기반 식별값
- [0063] (2) 위치 좌표(x, y, z) : Q-Block 내 상대적 3차원 위치 정보
- [0064] (3) 생성 시점(TSU) : 시간 특이점을 기준으로 단위 시간 경과 값을 나타냄
- [0065] (4) 메타데이터 집합: 정보 밀도, 에너지 상태, 중첩 확률, 상태 벡터 등
- [0066] 이러한 단위큐브들은 단순한 정보 표현 단위를 넘어서, 각 큐브가 GPU의 연산 스레드에 개별적으로 매핑되어 병렬 연산의 단위로 기능한다. 연산 흐름 상에서는 중심 단위큐브(anchor cube)를 기준으로 상대적 거리, 메타데이터 유사도 등에 따라 연산 순위가 결정되며, 시물레이션, 예측, 경로 탐색 등 다양한 연산에서 활용된다. Q-Block은 이러한 단위큐브들을 하나의 논리적·물리적 연산 블록으로 통합하여 시공간 데이터의 디지털 트윈화, 시계열 연산, 메타데이터 기반 정보 흐름 모델링을 가능하게 하며, Q-Twin™ 시스템 전반에서 디지털 트윈의 구축, 시물레이션, 예측 연산의 기반이 되는 핵심 구조이다.
- [0067] 2. 중심 단위큐브(anchor cube) 정의 및 역할
- [0068] Q-Block은 $8 \times 8 \times 8$ 의 정규 격자 구조로 구성되어 있으며, 그 중심 좌표에 해당하는 (4,4,4) 위치의 단위큐브는 연산의 기준점으로 정의된다. 본 발명에서 이 단위큐브를 ‘중심 단위큐브(anchor cube)’라 명명하며, 이는 Q-Block 내부 연산 흐름을 조직화하는 중심 노드로서 다음과 같은 주요 기술적 기능을 수행한다;
- [0069] (1) 연산 정렬의 기준점 역할 : Q-Block은 동일한 시간 단위(TSU) 내에서 생성된 512개의 단위큐브로 구성되며, 이들은 모두 하나의 병렬 연산 블록 내에 포함되지만, 그 각각은 위치 좌표(x, y, z), 생성 시의 메타데이터 상태(예: 정보 밀도, 에너지 상태, 중첩 확률 등)에 따라 상이한 정보를 포함하고 있다. 이러한 단위큐브들 간에는 연산 우선순위의 필요성이 존재하며, 특히 시공간 연산 또는 경로 탐색 과정에서는 정보 전파 순서 및 비교 연산의 기준이 되는 중심점을 설정하는 것이 필수적이다.
- [0070] 이때 Q-Block의 정중앙인 (4,4,4) 좌표에 위치한 단위큐브는, 그 구조적 대칭성과 연산적 중립성에 기반하여 연산의 기준점으로 정의된다. 해당 단위큐브는 나머지 511개 단위큐브 각각과의 유클리드 거리(또는 필요에 따라 맨해튼 거리)를 기반으로 연산 우선순위를 정렬할 수 있으며, 거리 외에도 메타데이터 간의 유사도 분석(예: 상태벡터의 코사인 유사도 등)을 추가 가중치 요소로 적용하여, 보다 정교한 연산 흐름의 정렬이 가능하다.

- [0071] 중심 단위큐브는 이러한 기준 설정 과정을 통해 전체 Q-Block 내의 연산 방향성, 정보 흐름, 우선 처리 순서를 논리적으로 정의하는 역할을 수행하며, GPU 기반 병렬 연산 환경에서는 CUDA 쓰레드 내 연산 배정 시점에도 이 기준점이 기준값(reference value)으로 활용된다. 이는 곧 중심 단위큐브가 단순한 구조상 중심을 넘어서, 연산 논리상 ‘우선 정렬의 앵커(anchor)’로 기능함을 의미하며, 정보 확산과 연산 트리거 작동, 경로 초기 설정 등 전체적인 연산 흐름의 기점을 제공하는 중요한 기술 요소가 된다.
- [0072] (2) 정보 확산의 기점 : Q-Block 내 중심 단위큐브는 연산 정렬의 기준점일 뿐만 아니라, 전체 병렬 연산 흐름의 출발점이자 정보 확산의 기점으로 작용한다. 특히 본 발명에서 정의하는 Q-Block은 GPU 기반의 병렬 연산 환경, 구체적으로 CUDA(CUDA: Compute Unified Device Architecture) 아키텍처를 활용하는 구조로 설계되어 있으며, Q-Block 내부의 512개 단위큐브는 각각 독립된 연산 태스크로써 개별 CUDA thread에 배정된다.
- [0073] 이러한 병렬 연산 구조 내에서 중심 단위큐브는 (4,4,4)라는 격자상의 대칭 중심 좌표를 차지하며, 물리적 위치의 중심성을 넘어서 연산 트리거(trigger) 역할을 수행한다. 구체적으로는, 중심 단위큐브의 상태값(예: 에너지 레벨, 중첩 확률, TSU 기반 정보 누적량 등)이 사전 정의된 임계값에 도달하면, 이를 기준으로 인접한 큐브들에 대한 연산이 동기화되어 병렬적으로 확산된다. 이러한 연산 흐름은 마치 중심에서 외곽으로 퍼지는 파동(wave propagation)처럼 전개되며, 각 큐브는 자신의 거리나 중요도에 따라 연산 큐 내에서 순차적으로 활성화된다.

표 7

[0074]

중심 단위큐브 트리거 기반 병렬 연산 확산 구조 예시(Python Pseudocode)

```
# 단위큐브 클래스
class UnitCube:
    def __init__(self, id, x, y, z, metadata):
        self.id = id
        self.coord = (x, y, z)
        self.metadata = metadata
        self.activated = False # 연산 활성화 여부

    def distance_from_anchor(self, anchor_coord=(4, 4, 4)):
        dx = self.coord[0] - anchor_coord[0]
        dy = self.coord[1] - anchor_coord[1]
        dz = self.coord[2] - anchor_coord[2]
        return (dx**2 + dy**2 + dz**2)**0.5

# 중심 단위큐브 임계값 판단
def should_trigger(center_cube, threshold_energy, threshold_overlap):
    energy = center_cube.metadata.get("energy", 0)
    overlap = center_cube.metadata.get("overlap", 0)
    return energy >= threshold_energy or overlap >= threshold_overlap

# 병렬 연산 큐 구성 (중심에서 외곽으로 거리순 정렬)
def construct_parallel_queue(unit_cubes, center_coord=(4,4,4)):
    return sorted(unit_cubes, key=lambda cube: cube.distance_from_anchor(center_coord))

# 병렬 연산 수행
def execute_parallel_processing(queue):
    for cube in queue:
        # 병렬 스레드에 할당된 연산 (가정)
        process_cube(cube)

# 큐브 연산 (예시)
def process_cube(cube):
    cube.activated = True
    # 실제 연산 로직 (예: 메타데이터 처리)
    print(f"[RUNNING] Cube ID {cube.id} is processed at {cube.coord}")

# 예시 실행
unit_cubes = generate_all_cubes() # 512개의 UnitCube 객체 리스트 생성
center_cube = [c for c in unit_cubes if c.coord == (4,4,4)][0]

if should_trigger(center_cube, threshold_energy=0.8, threshold_overlap=0.5):
    parallel_queue = construct_parallel_queue(unit_cubes)
    execute_parallel_processing(parallel_queue)
else:
    print("중심 단위큐브의 임계값 미달. 연산 보류.")
```

[0075]

중심 단위큐브의 에너지 상태나 중첩 확률이 사전 정의된 임계값을 초과하면, 이를 신호로 하여 연산이 트리거 되고, Q-Block 내 각 단위큐브는 중심 단위큐브로부터의 거리 순으로 정렬되어 연산 우선순위가 부여된다. 이러한 정렬 구조는 중심에서 외곽으로 파동이 확산되듯 연산 흐름을 형성하며, 각 단위큐브는 이 순서에 따라 병렬 큐에 배치되어 CUDA 기반의 GPU 환경에서 동기화된 방식으로 연산을 수행하게 된다. 다음은 상기 "중심 단위큐브 트리거 → 거리 기반 정렬 → CUDA 병렬 연산" 흐름을 CUDA C++ 스타일로 변환한 GPU 병렬 연산 예시이다.

이 코드는 중심 단위큐브 상태값 평가 및 연산 파동 확산을 CUDA 병렬 환경에 맞춰 표현한 것이다;

표 8

중심 단위큐브 기반 CUDA 병렬 연산 트리거 및 중요도 계산 예시 코드

```
// 전역 상수
#define BLOCK_SIZE 8
#define TOTAL_CUBES (BLOCK_SIZE * BLOCK_SIZE * BLOCK_SIZE)
#define CENTER_X 4
#define CENTER_Y 4
#define CENTER_Z 4

// 구조체 정의
struct Cube {
    float energy;
    float overlap;
    float importance;
    int tsu;
    float3 coord;
};

// 중심 단위큐브 기준 임계값
__device__ const float ENERGY_THRESHOLD = 0.75f;
__device__ const float OVERLAP_THRESHOLD = 0.6f;
```

```

// 거리 기반 중요도 계산
__device__ float compute_importance(Cube cube, float3 anchor) {
    float dx = cube.coord.x - anchor.x;
    float dy = cube.coord.y - anchor.y;
    float dz = cube.coord.z - anchor.z;
    float distance = sqrtf(dx*dx + dy*dy + dz*dz);

    // 중요도: 거리 반비례 + 중첩 가중치
    return 1.0f / (distance + 1e-5f) + cube.overlap;
}

// CUDA 커널: 중심 큐브를 기준으로 트리거 → 중요도 계산 → 병렬 연산
__global__ void cube_wave_propagation(Cube* cubes) {
    int idx = threadIdx.x + blockIdx.x * blockDim.x;
    if (idx >= TOTAL_CUBES) return;

    Cube& myCube = cubes[idx];

    // 중심 단위큐브에서만 연산 트리거 확인
    if (idx == ((CENTER_Z * BLOCK_SIZE + CENTER_Y) * BLOCK_SIZE + CENTER_X)) {
        if (myCube.energy > ENERGY_THRESHOLD || myCube.overlap > OVERLAP_THRESHOLD) {
            // 트리거 발생: shared memory 또는 global memory flag로 확산 가능
            // 이 예시에서는 모두 연산 수행
        } else {
            return; // 트리거 조건 미충족 시 skip
        }
    }

    // 중심 좌표
    float3 anchor = make_float3(CENTER_X, CENTER_Y, CENTER_Z);

    // 중요도 계산
    myCube.importance = compute_importance(myCube, anchor);

    // 연산 예시: importance에 따른 weighted update (의미 있는 연산으로 대체 가능)
    float compute_result = myCube.importance * myCube.energy;

    // 결과 기록 (예시용)
    cubes[idx].energy = compute_result; // 단순 덮어쓰기
}

```

[0077] 본 발명에 따른 Q-Block 기반 병렬 연산 구조에서는, 중심 단위큐브(anchor cube)가 연산 개시의 기준점으로 기능한다. 중심 단위큐브는 Q-Block의 중앙에 해당하는 좌표 (4,4,4)에 위치하며, 해당 큐브가 가지는 에너지 상태값(energy level) 또는 중첩 확률(overlap probability)이 사전 정의된 임계값(threshold)을 초과하는 경우, 전체 Q-Block에 대한 병렬 연산이 트리거된다. 이러한 연산 트리거는 GPU 기반 CUDA 병렬 연산 환경에서 각 단위큐브에 배정된 스레드(threadIdx)를 통해 즉시 전파된다. 이후 각 단위큐브는 중심 단위큐브로부터의 거리(Euclidean distance), 중첩 확률, 또는 메타데이터 유사도 등의 요소를 기반으로 상대적인 중요도(importance score)를 계산하며, 이는 다음의 가중치 기반 연산을 통해 수행된다;

[0078] $importance_i = f(1 / 거리_i) + overlap_i$

[0079] 여기서 거리_i는 중심 단위큐브와 해당 단위큐브 간의 3차원 유클리드 거리이며, overlap_i는 해당 큐브의 중첩 확률 값이다. 계산된 중요도는 연산 큐(priority queue)의 정렬 기준으로 사용되며, 중요도가 높은 큐브부터 우선적으로 연산이 할당된다. 이러한 연산은 각 단위큐브에 매핑된 GPU 스레드에서 병렬적으로 수행되며, 연산 결과

는 각 큐브의 메타데이터에 직접 갱신된다.

- [0080] 예를 들어, 중요도 기반 연산 결과를 각 큐브의 에너지 필드에 기록하거나, 연산 흐름의 가중치로 반영할 수 있다. 또한, 중심 단위큐브가 연산 트리거로 작동할 경우, 그 시점을 기준으로 전체 CUDA 쓰레드가 동기화되어 파동(wave) 형태로 연산이 확산되며, 이는 시공간 정보 흐름의 중심-외곽 전개 구조를 실현한다. 본 구조는 연산 병렬성, 반응 속도, 정보 중심성 보존을 동시에 만족하는 연산 모델로서, 자율비행체의 경로 탐색, 시공간 흐름 시뮬레이션 등 고속 연산 기반 응용에 적합하다.
- [0081] 또한, 중심 단위큐브는 경로 탐색 알고리즘의 초기값(initial seed) 설정 시에도 핵심적인 기준점이 된다. 예를 들어, A* 알고리즘 또는 Dijkstra 알고리즘에 있어 출발점이 되는 노드의 설정은 탐색 경로 전체에 큰 영향을 미치며, 이때 중심 큐브의 위치 및 메타데이터는 전체 경로 초기 조건을 결정짓는 핵심값으로 작용한다. 예를 들어, Q-Block 내 중심 단위큐브(좌표 (4,4,4))의 중첩 확률(overlap probability)이 0.92, 에너지 상태값(energy level)이 0.87, 그리고 TSU 기준 상대 시간 지연값이 최소($\Delta TSU = 0$)일 경우, 해당 큐브는 전체 Q-Block 내에서 가장 안정적이고 정보가 밀집된 중심점으로 판단된다. 이러한 상황에서 A* 알고리즘을 실행할 경우, 출발 노드(start node)는 이 중심 단위큐브로 설정되며, 이후의 탐색 노드는 메타데이터 유사도(similarity of vector), 정보 밀도(density), 거리(weighted distance) 등을 종합하여 $f(n) = g(n) + h(n)$ 기반으로 평가된다. 여기서;
- [0082] 가. $g(n)$: 중심 단위큐브로부터의 누적 연산 거리 (예: Euclidean distance 또는 TSU 누적값 기반),
- [0083] 나. $h(n)$: 현재 위치 큐브의 예측 잔여 경로 비용 (예: 메타데이터 유사도에 기반한 추정거리)
- [0084] 이와 같은 탐색은 중심 단위큐브의 연산 상태가 최적에 가깝다는 전제 하에, 경로 전체의 초기화 과정에서 높은 신뢰성과 예측 효율성을 보장하게 된다. 결과적으로, 본 발명에서는 중심 단위큐브가 단지 공간상의 중심에 그치지 않고, 병렬 연산 트리거이자 정보 전파의 중심축, 경로 탐색의 시발점 등 다층적인 연산 논리의 중심축으로 기능함을 통해, 고속 병렬 연산, 정보 흐름의 질서 있는 정렬, 그리고 자율 경로 탐색의 효율적 실행을 가능하게 하는 구조적 토대를 제공한다.
- [0085] (3) 메타데이터 집계 허브 : 중심 단위큐브(좌표 (4,4,4))는 Q-Block 내 512개 단위큐브 가운데 공간적으로 중앙에 위치할 뿐만 아니라, 정보 흐름의 중심성 판단을 위한 연산 기준점(reference hub)으로 기능한다. 본 발명에서 각 단위큐브는 고유 메타데이터를 포함하며, 그 항목에는 정보 밀도(density), 중첩 확률(overlap probability), 상태 벡터 유사도(state vector similarity), 에너지 레벨(energy level), TSU 기반 시간 누적값(time delta) 등이 포함된다. 중심 단위큐브는 이러한 메타데이터들을 주기적으로 집계하여 다음과 같은 역할을 수행한다;
- [0086] 가. 정보 중심성(weighted centrality) 계산 : 주변 단위큐브들의 메타데이터를 일정 반경 내에서 계층적으로 수집한 후, 가중치 기반 필터링 알고리즘을 통해 정보 중심성 스코어를 산출한다. 예컨대, 중심 단위큐브에서 반경 4 이하 거리 내 단위큐브들의 중요도를 계산하고, 이들로부터 연산 집중도 또는 흐름의 밀집 정도를 정량적으로 평가한다.
- [0087] 나. 연산 집중도 분석 및 병렬 연산 큐(priority queue) 구성 : 이렇게 집계된 중심성 정보는 Q-Block 내 연산 흐름의 방향 설정, 연산 순서 정렬, 연산 우선순위 배정 등에 활용된다. 특히 CUDA 기반 병렬 연산 환경에서는 중심 단위큐브가 가진 집계 결과를 기반으로, 각 단위큐브에 대한 threadIdx 배정 순서를 결정하거나, 특정 정보 흐름 경로 상의 병목 구간(bottleneck)을 사전에 파악하여 연산 최적화를 수행할 수 있다.
- [0088] 다. 정보 흐름의 논리적 중심 허브 : 중심 단위큐브는 단순히 데이터를 수신하는 정적 노드가 아니라, Q-Block 내부에서의 정보 흐름 재구성과 연산 흐름의 트리거링 기능을 수행하는 능동적 노드로서의 성격을 가진다. 집계된 메타데이터는 이후 경로 탐색 알고리즘(A*, Dijkstra 등), 텔레포테이션 가능성 평가, AI 기반 시공간 예측 모듈 등에 입력값으로 제공된다.

표 9

[0089]

중심 단위큐브 메타데이터 집계 및 중요도 기반 큐 구성 코드 예시

```
import math

# 단위큐브 클래스 정의
class UnitCube:
    def __init__(self, id, coord, metadata):
        self.id = id
        self.coord = coord # (x, y, z)
        self.metadata = metadata # dict: density, energy, overlap, tsu, vector
        self.importance = 0

# 중심 단위큐브 좌표 (4, 4, 4)
ANCHOR_COORD = (4, 4, 4)

# 메타데이터 가중치 테이블 정의
WEIGHTS = {
    "density": 0.25,
    "energy": 0.20,
    "overlap": 0.30,
    "tsu": 0.10,
    "vector": 0.15
}

# 유클리드 거리 계산 함수
def euclidean_distance(c1, c2):
    return math.sqrt(sum((a - b)**2 for a, b in zip(c1, c2)))

# 중요도 계산 함수
def compute_importance(cube):
    md = cube.metadata
    return (
```

```

        WEIGHTS["density"] * md.get("density", 0) +
        WEIGHTS["energy"] * md.get("energy", 0) +
        WEIGHTS["overlap"] * md.get("overlap", 0) +
        WEIGHTS["tsu"] * md.get("tsu", 0) +
        WEIGHTS["vector"] * md.get("vector", 0)
    )

# 병렬 연산 우선순위 큐 구성
def build_priority_queue(unit_cubes, anchor_coord, radius=4):
    filtered = []
    for cube in unit_cubes:
        dist = euclidean_distance(cube.coord, anchor_coord)
        if dist <= radius:
            cube.importance = compute_importance(cube)
            filtered.append((cube.importance, dist, cube))

    # 중요도 우선 정렬 (높은 중요도 → 낮은 거리 순)
    filtered.sort(key=lambda x: (-x[0], x[1])) # 중요도 내림차순, 거리 오름차순
    return [cube for _, _, cube in filtered]

# --- 예시 실행 ---

# Q-Block 내 단위큐브 512개 샘플 생성 (임의 데이터)
unit_cubes = [
    UnitCube(id=i, coord=(i % 8, (i // 8) % 8, i // 64),
        metadata={
            "density": i % 5,
            "energy": (i % 7) / 10,
            "overlap": (i % 9) / 10,
            "tsu": i % 4,
            "vector": (i % 6) / 10
        })
    for i in range(512)
]

# 중심 단위큐브 기준 병렬 연산 우선순위 큐 구성
priority_queue = build_priority_queue(unit_cubes, ANCHOR_COORD)

# 연산 태스크 할당 예시 출력
for idx, cube in enumerate(priority_queue[:10]):
    print(f"[Thread {idx}] Assigned Cube ID: {cube.id}, Importance Score: {cube.importance:.4f}")

```

[0090] 상기 코드는 중심 단위큐브 좌표를 기준으로 각 단위큐브의 중요도를 계산하고, 거리로 나누어 우선순위를 결정 한 뒤, 병렬 연산 큐에 배치하기 위한 기반 로직을 GPU에서 실행할 수 있게 구성한 것이다. 따라서 중심 단위큐 브는 단순한 중심 좌표 그 이상으로, Q-Block 내 연산 최적화와 정보 집중도 분석의 기준점으로 작용하며, 병렬 구조 하에서 정보 전파 및 연산 동기화의 기준 축을 형성한다. Q-Twin™ 시스템에서는 중심 단위큐브에 집계된 메타데이터를 기반으로, 각 TSU 단위에서의 정보 중심성을 실시간으로 평가하고, 이를 바탕으로 후속 트윈 적층 흐름(anchor-to-anchor trajectory) 및 예측 기반 시뮬레이션 흐름을 설계한다. 이 구조는 시간 흐름을 따라 누 적되는 디지털 트윈 레이어들 간의 정보 일관성을 확보하는 데 중요한 기준축으로 작용한다.

[0091] (4) GPU 스레드 동기화 중심 : 본 발명에서 정의하는 중심 단위큐브는 단순히 Q-Block 내의 연산 기준점 역할을 넘어, CUDA 기반 GPU 병렬 연산 환경에서 스레드 간 연산 타이밍과 흐름을 제어하는 동기화 허브 (synchronization hub)로 기능한다. 중심 단위큐브는 Q-Block의 정중앙 좌표인 (4,4,4)에 위치함으로써 물리적

중심성뿐 아니라 논리적 중심성도 함께 지니며, 이는 GPU 내에서 threadIdx나 blockIdx 등의 스레드 인덱스와 구조적으로 정렬될 수 있도록 설계된다.

[0092]

구체적으로, 중심 단위큐브가 보유한 에너지 값, 중첩 확률, 연산 이력 등 메타데이터 항목이 특정 임계값에 도달하게 되면, 해당 큐브는 인접 큐브들에 대해 연산 개시 신호(trigger signal)를 생성하며, 이는 CUDA의 __syncthreads(), __threadfence() 등의 내부 연산 제어 명령어들과 함께 활용될 수 있다. 이를 통해 중심 단위큐브는 병렬 연산 중 각 스레드가 특정 연산 단계로 진입하거나 종료하는 시점을 조절함으로써, 전체 병렬 프로세스의 데이터 정합성(data consistency)과 연산 타이밍 정확성을 확보하는 기준점이 된다. 또한, 복수의 Q-Block 간 anchor-to-anchor 통신이 필요한 경우, 중심 단위큐브는 해당 Q-Block의 대표값 또는 스케줄러 역할을 수행함으로써, 서로 다른 블록 간 연산 동기화 및 연결 우선순위 판단에 있어서도 중앙 제어 노드(central control node)로 작용할 수 있다. 예컨대, 중심 큐브가 연산 우선순위를 설정한 후, 해당 정보를 공유 메모리 또는 글로벌 메모리 상에 전달하고, 이를 다른 스레드 또는 블록에서 참조하여 연산 진입 조건을 동기화함으로써 다중 Q-Block 간 시공간 연산 흐름의 일관성을 유지하게 된다.

표 10

[0093]

중심 단위큐브 기반 GPU 스레드 동기화 및 연산 트리거 예시

```
// CUDA C++ 스타일 예시
__global__ void QBlockKernel(UnitCube* cubes, int totalCubes) {
    int tid = threadIdx.x + blockIdx.x * blockDim.x;
    if (tid >= totalCubes) return;

    UnitCube cube = cubes[tid];

    // 중심 단위큐브 조건
    if (cube.coord.x == 4 && cube.coord.y == 4 && cube.coord.z == 4) {
        // 중심 단위큐브의 임계값 평가
        if (cube.energy > ENERGY_THRESHOLD && cube.overlap > OVERLAP_THRESHOLD) {
            // 연산 시작 트리거 플래그를 공유 메모리에 설정
            __shared__ bool trigger;
            trigger = true;
        }
    }

    // 스레드 동기화
    __syncthreads(); // 중심 단위큐브의 trigger 정보 반영 후 진입

    // 트리거가 설정된 경우에만 연산 수행
    if (__syncthreads_or(trigger)){
        // 각 큐브별 연산 실행
        performComputation(cube);
    }
}
```

[0094]

중심 단위큐브는 에너지 값 및 중첩 확률이 사전에 정의된 임계값을 초과하는 경우, 내부적으로 연산 개시 신호에 해당하는 trigger = true 값을 설정하게 된다. 이후 __syncthreads() 명령어를 통해 GPU 내의 모든 스레드는 해당 중심 단위큐브의 상태를 기준으로 동기화되며, 이는 각 스레드가 연산에 진입하기 전 공통 조건을 점검하는 절차로 작용한다. 특히 __syncthreads_or(trigger) 구문을 통해 중심 큐브의 트리거 조건이 충족된 경우에만 전체 병렬 연산이 시작되도록 제어함으로써, 개별 단위큐브들이 무분별하게 연산을 시작하는 것을 방지하고, 연산 타이밍의 일관성을 유지할 수 있도록 한다. 이러한 설계는 중심 단위큐브의 상태를 전체 CUDA 스레드의 연산 개시 기준값으로 활용함으로써, 스레드 간 연산 시점 제어 및 타이밍 동기화를 가능하게 하며, 트리거 기반의 연산 흐름 통제를 구조적으로 구현하는 데 기여한다.

[0095] 나아가, 다수의 Q-Block이 동시에 병렬 연산을 수행하는 경우, 각 Q-Block의 중심 단위큐브는 해당 블록의 대표 값 또는 스케줄러 역할을 수행할 수 있으며, 중심-to-중심 연산(anchor-to-anchor computation) 구조를 통해 블록 간 연산 순서, 연결 우선순위, 동기화 타이밍 등을 실시간으로 조정할 수 있게 된다. 이는 전체 시스템 수준에서의 연산 흐름 정합성과 시공간 연산 네트워크의 예측 가능성을 크게 향상시키는 효과를 가져온다. 아울러, 이와 같은 구조는 단순한 병렬 연산 수행을 넘어, 시공간적 연산의 제어 흐름, 데이터 흐름, 정보 확산의 규칙성을 통제할 수 있는 고차원적 연산 구조를 제공한다는 점에서 기존의 GPU 기반 시뮬레이션 또는 연산 구조와 차별성을 가지며, 자율항법, 경로 탐색, 시공간 예측 연산 등의 다양한 고난도 응용 기술에 적용 가능하다. Q-Twin™ 구조에서 중심 단위큐브는 시점별 트윈 적층의 기준 축으로 작용하며, TSU 단위 연산 적층이 anchor-to-anchor 기반으로 연결될 때, 각 anchor는 시뮬레이션 및 현실 센서 데이터 간의 동기화 기준점으로 기능하게 된다. 따라서 중심 단위큐브를 통한 CUDA 연산 동기화는 단순 연산 타이밍 제어를 넘어, 디지털 트윈의 시간 정합성 유지에 기여하는 핵심 구조로 확장된다.

[0096] (5) 향후 연산 시드(anchor-to-anchor transfer seed) : 중심 단위큐브는 단일 Q-Block 내부에서의 병렬 연산 중심일 뿐만 아니라, 다수의 Q-Block들이 연속적으로 구성된 시공간 상에서 연산 흐름의 연결지점(transfer point) 역할도 수행한다. 특히, 중심 단위큐브는 Q-Block 전체의 연산 결과를 대표할 수 있는 핵심 정보를 집약적으로 보유하고 있기 때문에, 후속 Q-Block이 생성되거나 연산을 시작할 때 그 중심 단위큐브의 메타데이터 - 예컨대 정보 밀도(density), 상태 벡터(vector), 중첩 확률(overlap), 에너지 상태(energy), 생성 시점(TSU) 등 - 는 다음 연산 흐름의 기준점이자 초기값(seed)으로 활용된다.

[0097] 이러한 구조는 마치 anchor-to-anchor 방식의 연산 전이 구조를 형성하는 것으로, 각 Q-Block의 중심 단위큐브(anchor)가 서로 연산적으로 연결되는 연산 체인(logical chain) 또는 연산 루트(logical route)를 구성한다. 예를 들어, 자율비행체(UAM)나 가상공간 내 이동체가 다수 Q-Block을 순차적으로 경유하며 이동할 경우, 현재 Q-Block의 중심 단위큐브에서 수집된 메타데이터는 다음 Q-Block의 연산 방향, 초기 경로 탐색 조건, 텔레포트 가능성 평가 등에 핵심 입력값으로 작용하게 된다. 또한, 이러한 anchor-to-anchor 연산 연동 구조는 강화학습 기반 경로 탐색 정책(policy optimization using RL), 장기 연산 흐름 예측(long-range spatiotemporal inference), 멀티 Q-Block 경로 스케줄링(scheduling) 등 고차원 시공간 AI 연산에도 활용될 수 있다. 중심 단위큐브는 단순히 현재 Q-Block의 연산 허브를 넘어서, 시공간 연산 흐름의 추적·전이·예측 구조의 기준점이 되는 셈이다.

표 11

[0098] anchor-to-anchor 경로 탐색 시드값 전이 예시

```
# 현재 Q-Block의 중심 단위큐브에서 추출한 시드값
anchor_cube_current = get_anchor_cube(QBlock_current)
seed_vector = anchor_cube_current.metadata["vector"]
seed_energy = anchor_cube_current.metadata["energy"]

# 다음 Q-Block의 경로 탐색 초기값으로 설정
anchor_cube_next = get_anchor_cube(QBlock_next)
anchor_cube_next.set_initial_vector(seed_vector)
anchor_cube_next.set_energy_seed(seed_energy)

# 이를 기반으로 최적 경로 탐색 알고리즘 초기화
path = a_star_pathfinder(start=anchor_cube_next, goal=target_position)
```

[0099] 상기 예시는 QBlock_current에서 병렬 연산이 완료된 이후, 해당 Q-Block의 중심 단위큐브(anchor cube)가 보유한 주요 메타데이터, 특히 상태 벡터(state vector) 및 에너지 값(energy level)과 같은 핵심 정보가, 다음 시점에 생성되는 후속 Q-Block의 중심 단위큐브(anchor cube)로 전달되어 초기화 값(seed)으로 활용되는 구조를 보여준다. 이러한 전이 구조를 통해 후속 Q-Block은 독립적인 연산 블록으로 작동하는 것이 아니라, 이전 Q-Block의 연산 결과 및 정보 흐름을 기반으로 하여 경로 탐색 로직이나 연산 정책을 연속적으로 이어받을 수 있게 되며, 이는 Q-Block 간 anchor-to-anchor 기반 연산 연결 메커니즘을 실제로 구현한 구체적인 예에 해당한다. 이러한 연산 시드 전이 방식은 시공간적 연산 연속성을 보장하고, AI 정책의 시간적 전이 학습 또는

예측 기반 시뮬레이션의 정밀도 향상에 기여할 수 있다.

표 12

[0100]

강화학습 기반 anchor-to-anchor 전이 학습 구조 예시

```
# 현재 Q-Block의 anchor cube 정보
anchor_state = get_anchor_cube(QBlock_current).get_state_vector()

# 이전 anchor-to-anchor 전이 이력 기반 보상 평가
reward = compute_reward(anchor_state, previous_anchor_state)

# 강화학습 에이전트에 학습 피드백 제공
agent.learn(state=previous_anchor_state, next_state=anchor_state, reward=reward)
```

[0101]

상기 구조는 anchor-to-anchor 기반의 연산 전이 흐름이 강화학습(RL, Reinforcement Learning) 프레임워크 내에서 주요 구성 요소인 상태(state), 행동(action), 보상(reward)으로 체계적으로 매핑될 수 있음을 보여준다. 구체적으로, 각 중심 단위큐브(anchor cube)는 Q-Block 내 또는 Q-Block 간 시공간 전이 흐름에서 하나의 독립된 상태 노드(state node)로 정의되며, 이 상태는 해당 큐브가 보유한 메타데이터(예: energy, density, overlap, vector 등)를 통합한 고차원 벡터로 표현된다. 이후, 강화학습 에이전트는 이 상태를 기반으로 특정 연산 경로 선택, 큐브 활성화, 정보 확산 방향 결정 등과 같은 행동(action)을 선택하게 되며, 해당 행동이 전체 연산 최적화 또는 경로 효율성 향상에 얼마나 기여했는지를 보상(reward)으로 측정하게 된다. 이와 같은 구조는 시간적으로 누적되는 Q-Block 간 연산 흐름을 학습 가능한 시계열 데이터로 구조화할 수 있게 하며, AI가 장기적 정보 흐름 패턴을 인식하고, 미래 연산 경로 또는 최적 연산 정책을 예측·계획할 수 있도록 한다. 결과적으로, 중심 단위큐브를 기반으로 한 anchor-to-anchor 전이 구조는 단순한 데이터 흐름을 넘어서, 강화학습 기반 AI 최적화의 핵심 기반으로 기능할 수 있는 연산 시공간 모델을 제공한다.

표 13

[0102]

Q-Block 간 연산 흐름 지도(spatiotemporal flow map) 예시

```
flow_map = []
for i in range(N):
    anchor = get_anchor_cube(QBlocks[i])
    flow_map.append({
        "tsu": anchor.tsu,
        "density": anchor.metadata["density"],
        "vector": anchor.metadata["vector"]
    })

# 전체 anchor 흐름 시각화
plot_spacetime_flow(flow_map)
```

[0103]

상기 예시는 복수 Q-Block의 anchor cube 정보를 수집하여 시공간 정보 흐름 지도(flow map)를 생성하는 예이다. 이 흐름은 AI 기반 시공간 예측, 경로 최적화, 이상 탐지 등에 활용된다. 이와 같은 예시는 단일 Q-Block 내부에서의 병렬 연산 흐름에 국한되지 않고, 복수의 Q-Block 간 연산 연결 및 정보 전이 구조에 중점을 둔다는 점에서 기술적으로 중요한 차별성을 갖는다. 특히, 각 Q-Block의 중심 단위큐브(anchor cube) 간 연산 전이를 일관된 시드(seed) 흐름으로 모델링함으로써, 시간적·공간적 맥락을 유지한 채 연속적인 연산이 가능하게 되며, 이는 시공간 연산 체계의 연속성(spatiotemporal continuity)과 더불어 고차원 연산 예측(high-dimensional computation forecasting)의 실현 가능성을 크게 향상시킨다.

표 14

[0104]

Q-Twin™ 시스템 내 중심 단위큐브 기반 연산 제어 및 최적화 기능 요약

항목	설명	기술 효과
병렬 연산 트리거 조건	중심 단위큐브의 에너지 또는 중첩 확률이 임계값 초과 시 연산 트리거 발생	불필요한 연산 방지 및 병렬 큐 효율
연산 흐름 확산 모델	중심에서 외곽으로 퍼지는 파동처럼 연산 흐름이 전개됨	정보 흐름 기반 연산 우선순위 구성 가능
경로 탐색 초기값 설정	중심 단위큐브를 A* 또는 Dijkstra 알고리즘의 초기 노드로 사용	경로 탐색의 정확도 및 연산 효율 증대
메타데이터 기반 정보	중심 단위큐브가 주변 큐브들의 메타데이터를 집계하고 가중치 평가	정보 집중도 평가 및 병렬 큐 최적화
스레드 동기화 및 연산	중심 단위큐브의 상태를 기준으로 __syncthreads(), __syncthreads_or()를 통해 연산 타이밍 조절	GPU 스레드 간 연산 일관성
Q-Block 간 연산 시드	Q-Block 간 연산 흐름을 중심 단위큐브의 메타데이터를 통해 anchor-to-anchor 방식으로 연결	다중 Q-Block 연산의 연속성과 예측 가능성

[0105]

이러한 기능들은 Q-Block 구조의 병렬 연산 효율성을 높이는 동시에, 단일 블록 내에서의 정보 정렬 및 확산뿐만 아니라, 복수의 Q-Block 간 연산 연결성 확보에도 핵심적인 역할을 수행한다. 특히, 실시간 시뮬레이션, 텔레포테이션 연산, AI 기반 경로 탐색, 예측 모델링 등 고속·고정밀 연산이 요구되는 응용 기술에 있어서, 중심 단위큐브는 연산 흐름의 일관성, 트리거 조건의 명확성, 시드 기반 정보 전이의 안정성을 보장하는 기준점으로 작용함으로써 전체 시스템의 예측 가능성과 확장성을 함께 확보할 수 있게 한다. 특히 Q-Twin™ 시스템은 중심 단위큐브를 통해 현실 시점 기반 디지털 트윈 연산의 anchor-to-anchor 흐름을 구성하며, 이는 트윈 상태의 시점별 추적, 과거 상태 복원, 미래 경로 예측 등의 기능을 자연스럽게 연결짓는 핵심 구조로 기능한다. 이처럼 중심 단위큐브의 anchor-to-anchor 전이는 Q-Twin™의 시계열 디지털 트윈 시스템 구현에 있어 구조적 정합성과 예측 정밀도를 동시에 보장하는 중핵 기술 요소이다.

[0106]

3. 중심 단위큐브 중심 병렬 연산 순서 정렬 로직

[0107]

본 발명의 Q-Twin™ 시스템에서는, 디지털 트윈의 단위 연산 블록인 Q-Block 내 512개의 단위큐브가 중심 단위큐브(anchor cube, 좌표 (4,4,4))를 기준으로 병렬 연산 큐에 정렬된다. 이 연산 정렬 로직은, 현실 공간으로부터 수신된 센서 정보가 각 단위큐브에 매핑된 이후, 중심 단위큐브를 기준으로 연산 우선순위를 재조정하고, 그 결과를 실시간 디지털 트윈 갱신 시퀀스로 변환하는 데 활용된다. 특히 본 발명은 (1) 거리 기반 정렬과 (2) 메타데이터 가중치 기반 중요도 산정을 결합하여, 단위큐브별 정보 중심성과 공간적 중심성에 따른 연산 우선순위 큐를 구성함으로써, 실시간 트윈 상태 반영의 정밀도와 속도를 동시에 확보할 수 있다.

[0108]

(1) 거리 기반 기본 우선순위 계산

[0109]

각 단위큐브 i 는 중심 단위큐브와의 3차원 거리(예: Euclidean 거리 또는 Manhattan 거리)를 기준으로 기본적인 연산 우선순위를 부여받는다. 각 단위큐브와 중심 단위큐브 간의 거리 값은, 예를 들어 중심 단위큐브의 좌표가 (4, 4, 4)인 경우, 다음과 같은 유클리드 거리(Euclidean distance) 산식에 따라 산정될 수 있다;

[0110]

$$\text{distance}_i = \sqrt{((x-4)^2 + (y-4)^2 + (z-4)^2)}$$

[0111]

상기 식에서 x_i, y_i, z_i 는 각각 단위큐브 i 의 x, y, z 좌표를 의미하며, 산출된 거리 값은 연산 우선순위 정렬의 기준값으로 사용된다. 해당 거리 값은 이후 단계에서 메타데이터 기반 중요도 점수와 함께 병렬 연산 큐(priority queue) 구성을 위한 가중치로 적용된다. 이때 중심 단위큐브는 거리 0으로 가장 높은 연산 우선순위를 가지며, 거리값이 증가할수록 상대적으로 낮은 우선순위가 부여된다. 다음 코드는 연산 우선순위 계산의 기본 골격만 다루며, 이후 중요도 점수(importance score) 연산과는 구별된다;

표 15

[0112]

Q-Twin™ 연산 흐름 정렬을 위한 거리 기반 우선순위 산정 알고리즘 예시

```
# 중심 단위큐브 좌표
ANCHOR_COORD = (4, 4, 4)

# 단위큐브 클래스 정의
class UnitCube:
    def __init__(self, id, x, y, z):
        self.id = id
        self.coord = (x, y, z)
        self.distance = 0 # 중심 큐브로부터의 거리
        self.priority = 0 # 연산 우선순위 (낮을수록 우선)

# 거리 계산 함수 (Euclidean 거리)
def compute_distance(cube, anchor_coord=ANCHOR_COORD):
    dx = cube.coord[0] - anchor_coord[0]
    dy = cube.coord[1] - anchor_coord[1]
    dz = cube.coord[2] - anchor_coord[2]
    return (dx**2 + dy**2 + dz**2)**0.5

# Q-Block 내 단위큐브 512개를 초기화했다고 가정
unit_cubes = [UnitCube(i, x, y, z)
               for i, (x, y, z) in enumerate(generate_cube_coordinates(8, 8, 8))]

# 중심 큐브 기준 거리 계산 및 우선순위 부여
for cube in unit_cubes:
    cube.distance = compute_distance(cube)
    cube.priority = cube.distance # 초기 우선순위는 거리값 그대로

# 거리 기준 정렬 (연산 큐 구성)
priority_queue = sorted(unit_cubes, key=lambda c: c.priority)
```

[0113]

기술적 관점에서, 상기 정렬 로직은 디지털 트윈의 연산 흐름을 시간-공간 기반으로 정렬하기 위한 첫 단계로, 중심 단위큐브로부터의 상대 거리 정보를 활용하여 단위큐브별 초기 연산 우선순위를 결정하는 방식이다. 이 정렬 구조는 이후 단계에서 적용되는 가중치 기반 중요도 점수(importance score) 계산과 결합되어, 정보 중심성과 공간 중심성을 동시에 고려하는 복합 정렬 구조로 확장된다. 특히 중심 단위큐브는 좌표상 정중앙(4,4,4)에 위치하며, 거리 계산 결과 0을 가지므로 별도의 보정 없이도 트윈 연산 흐름의 출발점이자 시뮬레이션 기준점으로 자연스럽게 지정된다. 이와 같은 설계는 트윈의 상태 변화 전파, 연산 타이밍 제어, 병렬 연산 큐 구성에 있어 정보 정렬의 일관성과 시간 동기화의 정밀성을 동시에 확보할 수 있게 한다.

표 16

[0114]

Q-Twin™ 중심 단위큐브 기준 CUDA 병렬 연산 우선순위 큐 구성 코드 예시

```
struct UnitCube {
    int id;
    int x, y, z;
    float priority;
};

// 중심 단위큐브 좌표 고정
#define CENTER_X 4
#define CENTER_Y 4
#define CENTER_Z 4

// 거리 기반 우선순위 계산 커널
__global__ void computeDistancePriority(UnitCube* cubes, int totalCubes) {
    int tid = threadIdx.x + blockIdx.x * blockDim.x;
    if (tid >= totalCubes) return;

    UnitCube* cube = &cubes[tid];

    int dx = cube->x - CENTER_X;
    int dy = cube->y - CENTER_Y;
    int dz = cube->z - CENTER_Z;

    float distance = sqrtf((float)(dx*dx + dy*dy + dz*dz));

    // 거리 값을 연산 우선순위로 저장 (작을수록 우선)
    cube->priority = distance;
}
```

[0115]

단위큐브(UnitCube) 구조체는 Q-Block 내 하나의 단위큐브를 디지털 객체로 표현한 데이터 단위이며, 고유 ID, 3차원 위치 좌표(x, y, z), 그리고 중심 단위큐브를 기준으로 산출된 연산 우선순위(priority) 값을 포함한다. 본 발명의 CUDA 기반 병렬 연산 환경에서 사용되는 computeDistancePriority 커널 함수는, 각 GPU 스레드가 할당받은 단위큐브의 좌표와 Q-Block의 중심 좌표(CENTER_X, CENTER_Y, CENTER_Z)를 비교하여 유클리드 거리(Euclidean distance)를 계산하고, 이를 해당 단위큐브의 초기 연산 우선순위로 지정하는 방식으로 구성된다. 이렇게 산출된 거리 기반 priority 값은, 이후 병렬 연산 큐(priority queue)를 구성하는 핵심 기준으로 작용하며, 연산 흐름 상의 정보 정렬 및 연산 타이밍 제어에 실질적인 영향을 미친다. 특히, 거리값이 작을수록 중심 단위큐브에 가까운 것으로 판단되므로, 해당 큐브는 디지털 트윈 시뮬레이션 시, 변화 상태 전파의 선두 노드로 지정될 수 있으며, GPU 병렬 환경에서는 이를 기준으로 스레드 간 병렬 연산 흐름의 공간적 일관성을 유지할 수 있다. 이와 같은 구조는 단위큐브 간의 공간적 연산 정렬을 가능케 하며, Q-Twin™ 시스템이 요구하는 정밀한 시공간 연산 흐름 제어 및 실시간 시뮬레이션 타이밍 최적화에 핵심적인 기반이 된다.

[0116]

(2) 가중치 기반 중요도 점수 계산 : Q-Twin™ 시스템은 단위큐브별 거리 기반 연산 정렬 외에도, 큐브가 보유한 메타데이터 항목들의 정보 가치를 종합적으로 고려하여 연산 우선순위를 정밀하게 조정한다. 이는 디지털 트윈 시뮬레이션의 정확도 및 자원 배분 최적화를 위한 핵심 기반으로 작용한다. 연산 중요도(importance score) 산출을 위해 각 단위큐브는 다음과 같은 메타데이터 요소를 포함하며, 이는 사전에 정의된 가중치 집합($W = \{w_d, w_e, w_o, w_t, w_v\}$)과 함께 연산된다;

표 17

[0117]

Q-Twin™ 단위큐브 중요도 점수 계산을 위한 메타데이터 구성 요소 요약

항목	내용
density_i	해당 큐브 내 정보 밀도
energy_i	연산 활성도 또는 에너지 수준

overlap_i	인접 큐브와의 중첩 확률
tsu_i	중심 큐브로부터의 시간 거리 (TSU 차이)
vector_i	상태 벡터의 크기 또는 방향성 정보

[0118] 이로부터 계산되는 중요도 점수는 다음 수식으로 표현된다;

[0119]
$$\text{importance}_i = w_d \cdot \text{density}_i + w_e \cdot \text{energy}_i + w_o \cdot \text{overlap}_i + w_t \cdot \text{tsu}_i + w_v \cdot \text{vector}_i$$

[0120] 이렇게 산출된 importance score는 디지털 트윈 환경에서 시뮬레이션의 우선 처리 방향, 연산 집중도 분석, 병렬 큐 정렬, 시각화 흐름 순서 등에 직접 적용되며, 특히 AI 기반 트윈 예측 모듈에서 강화학습 알고리즘의 초기 가중치로도 활용된다. 예를 들어, 에너지 밀도나 정보 중첩도가 높은 큐브는 시뮬레이션 중 상태 변화가 활발하게 일어날 가능성이 높아 우선 연산 대상으로 설정되며, 이는 GPU 스레드에서의 효율적인 연산 태스크 분배를 가능하게 한다. 특히 Q-Twin™에서는 이러한 중요도 점수가 향후 생성될 Q-Block의 anchor-to-anchor 시드값으로도 연계되므로, 트윈 적층 연산의 흐름성과 연속성 유지에도 중요한 역할을 수행한다. 또한, CUDA 환경에서의 연산 예시는 단위큐브의 중요도를 계산하고 이를 기반으로 연산 스케줄링을 정렬하여, 정보 가치가 높은 단위부터 연산을 실행하도록 구성되어 있다. 이는 다음 항목에서 구체적으로 코드와 함께 제시된다.

[0121] (3) 연산 스케줄링 흐름: Q-Twin™ 시스템은 각 단위큐브의 정보 밀도, 에너지 상태, 중첩 확률, TSU 기반 시간 거리, 상태 벡터 크기 등의 메타데이터를 기반으로 중요도를 산출하고, 이를 기준으로 병렬 연산 큐를 구성하여 GPU 스레드에 우선 배정한다. 이 흐름은 시간 누적형 디지털 트윈 시뮬레이션의 정확도와 실시간성 확보에 핵심적인 역할을 한다.

표 18

[0122] Q-Twin™ 단위큐브 구조 및 CUDA 기반 중요도 점수 계산 로직

```
// Q-Twin™ 시뮬레이션용 단위큐브 구조
struct UnitCube {
    int id;
    float3 coord;           // 단위큐브의 좌표 (x, y, z)
    float density;          // 정보 밀도
    float energy;           // 에너지 상태
    float overlap;          // 중첩 확률
    float tsu;              // TSU 기준 시간 거리
    float vectorMagnitude;  // 상태 벡터 크기
    float importance;       // 연산 중요도 점수
};

// 가중치 정의 (Q-Twin 시뮬레이션 기준)
__constant__ float wd = 0.25f; // 정보 밀도
__constant__ float we = 0.20f; // 에너지
__constant__ float wo = 0.25f; // 중첩 확률
__constant__ float wt = 0.15f; // 시간 거리
__constant__ float wv = 0.15f; // 상태 벡터

// 중요도 점수 계산 (device function)
__device__ float compute_importance(UnitCube cube) {
    return (
        wd * cube.density +
        we * cube.energy +
        wo * cube.overlap +
        wt * cube.tsu +
        wv * cube.vectorMagnitude
    );
}
```

[0123] 상기 표는 Q-Twin™ 시스템에서 각 단위큐브(UnitCube)가 보유한 메타데이터를 바탕으로 연산 중요도(importance)를 계산하는 CUDA 구조체 및 디바이스 함수의 구현 예시를 제시한 것이다. 정의된 가중치 값들은 정보 밀도, 에너지 상태, 중첩 확률, TSU 기반 시간 거리, 상태 벡터 크기 등의 요소별 연산 영향도를 반영하며, 이 중요도 값은 후속 정렬, 병렬 연산 큐 구성, AI 정책 입력 등에서 핵심 기준으로 활용된다.

표 19

[0124] Q-Twin™ 기반 중요도 점수 연산 및 스케줄링 전체 흐름 예시

```
// 커널: 중요도 점수 계산
__global__ void ComputeImportance(UnitCube* cubes, int totalCubes) {
    int tid = threadIdx.x + blockIdx.x * blockDim.x;
    if (tid >= totalCubes) return;
    cubes[tid].importance = compute_importance(cubes[tid]);
}

// 연산 스케줄링 (host function)
void ScheduleAndRun(UnitCube* d_cubes, int totalCubes) {
    const int threadsPerBlock = 256;
    int blocks = (totalCubes + threadsPerBlock - 1) / threadsPerBlock;

    // Step 1: 중요도 점수 계산 (GPU)
    ComputeImportance<<<blocks, threadsPerBlock>>>(d_cubes, totalCubes);
    cudaDeviceSynchronize();

    // Step 2: 중요도 기준 정렬을 위해 GPU → CPU 복사
    UnitCube* h_cubes = new UnitCube[totalCubes];
    cudaMemcpy(h_cubes, d_cubes, totalCubes * sizeof(UnitCube), cudaMemcpyDeviceToHost);

    // Step 3: CPU에서 중요도 내림차순 정렬
    std::sort(h_cubes, h_cubes + totalCubes, [](UnitCube a, UnitCube b) {
        return a.importance > b.importance;
    });

    // Step 4: 정렬 결과를 다시 GPU로 전송
    cudaMemcpy(d_cubes, h_cubes, totalCubes * sizeof(UnitCube), cudaMemcpyHostToDevice);
    delete[] h_cubes;

    // Step 5: 연산 커널 실행 (정렬 반영)
    LaunchComputation<<<blocks, threadsPerBlock>>>(d_cubes, totalCubes);
}

// 실제 연산 실행 (시뮬레이션 상태 갱신)
__global__ void LaunchComputation(UnitCube* cubes, int totalCubes) {
    int tid = threadIdx.x + blockIdx.x * blockDim.x;
    if (tid >= totalCubes) return;

    UnitCube& cube = cubes[tid];

    // 중요도 기반 가중치 적용 상태 갱신 예시
    cube.energy += 0.05f * cube.importance; // 에너지 필드에 누적
    cube.vectorMagnitude *= 1.01f;          // 상태 벡터 약간 증폭
}
```

[0125] 상기 예시는 Q-Twin™ 시스템에서 각 단위큐브의 중요도를 계산하고, 이를 기반으로 연산 순서를 정렬한 후, GPU 상에서 실시간으로 연산을 수행하는 전체 스케줄링 흐름을 보여준다. 이러한 구성은 정보 밀도 및 상태 특성이 높은 큐브부터 우선적으로 연산 리소스를 배정함으로써 연산 효율성을 극대화하며, 시뮬레이션 결과의 정밀도와 동적 반응성을 동시에 향상시킨다.

표 20

[0126] Q-Twin™ 기반 병렬 연산 구조의 기술적 효과 요약

구분	설명
중요도 계산 기준	현실 센서 기반 정보 밀도, 에너지, TSU 시간 거리 등 디지털 트윈에서 의미 있는 메타데이터를 반영
정렬 구조	병렬 연산 전에 GPU ↔ CPU 간 동기화 및 정렬 과정 포함 (GPU 내 Thrust 정렬로 대체 가능)
우선 연산 큐 구성	중요도 점수가 높은 단위큐브부터 연산 큐에 배치되어, 변화 중심 영역 먼저 시뮬레이션 반영
시뮬레이션 확장성	상태 벡터 및 에너지 필드 갱신은 후속 Q-Block 적층 또는 anchor-to-anchor 연결 시 seed 값으로 전이됨
연산 전략의 특성	Q-Twin은 단위큐브 간 상호작용이 아닌, 정보 중심 연산 기반의 적층 흐름이라는 점에서 기존 pathfinder와 차별화됨

[0127] 상기 표는 Q-Twin™ 시스템이 단위큐브의 메타데이터 기반 중요도 정렬을 통해 연산 흐름을 구성하고, 변화가 집중되는 영역부터 연산을 우선 수행함으로써, 시뮬레이션의 정밀도와 반응 속도를 동시에 확보하는 기술적 특징을 요약한 것이다. 특히, 정렬된 연산 결과는 후속 Q-Block의 anchor-to-anchor 전이 구조에도 직접적으로 영향을 미치며, 실시간 디지털 트윈 환경에서 정보 중심의 연산 확산과 시뮬레이션 예측의 연속성을 효과적으로 구현할 수 있는 기반을 제공한다.

[0128] 도 10은 본 발명에 따른 Q-Twin™ 시스템에서, TSU(Time Singularity Unit) 단위로 연속 생성된 Q-Block 간 중심 단위큐브(anchor cube)의 메타데이터(정보 밀도, 에너지 상태, 중첩 확률 등)를 시계열 흐름으로 시각화한 구조를 나타낸다. 본 시각화는 Q-Twin™이 구현하는 시간 적응형 디지털 트윈 구조 내에서 anchor-to-anchor 기반 연산 흐름이 시간축을 따라 어떻게 전이되며, AI 기반 연산 예측, 경로 탐색, 연산 우선순위 판단 등에 필요한 메타데이터가 어떤 방식으로 누적·활용되는지를 정량적으로 보여준다. 이 흐름 구조는 이후의 강화학습 기반 탐색 시드 구성, 다중 Q-Block 연산 연결 및 정책 최적화 등에 있어 핵심 해석 자료로 기능한다. 이러한 시계열 흐름 분석을 통해 중심 단위큐브의 메타데이터가 시간 축을 따라 어떻게 변화하고, 각 지점에서 어떤 의미를 가지는지를 구조화하면 다음과 같은 해석이 가능하다;

표 21

[0129] Q-Twin™ 시계열 메타데이터 흐름 해석 항목 및 적용 효과 요약

항목	설명	Q-Twin™ 적용 해석
X축(TSU)	시간 특이점 단위로 정렬된 시점	디지털 트윈의 시간적 적층 구조를 반영
Y축(Normalized Value)	각 항목을 0~1 사이 정규화한 값	메타데이터의 비교 및 예측 기반화
정보 밀도(Density)	시점별로 축적된 정보량	고밀도 영역 탐지 및 위험 예측에 활용
에너지 상태(Energy)	정보 활성화도 또는 연산 역량 수준	경로 결정 또는 텔레포트 가능성 평가에 반영
중첩 확률(Overlap)	인접 Q-Block과의 정보 중복도	데이터 흐름 최적화 및 AI 정책 seed에 활용

[0130] 이러한 시계열 시각화는 Q-Twin™ 시스템의 핵심 개념인 "시간 축 기반의 디지털 적층 흐름"을 실제 메타데이터 변동 패턴으로 표현한 것으로, 각 TSU 단위에서 중심 단위큐브(anchor cube)에 축적된 정보를 기반으로 연산 흐름을 예측하거나 AI 탐색 정책을 유도할 수 있는 근거를 제공한다. 도 10의 그래프에서, 정보 밀도(Density)가 급증하는 구간은 외부 센서 이벤트나 사용자 입력 등 현실 계층 변화가 발생한 시점을 반영하며, 에너지 상태

(Energy)의 변화는 해당 시점의 연산 집중도 혹은 연산 필요성이 증가했음을 나타낸다. 중첩 확률(Overlap)이 높은 구간은 과거 Q-Block들과의 정보 중복 또는 상태 유사도가 높은 지점으로, 데이터 흐름 병목이나 예측 오류 가능성을 사전에 경고하는 지표가 된다. 따라서 도 10은 단순한 시각화 이상으로, Q-Twin™이 시간에 따라 어떻게 정보를 축적하고, 이를 기준으로 어떤 시점에서 경로를 전이(anchor-to-anchor transfer)하거나, AI가 어떤 패턴을 기준으로 정책(policy)을 설정해야 하는지를 판단하는 중요한 시뮬레이션 레퍼런스로 기능한다. 특히 이 메타데이터 시계열은 후속 Q-Block의 초기 상태 설정, 연산 큐 우선순위 배정, 또는 강화학습 기반 연산 정책 최적화에 직접 활용될 수 있는 전략적 기반 자료로 해석된다.

표 22

[0131]

Q-Twin™ 시스템의 Anchor 시계열 메타데이터 시각화(Matplotlib 기반)

```
# [도 10]Q-Twin™ 시스템 - TSU 단위 anchor 메타데이터 시계열 시각화 예시
import matplotlib.pyplot as plt

# TSU 단위 (시간 흐름)
tsu = list(range(10))

# 각 메타데이터 예시 값 (정규화된 값)
density = [0.2, 0.3, 0.5, 0.65, 0.7, 0.6, 0.75, 0.8, 0.85, 0.9]
energy = [0.1, 0.2, 0.35, 0.5, 0.55, 0.5, 0.6, 0.7, 0.78, 0.8]
overlap = [0.05, 0.1, 0.12, 0.15, 0.2, 0.25, 0.28, 0.3, 0.35, 0.4]

# 그래프
plt.plot(tsu, density, label='Density', marker='o')
plt.plot(tsu, energy, label='Energy', marker='s')
plt.plot(tsu, overlap, label='Overlap', marker='^')

plt.title("Q-Twin™ Anchor Metadata over TSU")
plt.xlabel("TSU (Time Singularity Unit)")
plt.ylabel("Normalized Metadata Value")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

[0132]

도 11은 Q-Twin™ 시스템의 anchor-to-anchor 기반 시공간 연산 예측 흐름을 나타낸 것으로, 각 TSU(Time Singularity Unit)마다 생성된 중심 단위큐브(anchor cube)의 메타데이터 값—정보 밀도(density), 에너지 상태(energy), 중첩 확률(overlap)의 시계열 변화—를 시뮬레이션한 결과를 시각화한 것이다. 이를 통해 다음과 같은 분석이 가능해진다. 첫째, 정보 밀도와 중첩 확률 곡선이 급격히 상승하는 구간은 연산 밀도 집중 영역으로 판단되며, 이는 Q-Block 적층 방향에서의 연산 집중 가능성을 예측하는 데 활용된다. 둘째, 에너지 상태 값이 지속적으로 증가하는 구간은 anchor-to-anchor 전이 경로의 초기값으로 설정될 수 있어, 연산 방향 결정에 중요한 기준이 된다. 셋째, 중첩 확률(overlap) 값이 0.7 이상으로 급등하는 시점은 연산 충돌 위험이 높은 TSU로 간주되어, 미래 시점에서의 경로 충돌 가능성을 조기에 탐지하고 회피 전략을 수립할 수 있다. 이러한 anchor-to-anchor 시뮬레이션 흐름 구조는 Q-Twin™의 시공간 디지털 트윈 환경에서 연산 흐름의 예측성과 확장성을 제공하는 핵심 기반이 되며, AI 기반 강화학습 연산 정책(seed policy)의 초기화, 경로 시뮬레이션, 텔레포테이션 가능성 평가 등과도 통합적으로 연계되어 고차원적 연산 최적화를 지원한다.

[0133]

이러한 중요도 기반 병렬 연산 큐 구성 방식은 GPU 내에서 실시간으로 연산 우선순위를 정렬할 수 있어, 병렬 연산 흐름의 타이밍 정합성과 정보 중심 연산 전략의 실행 가능성을 높인다. 아래는 Q-Twin™ 시스템 내에서 각 단위큐브의 중요도(importance)를 기준으로 GPU 디바이스 메모리 상에서 직접 정렬하는 CUDA Thrust 기반 구현 예시이다.

표 23

[0134]

Thrust 기반 Q-Twin™ 중요도 정렬 구조(GPU 디바이스 메모리 상 직접 정렬)

```

#include <thrust/device_vector.h>
#include <thrust/sort.h>
#include <thrust/functional.h>
#include <thrust/tuple.h>

// Q-Twin™ 단위큐브 구조
struct UnitCube {
    float importance; // 가중치 기반 중요도 점수
    float3 coord;     // 큐브 좌표 (x, y, z)
    float tsu;        // TSU 정보
    // ... 기타 메타데이터 생략
};

// 중요도 기준 비교 함수
struct compare_by_importance {
    __host__ __device__
    bool operator()(const UnitCube& a, const UnitCube& b) {
        return a.importance > b.importance; // 중요도 높은 순
    }
};

void thrust_sort_cubes(UnitCube* d_cubes, int totalCubes) {
    // Thrust 디바이스 포인터를 이용한 정렬
    thrust::device_ptr<UnitCube> dev_ptr = thrust::device_pointer_cast(d_cubes);

    thrust::sort(
        dev_ptr,
        dev_ptr + totalCubes,
        compare_by_importance()
    );
}

```

[0135]

상기 예시는 CPU ↔ GPU 간 불필요한 메모리 이동 없이, GPU 상에서 직접 중요도 정렬을 수행함으로써, 대규모 Q-Block 병렬 연산 환경에서 효율적인 연산 큐 구성이 가능함을 보여준다. 특히 Q-Twin™의 anchor-to-anchor 연산 흐름이나 연산 밀도 기반 경로 탐색 시뮬레이션에서 이와 같은 구조는 실시간 연산 결정 및 스레드 우선 배정의 핵심 기반이 된다.

[0136]

상기 Q-Twin™ 적용 맥락은, Q-Block 내 모든 단위큐브(512개 또는 그 이상으로 확장 가능)를 대상으로 하여, 각 큐브에 포함된 메타데이터 기반 중요도 점수($\text{importance} = w_d \cdot \text{density} + w_e \cdot \text{energy} + \dots$)를 기준으로 정렬하는 연산 구조를 의미한다. 이 정렬은 주로 병렬 연산 스케줄링 직전에 수행되며, 각 큐브의 연산 우선순위를 효율적으로 설정하기 위한 기반 절차로 활용된다. 특히, 정렬된 큐브 정보를 CPU로 복사하지 않고 GPU 내부에서 직접 처리함으로써, 실시간 시뮬레이션 환경에서도 높은 처리 속도와 동기화 효율성을 유지할 수 있다는 점에서 연산 최적화에 매우 유리하다.

[0137]

이러한 연산 정렬의 결과는 후속적으로 CUDA 기반 연산 큐 구성, anchor-to-anchor 기반 연산 전이, 시뮬레이션 시각화 흐름 제어 등에 연계되어 실시간 정보 확산 및 디지털 트윈 상태 변화 추적을 가능하게 한다. 나아가 Q-Twin™ 시스템에서는 이 정렬 구조가 실시간 트윈 시뮬레이션 갱신, 메타데이터 기반 중첩 예측, 연산 우선 배정, AI 학습 데이터 구성 등의 핵심 기준으로 작용하며, anchor-to-anchor 기반 경로 흐름과 결합됨으로써 고속·고정밀 디지털 트윈 연산 및 예측 기능을 동시에 강화하는 데 기여한다.

표 24

[0138]

Q-Twin™ 시스템과 기존 Thrust 정렬 방식 간의 기술적 차이점 비교표

구분	기존 Thrust 정렬 방식	Q-Twin™ 시스템 내 Thrust 정렬 방식
정렬 대상	일반적인 배열, 구조체 필드 (예: 값, 인덱스, ID 등)	Q-Block 내 단위큐브(UnitCube)의 메타데이터 기반 중요도 (density, energy, overlap 등)
정렬 목적	연산 단순화, 버퍼 최적화, 시각화 전처리 등 일반적 연산 흐름 정리	병렬 연산 스케줄링의 연산 우선순위 결정 + anchor-to-anchor 연산 흐름 연결용 시드 생성
기준 필드	단일 필드 값 (예: key, priority)	5가지 메타데이터의 가중 합 중요도 score (importance = $w_d \cdot \text{density} + w_e \cdot \text{energy} + \dots$)
정렬 결과 활용 구조	정렬된 데이터 출력 또는 단순 참조 구조	정렬 후 CUDA 커널로 직접 연결되어 연산 밀도 기반 흐름 트리거, 연산 자원 배정, AI 정책 초기값으로 활용됨
디바이스 내 처리 여부	GPU 내 정렬은 가능하나 CPU ↔ GPU 간 수시 전송 동반	GPU 내 정렬 및 후속 연산 전부 device 메모리 내에서 연속적으로 처리 가능하도록 구조화됨
연산 흐름 상 위치	독립적 연산 또는 전처리 단계로 존재	Q-Twin™ 내 연산 트리거 직전 핵심 스케줄링 단계로 내재화됨
AI 정책과의 연계성	없음	정렬된 큐브는 AI 탐색 정책(seed policy)의 초기 입력값으로 사용됨
시공간 연산 구조와의 결합성	연산 순서 정렬 이상의 기능 없음	Q-Block의 TSU 기반 적층 구조 및 anchor-to-anchor 흐름 예측과 직접 결합됨
정렬 단위	단일 배열 또는 구조체	시간축(TSU) 기반 복수 Q-Block 내 중심 단위큐브 간 정렬 및 연산 흐름 연계까지 포함
시뮬레이션 적용성	시뮬레이션 직접 반영 어려움	중요도 기반으로 정렬된 큐는 곧 연산 파장 확산 시각화 및 실시간 시뮬레이션 트리거로 연결됨

[0139]

다음은 Q-Twin™ 시스템 내에서 중심 단위큐브(anchor cube)의 메타데이터를 기반으로 강화학습(RL) 에이전트의 정책 시드(seed policy)를 초기화하는 예시 코드 및 연산 구조 흐름도를 나타낸 것이다. 본 예시는 Q-Block의 시간 특이점(TSU) 단위 적층 구조와 병렬 연산 흐름을 반영하여, 각 중심 단위큐브에서 수집된 시공간 메타데이터(예: 정보 밀도, 에너지 상태, 중첩 확률 등)를 강화학습의 초기 상태(state)로 구성한다. 이를 통해 에이전트는 anchor-to-anchor 경로 예측, 연산 우선순위 결정, 텔레포트 가능성 평가와 같은 고차원 연산 문제에 대해 정책 함수(policy function)를 효과적으로 학습할 수 있다. 특히 이 구조는 시뮬레이션 환경과 병렬 연산 흐름 간의 시간 정합성을 유지하며, 실시간 연산 흐름 최적화와 자율적인 연산 자원 배분의 기반으로 활용된다.

표 25

[0140]

Q-Twin™: AI 정책 시드 초기화 예시 (Python + PyTorch 스타일)

```
import torch
import torch.nn as nn

# 간단한 정책 네트워크 정의
class QTwinPolicyNet(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(QTwinPolicyNet, self).__init__()
        self.net = nn.Sequential(
            nn.Linear(input_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, output_dim)
        )

    def forward(self, x):
        return self.net(x)

# 중심 단위큐브에서 추출된 메타데이터 (예시)
anchor_metadata = {
    "density": 0.92,
    "energy": 0.81,
    "overlap": 0.63,
    "tsu_delta": 0.0,          # 기준 anchor에서 생성됨
    "vector_magnitude": 0.77
}

# 시드 벡터로 변환 (5차원 입력)
seed_vector = torch.tensor([
    anchor_metadata["density"],
    anchor_metadata["energy"],
    anchor_metadata["overlap"],
    anchor_metadata["tsu_delta"],
    anchor_metadata["vector_magnitude"]
], dtype=torch.float32)

# 정책 네트워크 초기화 및 예측
policy_net = QTwinPolicyNet(input_dim=5, hidden_dim=32, output_dim=3)
initial_action = policy_net(seed_vector)

print("Anchor-to-anchor 정책 시드 기반 초기화 결과:", initial_action)
```

[0141]

Q-Twin™에서의 AI 시드 초기화 흐름은 다음과 같은 단계로 구성된다. 첫째, 중심 단위큐브(좌표 (4,4,4))로부터 anchor 메타데이터를 추출하여 강화학습에 필요한 초기 상태 벡터로 변환한다. 둘째, density, energy, overlap, tsu_delta, vector_magnitude 등의 항목을 통합하여 시드 벡터를 구성하고, 셋째, 이를 정책 네트워크의 입력값으로 제공함으로써, 강화학습 기반 시뮬레이션에서 연산 방향성을 추정하거나 탐색 정책을 설정하는데 사용한다. 넷째, 이 시드 벡터는 다음 Q-Block의 중심 단위큐브로의 anchor-to-anchor 연산 예측에 활용되며, 연결 가능성, 경로 선택, 연산 트리거 발생 가능성 등의 판단 기준으로 적용된다.

[0142]

이러한 구조는 다양한 확장 시나리오와 연계될 수 있다. 예를 들어, Q-Block 간 연결 우선순위를 예측할 경우, AI 정책 네트워크는 $action = policy(anchor_state)$ 형태의 출력을 통해 다음 연산 경로가 될 Q-Block을 사전에 선택할 수 있으며, 이는 anchor-to-anchor 연산 흐름 최적화에 기여한다. 또한, 시공간 시뮬레이션 흐름 예측 시에는 anchor-to-anchor 전이 경로를 정책 함수의 출력으로 모델링함으로써 연산 파장 확산 방향을 예측할 수 있고, 결과적으로 트윈 시뮬레이션의 선제적 경로 확보 및 자원 할당에 활용 가능하다. 나아가, 이 시드 초

기화 구조는 강화학습 기반 학습 루프(state → action → reward)와 자연스럽게 연동될 수 있어, 반복 학습을 통해 시뮬레이션 내 연산 정책의 성능을 지속적으로 개선하는 구조로도 확장 가능하다. Q-Twin™의 이러한 연산 기반 AI 연계 구조는 단순한 트윈 모델링을 넘어, 지능적 경로 결정과 실시간 시공간 변화 예측을 가능하게 하는 핵심 기술 요소로 작용한다.

[0143] 4. 중첩 정보 집중 분석 및 가중치 필터링 알고리즘

[0144] 본 발명에서 정의하는 중첩 정보 집중 분석 및 가중치 필터링 알고리즘은 Q-Twin™ 시스템의 연산 흐름을 실질적으로 주도하는 핵심 기술 요소로서, 단순한 시공간 격자 구조를 넘어서 정보 중심의 디지털 트윈 환경을 구현하는 데 결정적인 역할을 수행한다. 본 알고리즘은 Q-Block 내 512개 이상의 단위큐브 중, 어떤 큐브가 연산의 출발점이 될지를 전략적으로 선별하는 역할을 수행하며, 이를 위해 각 큐브에 포함된 정보 밀도(density), 에너지 상태(energy), 중첩 확률(overlap), TSU 기반 시간 거리(tsu), 상태 벡터 크기(vector magnitude) 등 다양한 의미 기반 메타데이터 항목을 통합적으로 평가한다.

[0145] 이러한 중요도 기반 정렬 결과는 GPU 내 연산 큐(priority queue) 구성, anchor-to-anchor 전이 구조의 초기값(seed) 설정, 강화학습 기반 AI 정책의 학습 입력값 정제, 실시간 시각화 및 경로 탐색 흐름 생성 등 Q-Twin™ 시스템 내 주요 연산 모듈 전반에 걸쳐 핵심 기준으로 활용된다. 특히, 해당 알고리즘은 단순한 큐브 간의 구조적 인접성에 기반한 연산 순서를 넘어, 정보 중심성과 시뮬레이션 중요도를 반영한 질서 있는 연산 흐름을 생성함으로써, 본 발명의 고차원적 예측성, 확장성, 실시간성을 동시에 달성하는 데 중추적인 역할을 한다.

[0146] Q-Twin™ 시스템에서 연산 흐름의 출발점과 우선순위를 결정하는 핵심 절차는, 단순 거리 기준이 아닌 정보 중심성에 기반한 중첩 정보 필터링 알고리즘에 의해 수행된다. 이 알고리즘은 중심 단위큐브를 기준으로 주변 단위큐브들의 메타데이터를 수집하고, 각 항목에 가중치를 적용하여 연산 중요도(importance score)를 계산한다. 이후, 상위 점수를 가진 큐브들만을 선별해 후속 연산 모듈에 연계함으로써, 정보 밀도가 높고 연산 가치가 높은 영역부터 우선적으로 처리하는 전략적 연산 흐름을 실현할 수 있다. 그 절차는 다음과 같다;

표 26

[0147] Q-Twin™ 기반 중첩 정보 집중 분석 및 필터링 알고리즘 단계 요약

단계	설명
① 분석 대상 선별	중심 단위큐브(anchor cube)를 기준으로, 일정 반경 내 단위큐브들만 필터링
② 메타데이터 수집	각 단위큐브로부터 density, energy, overlap, TSU delta, 상태 벡터 등 주요 메타데이터 항목 수집
③ 가중치 적용	사전 정의된 가중치($w_1 \sim w_n$)를 곱하여 importance score 산출
④ 정렬 및 선택	score 기준으로 내림차순 정렬 후 상위 N개 또는 임계값 이상 항목 선별
⑤ 후속 연산 연계	선별 큐브 정보를 기반으로 anchor-to-anchor 예측, 경로 탐색 초기화, 텔레포트 후보 분석 등 수행

[0148] 이러한 다단계 필터링 프로세스는 단위큐브 간 단순한 구조적 배열을 넘어, 정보 중심 연산의 질서를 구성하는 핵심 알고리즘으로 기능하며, 특히 시공간 누적 기반의 연산 트윈 시스템인 Q-Twin™에서 연산 효율성과 예측 가능성을 동시에 보장하는 데 필수적인 역할을 수행한다. Q-Twin™ 시스템에서 연산 우선순위를 정보 중심적으로 정렬하기 위해, 단위큐브들의 메타데이터를 바탕으로 중요도 점수(importance score)를 계산하고 필터링하는 절차가 수행된다. 아래의 파이썬 스타일 의사코드(pseudocode)는 중심 단위큐브를 기준으로 일정 반경 내의 큐브들을 선별하고, 사전 정의된 가중치를 통해 중요도를 평가한 후, 가장 높은 N개의 큐브만을 연산 대상으로 선택하는 과정을 보여준다.

표 27

[0149]

Q-Twin™ 기반 중요도 필터링 알고리즘 (Pseudocode 예시)

```

class UnitCube:
    def __init__(self, coord, metadata):
        self.coord = coord
        self.metadata = metadata
        self.importance = 0

def compute_importance(cube, weights):
    return sum(weights[k] * cube.metadata[k] for k in weights)

def filter_high_importance_cubes(cubes, anchor_coord, radius, weights, top_n):
    filtered = []
    for cube in cubes:
        if euclidean_distance(cube.coord, anchor_coord) <= radius:
            cube.importance = compute_importance(cube, weights)
            filtered.append(cube)
    filtered.sort(key=lambda c: c.importance, reverse=True)
    return filtered[:top_n]

```

[0150]

상기 구조는 단순한 거리 기반 선별을 넘어서, 정보 밀도, 에너지 상태, 중첩 확률, TSU 시간 거리 등 의미 기반 요소를 통합 반영함으로써, Q-Twin™의 anchor-to-anchor 연산 흐름 초기화, AI 탐색 정책 학습, 병렬 연산 최적화 등에 있어 전략적 큐브 선별 기준으로 활용된다. 특히 이 알고리즘은 실제 CUDA 커널 또는 Thrust 기반 정렬과 연계될 수 있는 사전 연산 필터링 단계로서, 시공간 시뮬레이션의 정밀도와 효율성을 동시에 향상시키는 기술적 기반이다. 다음 CUDA 커널 코드는 Q-Twin™ 시스템 내에서 병렬 처리되는 수백 개의 단위큐브를 대상으로 하여, 중심 단위큐브(anchor cube)로부터 일정 반경 내에 위치한 큐브들만을 필터링하고, 각 큐브의 메타데이터에 가중치를 적용하여 중요도 점수(importance score)를 실시간으로 계산하는 구조를 나타낸다. 이 커널은 GPU 상에서 직접 실행되어 연산 속도를 극대화하며, 이후 단계인 중요도 정렬(Thrust sort 등) 및 연산 큐 스케줄링의 입력값으로 활용된다;

표 28

[0151]

Q-Twin™ 기반 CUDA 커널: 중요도 계산 및 반경 필터링 예시

```

__device__ float compute_importance(UnitCube cube, float wd, float we, float wo, float wt,
float ww) {
    return wd * cube.density + we * cube.energy + wo * cube.overlap + wt * cube.tsu_delta +
    ww * cube.vector_magnitude;
}

__global__ void computeImportanceScores(UnitCube* cubes, float* scores, int totalCubes,
float3 anchorCoord, float radius,
float wd, float we, float wo, float wt, float ww) {
    int tid = threadIdx.x + blockIdx.x * blockDim.x;
    if (tid >= totalCubes) return;
    UnitCube cube = cubes[tid];
    float dist = length(cube.coord - anchorCoord);
    scores[tid] = (dist <= radius) ? compute_importance(cube, wd, we, wo, wt, ww) : -1.0f;
}

```

[0152]

이 구현은 Thrust 정렬과 연계하여 GPU 내부에서 완전 자동화된 연산 우선순위 흐름을 구성하는 데 활용되며, CPU로 복사하는 과정 없이 Q-Twin™의 실시간 anchor-to-anchor 연산 전이, AI 정책 탐색 방향 설정, 시뮬레이션 초기화 등 핵심 흐름 제어에 직접 연결될 수 있다. 특히 시공간 상의 연산 충돌 예측이나 텔레포테이션 가능성 평가에도 실시간 필터링 로직으로 활용 가능하며, Q-Twin™의 고속 병렬 처리 기반 연산 최적화에 중요한 기

여를 한다. Q-Twin™ 시스템에서는 연산에 앞서 각 단위큐브의 중요도 점수(importance)를 기준으로 연산 우선 순위를 설정해야 하며, GPU 메모리 내에서 직접 정렬을 수행함으로써 실시간성과 병렬 처리 효율을 극대화할 수 있다. 아래는 NVIDIA의 Thrust 라이브러리를 활용하여 GPU 디바이스 메모리 상에서 직접 중요도 내림차순 정렬을 수행하는 예시 코드이다.

표 29

Q-Twin™ 시스템 내 Thrust 기반 중요도 정렬 예시

```
struct ImportanceComparator {
    __host__ __device__
    bool operator()(const UnitCube& a, const UnitCube& b) {
        return a.importance > b.importance;
    }
};

void sortCubesByImportance(UnitCube* d_cubes, int totalCubes) {
    thrust::device_ptr<UnitCube> begin(d_cubes);
    thrust::device_ptr<UnitCube> end = begin + totalCubes;
    thrust::sort(begin, end, ImportanceComparator());
}
```

이 구조는 Q-Block 내 수백 개의 단위큐브를 연산 중요도 순으로 자동 정렬하며, 후속 연산 스케줄링이나 AI 탐색 초기화, anchor-to-anchor 기반 시뮬레이션 전이에 직접 연결된다. Thrust 정렬은 CUDA 커널 호출과도 쉽게 연계되므로 Q-Twin™ 시스템에서 실시간 연산 흐름 최적화의 핵심 도구로 활용될 수 있다. Q-Twin™ 시스템의 병렬 연산은 단위큐브의 초기화부터 연산 스케줄링, AI 예측 모듈 연계에 이르기까지 일련의 단계를 따라 구조적으로 실행된다. 아래 표는 각 단계에서의 주요 처리 내용과 그 기술적 역할을 정리한 것이다;

표 30

Q-Twin™ 시스템 전체 연산 흐름 요약

단계	처리 내용	설명
1단계	UnitCube 초기화	ID, 위치, TSU, 메타데이터 부여
2단계	거리 계산	중심 큐브 기준 Euclidean 거리 기반 초기 우선순위 산정
3단계	중요도 계산	가중치 기반 importance score 산출
4단계	정렬 수행	Thrust or CPU sort 활용한 내림차순 정렬
5단계	병렬 큐 배정	중요도 순으로 연산 우선 배정 (CUDA 큐)
6단계	AI/탐색 연계	anchor-to-anchor 기반 예측, 경로 탐색, 텔레포트 후보 결정 등

Q-Twin™의 이 연산 흐름 구조는 단순한 GPU 연산 큐 구성에 그치지 않고, 각 단계가 강화학습 기반 정책 학습, 시뮬레이션 파라미터 튜닝, 디지털 트윈 상태 갱신 등과 유기적으로 연계되도록 설계되어 있다. 특히 병렬 연산의 효율성과 시공간 정보 중심성의 균형을 유지하는 데 있어 핵심적인 토대가 된다. Q-Twin™ 시스템에서 각 단위큐브(UnitCube)는 고유 식별자와 위치 좌표뿐만 아니라, 시공간 메타데이터와 연산 중요도 점수를 포함하는 구조체로 정의된다. 아래는 CUDA 및 병렬 연산 시뮬레이션에서 활용되는 구조 예시이다.

표 31

[0157]

Q-Twin™ 시스템용 UnitCube 구조체 정의(메타데이터 기반)

```
struct UnitCube {
    int id;                // 고유 식별자
    int x, y, z;           // Q-Block 내 3차원 위치 좌표
    float tsu;             // 생성 시점 (Time Singularity Unit)
    float density;         // 정보 밀도 (density)
    float energy;          // 에너지 상태 (energy)
    float overlap;         // 인접 큐브와의 중첩 확률 (overlap)
    float vector_magnitude; // 상태 벡터의 크기 또는 방향성 지표
    float importance_score = 0.0f; // 가중치 기반 연산 중요도 점수
};
```

[0158]

상기 구조체는 Q-Twin™에서 병렬 연산 큐 구성, anchor-to-anchor 연산 전이, AI 정책 초기화 등 다양한 연산 흐름의 기준 데이터 단위로 사용된다. 특히 importance_score 필드는 연산 우선순위 결정, 연산 파장 확산, 경로 탐색 초기값 설정 등에 실시간으로 활용된다. Q-Twin™의 디지털 트윈 연산 모델에서 이 필터링 구조는 단순한 연산 선별 기능을 넘어, 정보 중심적 병렬 연산의 핵심 엔진으로 작동하며, 자율 경로 탐색, 시뮬레이션 예측, 연산 우선 순위 설정과 같은 고차원 응용 기능의 기반이 된다. 이를 통해 Q-Twin™은 고밀도 정보 환경에서도 실시간 연산 효율성을 유지하고, 강화학습 기반 AI 정책 수립, 텔레포테이션 가능성 평가, 다중 Q-Block 연산 확장 등 다양한 고정밀 시뮬레이션 기술에 유연하게 대응할 수 있는 구조적 장점을 확보하게 된다.

[0159]

5. Q-Twin™ 시스템에서의 GPU 연산 내 중심 단위큐브의 논리적 제어 기능

[0160]

Q-Twin™ 시스템은 GPU 기반 병렬 연산 구조, 특히 CUDA 환경 내에서 중심 단위큐브(anchor cube)를 단순한 위치상 중심 좌표 이상의 역할로 정의한다. 중심 단위큐브는 Q-Block의 좌표상 중심((4,4,4))에 위치하며, 디지털 트윈 연산의 타이밍 조정, 연산 트리거링, 연산 흐름 확산, 연산 기준값 제공 등 다층적인 제어 기능을 수행하는 핵심 노드로 작동한다. Q-Twin™의 실시간 시뮬레이션 및 anchor-to-anchor 기반 연산 전이 구조 내에서 중심 단위큐브는 다음과 같은 논리적 기능을 수행한다:

[0161]

(1) 기준값 제공 노드(Reference Value Anchor) : 중심 단위큐브는 인접 큐브들의 연산 흐름에서 공통으로 참조되는 기준값을 제공한다. 에너지 상태, 중첩 확률, 상태 벡터 등의 메타데이터는 인접 단위큐브들의 연산 가중치 계산 또는 상태 비교 연산에서 기준값으로 작용하며, 연산 결과의 일관성과 공간적 정합성을 보장한다. 이는 디지털 트윈 상태 재현의 정확도에 직결된다.

[0162]

(2) 연산 트리거 결정자(Computation Trigger Origin) : 중심 단위큐브는 연산 시작 조건을 결정짓는 트리거 기준점으로 작동한다. 메타데이터 값(예: energy, overlap)이 임계값을 초과할 경우, 중심 단위큐브는 연산 개시 신호(trigger flag)를 설정하며, 이에 따라 CUDA 커널 내 다른 스레드들은 __syncthreads_or() 등을 통해 트리거 조건을 공유하고 연산 진입 여부를 판단한다. 이 구조는 시공간 연산 파장의 초기화 및 anchor-to-anchor 연산 흐름 시작점으로 기능한다.

[0163]

(3) 스레드 동기화 허브(Synchronization Anchor) : 연산 수행 중 CUDA 스레드 간의 동기화를 위해 중심 단위큐브는 기준 상태를 공유한다. __syncthreads()를 통해 중심 큐브의 연산 상태가 모든 스레드에 전달되며, 이로써 연산 타이밍의 일관성이 확보된다. 이러한 구조는 시뮬레이션 연산 중 스레드별 상태 갱신의 타이밍 충돌을 방지하며, 실시간 예측 연산의 안정성을 높인다.

[0164]

(4) 연산 흐름 정렬 및 타이밍 조율자(Wave Timing Coordinator) : 중심 단위큐브를 기점으로 연산 흐름이 공간적으로 확산되도록 구성되며, 이는 시뮬레이션 내에서 연산 파장이 전파되는 논리적 출발점이 된다. 거리 기반 정렬 및 중요도 기반 우선순위 큐 구성의 기준 노드로 작동하며, 연산 파장의 방향성 및 스케줄링 타이밍 조정의 기준이 된다.

[0165]

(5) anchor-to-anchor 연산 연결 지점(Seed for Inter-QBlock Transmission) : 다중 Q-Block으로 구성된 디지털 트윈 환경에서, 각 Q-Block의 중심 단위큐브는 블록 간 연산 연계의 anchor 역할을 수행한다. 현재 anchor에서 추출된 메타데이터는 다음 Q-Block의 연산 흐름 시드값으로 활용되며, 이는 A* 탐색 초기값 설정, 시뮬레이션 되감기 및 미래 상태 예측 등 다양한 anchor-to-anchor 기반 연산 흐름에 적용된다.

[0166]

이와 같이 Q-Twin™ 시스템에서 중심 단위큐브는 단순한 공간 좌표의 중심이 아니라, GPU 연산의 트리거, 연산

기준값, 동기화 허브, 연산 흐름 조율자, 그리고 Q-Block 간 연산 연결 지점이라는 복합적인 기능을 수행한다. 특히 도 12는 이러한 중심 단위큐브의 역할을 시각적으로 표현한 것으로, 병렬 연산 과장이 중심에서 외곽으로 확산되는 구조와 그 정렬 흐름이 애니메이션처럼 재현된다. 이는 기존의 grid-stride loop 방식처럼 단순 반복 처리에 기반한 병렬 연산 구조와는 명확히 구분되며, Q-Twin™ 시스템만의 anchor-to-anchor 기반 연산 타이밍 조율 및 흐름 중심 연산 제어 기능이 어떻게 시공간적 질서를 부여하는지를 직관적으로 보여준다. 결국 중심 단위큐브는 디지털 트윈 연산의 고속성, 예측 가능성, 정합성 유지라는 3대 핵심 요구를 충족시키는 정보 중심 병렬 연산 구조의 실질적 제어 허브로 작용하게 된다.

[0167] 6. Q-Twin™ 시계열 연산 그래프 구조 및 Anchor-to-Anchor 연결 메커니즘

[0168] Q-Twin™ 시스템은 단일 Q-Block 내의 병렬 연산 구조를 넘어서, 시간 특이점(Time Singularity)을 기준으로 순차 생성되는 다중 Q-Block 간의 연산 흐름 연결 구조를 구현한다. 이는 디지털 트윈 연산이 단일 시점의 정적 모사에 머무르지 않고, TSU(Time Singularity Unit) 단위의 시공간 흐름 전체를 지속적으로 적층하고, 예측 가능하도록 하기 위한 핵심 메커니즘이다. Q-Twin™ 구조에서는 매 TSU 단위로 새로운 Q-Block이 생성되며, 이들은 시간축 상에 위쪽으로 연속 적층되는 방식으로 누적된다. 이때 각 Q-Block의 중심 단위큐브(좌표 (4,4,4))는 해당 시점의 연산 중심 노드이자, 연산 시드 및 예측 기준점으로 작동하며, 이전 및 이후 Q-Block과의 연산 흐름 연결(anchor-to-anchor connection)을 담당한다. Q-Twin™의 시공간 연산 연결은 단순한 시점 연결을 넘어, 다음 요소를 기반으로 연결 강도(edge weight)를 정량화하고 연산 흐름을 제어한다. 아래 표는 Q-Twin™ 시스템에서 중심 단위큐브(anchor cube) 간 연산 흐름 연결 강도를 평가하기 위해 사용하는 주요 항목들을 요약한 것이다. 각 항목은 anchor-to-anchor 전이 경로의 안정성, 연산 연속성, 정보 일관성 등을 정량화하는 데 활용된다.

표 32

[0169] Q-Twin™ 시스템에서 Anchor-to-Anchor 연결 강도 산출 기준 요약

평가 항목	설명
메타데이터 유사도	인접 Q-Block 중심 anchor 간 상태 벡터의 유사도 (예: 코사인 유사도)
중첩 확률	두 anchor 간 정보 영역의 중첩 정도
정보 전이 이력	과거 anchor 간 정보 흐름의 빈도 및 안정성
에너지 변화량	연산 에너지 수준의 시점 간 차이 또는 전이 효율

[0170] 상기 요소들은 anchor 간 연결 강도를 edge weight 형태로 변환하여, 시공간 그래프 기반 경로 탐색 알고리즘(A*, Dijkstra 등)에 활용되며, 시물레이션 예측 정확도 향상 및 AI 학습 효율 최적화에 기여한다. Q-Twin™ 시스템에서 anchor-to-anchor 기반 연산 흐름 연결은 다음과 같은 단계로 체계적으로 수행된다. 첫째, 시물레이션 또는 경로 탐색을 시작하기 위해 현재 시점(TSU?)의 중심 단위큐브(anchor cube)를 출발점(seed anchor)으로 설정하며, 이 출발점은 GPU 기반 연산 초기화나 AI 정책 시드(seed policy) 설정의 기준값으로도 활용된다. 둘째, 각 중심 anchor 간의 연결 가중치(edge weight)를 기반으로 시공간 그래프(spatiotemporal graph)를 구성한 후, A* 또는 Dijkstra 알고리즘과 같은 경로 탐색 기법을 통해 최적 경로를 탐색한다. 이 과정에서는 메타데이터 유사도, 정보 일관성, 연산 효율성, 중첩 안정성 등의 요소가 종합적으로 고려되어 경로의 신뢰성과 연속성을 확보한다. 셋째, 단순히 물리적 거리나 시간 간격을 기준으로 하지 않고, 각 anchor에 축적된 메타데이터의 정보 중심성을 반영하여 의미 기반 경로(semantic path)를 도출한다. 이를 통해 Q-Block 간의 연산 흐름은 고차원적 정보 가치에 따라 확산되고, 이는 곧 시물레이션 예측 정확도와 AI 연산 효율을 동시에 극대화할 수 있는 핵심 전략으로 작동하게 된다.

[0171] 이와 같은 anchor-to-anchor 연결 구조는, TSU(Time Singularity Unit) 단위로 시계열적으로 적층된 Q-Block 그래프 상에서 각 중심 anchor를 기준으로 연산 흐름이 시간 축을 따라 확산되고 예측되는 연산 구조로 시각화될 수 있다. 특히 도 11에 도시된 바와 같이, 각 anchor cube의 메타데이터(예: 정보 밀도, 에너지 상태, 중첩 확률 등)의 시계열 변화 양상을 통해, 연산 집중 구간의 탐지, 텔레포트 가능성의 평가, 연산 충돌 가능성의 조기 예측 등 다양한 시물레이션 결과를 정량적으로 도출할 수 있다. 이러한 anchor-to-anchor 기반 연산 흐름 연결 메커니즘은 다음과 같은 시나리오에서 실질적으로 활용된다;

[0172] (1) 실시간 디지털 트윈 시물레이션에서는 시점별 변화 상태를 anchor를 매개로 연속적으로 연결함으로써, 변화의 추적뿐 아니라 예측까지 가능하게 한다.

- [0173] (2) 자율비행체(UAM, 드론 등)의 항법 계획에 있어서는 Q-Block의 연산 밀도 예측을 기반으로 anchor 시드를 따라 경로를 유동적으로 재설정할 수 있으며,
- [0174] (3) AI 시계열 학습 데이터 구성 시에는 anchor 간 전이 정보를 학습 loop로 구조화하여 예측 정확도와 정책 최적화를 동시에 도모할 수 있다.
- [0175] (4) 또한, 연산 되감기 또는 특정 시점 상태의 복원이 요구되는 환경에서는 anchor 메타데이터를 통해 과거 상태를 신속하게 재구성하는 것도 가능하다. 이와 같이 Q-Twin™ 시스템은 단일 시점의 병렬 연산 최적화에 머무르지 않고, 시공간 연산의 연속성, 정보 흐름의 의미 중심화, AI 기반 예측 가능성까지 통합적으로 고려하여 설계되어 있으며, 그 중심에는 anchor-to-anchor 연산 연결 구조가 자리하고 있다. 이는 Q-Block 기반 디지털 트윈 시뮬레이션의 전 영역에서 연산의 연속성과 예측성을 보장하는 핵심 기술 요소로 기능한다.

[0176] 7. Q-Twin™에서의 AI 연산 최적화 및 강화학습 기반 흐름 개선

[0177] Q-Twin™ 시스템은 단일 시점의 병렬 연산 최적화에 국한되지 않고, 다중 Q-Block 간의 시공간 연산 흐름을 학습 가능한 형태로 연결하고 최적화하는 AI 기반 강화 학습 구조를 포함한다. 이 구조는 중심 단위큐브(anchor cube)를 기준으로 수집된 정보 밀도, 연산 집중도, 전이 이력 등의 시공간 메타데이터를 바탕으로, AI가 스스로 연산 정책 및 경로 탐색 전략을 학습하고 강화하는 프레임워크를 제공한다. 특히, Q-Block 간 중심 단위큐브(anchor-to-anchor)의 연결 구조는 시공간 그래프(spatiotemporal graph)로 구성되며, 강화학습 기반 에이전트는 이 그래프 상에서 경로 선택, 연산 우선순위 결정, 연산 흐름 예측 등을 반복 학습한다. Q-Twin™ 시스템은 AI 기반 시뮬레이션 최적화 구조를 중심으로, anchor-to-anchor 흐름 예측과 정책 학습을 통합하여 연산 흐름의 방향성과 우선순위를 자율적으로 학습하고 결정할 수 있도록 설계된다. 다음은 이러한 AI 기반 연산 최적화 흐름의 전체 절차를 요약한 것이다;

표 33

[0178] Q-Twin™ 시스템의 AI 기반 연산 최적화 절차 요약

단계	설명
① Anchor 수집	TSU 순서대로 생성된 Q-Block에서 중심 단위큐브(anchor)를 추출
② 연결 가중치 계산	메타데이터 유사도, 중첩 확률, 에너지 변화량 등으로 edge weight 계산
③ 시공간 그래프 구성	anchor 노드를 중심으로 A* 또는 Dijkstra 탐색을 위한 시공간 그래프 구성
④ 탐색 정책 실행	그래프 상에서 anchor-to-anchor 흐름을 따라 최적 경로 및 연산 방향 탐색
⑤ 강화학습 연계	정책(policy)을 강화학습 기반으로 학습하고 reward 기반 최적화 수행
⑥ GPU 병렬화 및 흐름 정렬	CUDA 기반 환경에서 anchor 상태 기준 연산 트리거 및 스레드 동기화 수행

[0179] 상기 표에서 요약된 각 단계는 Q-Twin™이 단순한 시뮬레이션 도구를 넘어, 실시간으로 스스로 학습하고 최적화하는 디지털 트윈 시스템으로 작동할 수 있도록 지원하는 핵심 알고리즘 흐름이다. 특히, 이러한 연산 최적화 흐름은 시공간 상에서 시간 순으로 적층된 Q-Block 간 중심 단위큐브(anchor cube)를 중심으로 구성된 연산 구조와 밀접하게 연결된다. 이러한 전체 흐름은 도 13에 도시된 바와 같이, 연산 흐름 생성 → anchor-to-anchor 연결 → 시공간 그래프 탐색 → AI 정책 학습이라는 일련의 과정을 시각적으로 구성한 것이다. 이를 통해 Q-Twin™ 시스템이 단순 병렬 처리 수준을 넘어, 강화학습 기반으로 연산 정책을 점진적으로 최적화하며, 정보 중심 기반의 예측 가능한 경로 탐색을 수행함을 명확히 이해할 수 있다.

[0180] 이후 단계에서는 이러한 연산 흐름이 실제로 CUDA 환경에서 anchor 간 유사도 기반으로 어떻게 실현되는지에 대해 구체적인 예시를 통해 살펴본다. 앞서 정리한 Q-Twin™의 AI 기반 연산 최적화 절차는 실제 구현에서 anchor-to-anchor 흐름을 시공간 그래프 형태로 구성하고, 이를 기반으로 경로를 예측 및 최적화하는 알고리즘 구조로 구체화된다. 다음은 각 Q-Block에서 중심 anchor를 추출하고, 이를 연결하여 A* 알고리즘 기반의 최적 경로를 탐색하는 Pseudocode 예시이다;

표 34

[0181]

Q-Twin™ 기반 Anchor-to-Anchor 시공간 경로 탐색 구조(Pseudocode)

```
# Anchor 간 edge를 구성하는 시공간 그래프 생성
graph = Graph()
prev_anchor = None

for qblock in sorted(QBlocks, key=lambda b: b.TSU):
    anchor = qblock.get_anchor()
    if prev_anchor:
        w = compute_weight(prev_anchor, anchor)
        graph.add_edge(prev_anchor.id, anchor.id, w)
    prev_anchor = anchor

# A* 기반 경로 탐색 초기화
start = QBlocks[0].get_anchor()
goal = QBlocks[-1].get_anchor()
optimal_path = a_star_search(graph, start, goal)
```

[0182]

이 구조는 Q-Twin™의 anchor-to-anchor 연산 흐름을 정렬하고 예측하는 데 활용되며, 단순한 시점 간 연결을 넘어 강화학습 기반 정책(policy)을 점진적으로 최적화하는 시공간 연산 프레임워크를 실현한다. 이러한 경로 탐색 결과는 이후 GPU 기반 시뮬레이션, AI 연산 트리거링, 디지털 트윈의 예측 정책 강화 등에 직접적으로 반영될 수 있다. 앞서 Pseudocode를 통해 anchor-to-anchor 기반 시공간 경로 탐색의 로직 구조를 살펴보았다면, 이번에는 이를 GPU 환경에서 실시간으로 처리하기 위한 CUDA 기반 구현 예시를 살펴본다. 특히, 수백 개 이상의 중심 단위큐브 간의 유사도를 병렬로 연산해야 하는 Q-Twin™ 구조의 특성상, 각 anchor 간 메타데이터 유사도를 빠르게 평가하고 연결 가중치(edge weight)를 계산하는 병렬 커널이 필요하다.

표 35

[0183]

CUDA 기반 Anchor-to-Anchor 유사도 평가(Q-Twin™ 적용 예시)

```
__global__ void computeAnchorConnections(AnchorCube* anchors, float* weights, int
totalAnchors) {
    int i = blockIdx.x;
    int j = threadIdx.x;

    if (i == j || i >= totalAnchors || j >= totalAnchors) return;

    AnchorCube a = anchors[i];
    AnchorCube b = anchors[j];

    float sim = 0.0f;
    for (int k = 0; k < 5; ++k) {
        float diff = a.metadata[k] - b.metadata[k];
        sim += diff * diff;
    }

    weights[i * totalAnchors + j] = __expf(-sim); // 유사도 기반 가중치
}
```

[0184]

상기 CUDA 커널은 Q-Twin™ 시스템 내에서 anchor-to-anchor 연결 강도를 실시간으로 산출하기 위한 구조이며, GPU 상에서 수백~수천 개 anchor 간의 유사도를 동시에 계산하여 A* 또는 Dijkstra 기반 탐색 알고리즘에 입력으로 제공된다. 이 구조는 anchor 기반 연산 흐름 예측, AI 탐색 정책 초기화, 연산 우선순위 정렬 등에 활용되어, 시공간 연산 예측의 실시간성과 연산 밀도 기반 경로 결정 정확도를 크게 향상시킨다.

[0185] 이와 같은 기술적 효과 및 확장성은 Q-Twin™ 시스템이 단순한 병렬 연산 구조를 넘어서, 실시간 연산 흐름의 예측성과 자율성을 동시에 확보하는 고차원적 AI 연산 프레임워크임을 입증한다. 강화학습 기반 정책 최적화를 통해 연산 방향성 및 AI 추론 흐름의 품질이 지속적으로 향상되며, anchor를 중심으로 병렬 스트림 간 연산 타이밍을 정밀하게 동기화함으로써 고속 처리가 가능해진다. 특히, anchor 간 전이 구조를 AI 시드로 활용하는 예측 메커니즘은 후속 Q-Block의 연산 흐름을 사전 시뮬레이션할 수 있게 하여, 시공간적 예지력(spatiotemporal foresight)을 제공한다. 또한, cuGraph나 Thrust와 같은 GPU 최적화 라이브러리와 연계를 통해, 대규모 시공간 그래프에서도 효율적인 병렬 탐색이 가능하도록 구조적 확장이 자연스럽게 이뤄진다. 결과적으로, Q-Twin™ 시스템은 정보 중심 연산의 방향성 추론과 강화학습 기반의 정책 최적화를 통합하여, 자율비행체 항법, 공중권(Air Rights) 기반 도시 설계, 시계열 기반 디지털 트윈 예측 등과 같은 실시간성이 요구되는 고성능 응용 환경에서, 탁월한 확장성과 예측 가능성을 갖춘 차세대 AI 연산 인프라로 자리매김하게 된다.

[0186] 8. Q-Twin™ 기반 시각화 및 시뮬레이션 인터페이스

[0187] Q-Twin™ 시스템은 단순한 연산 결과 출력 수준을 넘어, 디지털 트윈 상태의 실시간 병렬 연산 흐름을 시각적 요소로 표현하고, 사용자 조작 기반의 인터랙티브 시뮬레이션 환경을 제공한다. 도 14에서 보이듯, Q-Twin™의 시각화 인터페이스는 단순한 연산 출력 UI를 넘어, 시공간 연산 흐름의 이해, 시뮬레이션 상태 진단, AI 학습 강화까지 통합적으로 지원하는 고차원적 도구로 설계된다. 이하 표는 Q-Twin™ 시뮬레이션 인터페이스의 주요 구성 요소와 그 기능, 기술적 효과를 요약한 것이다;

표 36

[0188] Q-Twin™ 시각화 인터페이스 구성 요소 요약

구성 요소	설명	효과
3D 인터페이스	8×8×8 Q-Block 격자, 중심 anchor 시각적 강조	연산 기준점 명확화
병렬 연산 상태 표현	importance, energy, 연산 파장 등 시각적 속성 적용	연산 흐름 실시간 인지
anchor 경로 탐색 시뮬레이션	A*/RL 기반 경로 탐색을 3D 연결선으로 시각화	경로 신뢰도 비교 및 검증
연산 히트맵	high-importance 영역 강조 컬러맵 출력	병목 분석, 리소스 재배치에 활용
사용자 인터랙션	큐브 단위 접근 및 시뮬레이션 조건 조정	정책 테스트 및 피드백 기반 학습

[0189] (1) 디지털 트윈 기반 3D 인터페이스 : Q-Twin™ 내 Q-Block은 8×8×8 정규 격자로 구성되며, 각 단위큐브(Unit Cube)는 디지털 트윈의 시공간 정보를 표현하는 최소 단위로서 3차원 공간 상에 시각화된다. 중심 단위큐브(anchor cube)는 투명도, 색상 하이라이트, 상징적 애니메이션 등을 통해 명확히 구분되며, 병렬 연산의 기점으로 시각적 기준축이 된다.

[0190] (2) 병렬 연산 상태 시각화 및 파장 애니메이션 : 이 기능은 각 단위큐브의 연산 상태를 색상, 투명도, 크기 진동 등의 시각적 속성으로 표현함으로써, 시뮬레이션의 정적 결과를 넘어서 연산 흐름 자체의 동적인 진행 과정을 실시간으로 가시화할 수 있도록 한다. 구체적으로, 각 큐브는 연산 중요도(importance score) 또는 우선순위에 따라 컬러 스케일이 조정되며, 에너지 소모량 또는 연산 누적량에 따라 투명도가 조절되어 정보 집중 구역을 시각적으로 강조한다. 또한 중심 큐브에서 외곽으로 연산이 확산되는 과정을 파동(wave) 형태의 크기 진동이나 애니메이션으로 표현함으로써, 사용자는 병렬 연산이 공간적으로 어떻게 전개되는지를 직관적으로 이해할 수 있다. 이를 통해 Q-Twin™은 단순한 연산 결과 출력 수준을 넘어서, 정보 흐름의 시작점, 확산 경로, 연산 집중도 등을 동적으로 시각화하는 실시간 인터페이스를 제공하며, 시뮬레이션 예측 정확도 향상 및 AI 정책 검증 등 다양한 응용에도 기여한다.

[0191] (3) Anchor-to-Anchor 경로 탐색 시뮬레이션 : 강화학습 기반 또는 A*/Dijkstra 알고리즘 기반의 anchor 간 경로 탐색 결과는 3D 공간 내 선형 그래프 구조로 표현된다. 연산 우선경로는 실선, 대안 경로는 점선 또는 반투명 연결선으로 구분되며, 각 연결선(edge)의 색상은 연결 강도(weight) 또는 전이 신뢰도(similarity confidence)를 표현한다.

[0192] (4) 연산 가치치 맵 및 히트맵 출력 : Q-Twin™ 시스템에서는 각 단위큐브의 연산 중요도(importance score),

메타데이터 기반 정보 밀도, 그리고 AI 학습 기여도 등을 종합적으로 분석하여, 고정밀 컬러 히트맵 또는 벡터 분포 맵을 생성할 수 있다. 이 시각화 결과에서는 고가중치 연산이 집중된 영역이 붉은 계열로 표시되고, 반대로 정보 밀도가 낮거나 비활성화된 영역은 푸른 계열로 표현되어, 전체 연산 공간 내에서의 연산 부하 분포를 직관적으로 파악할 수 있다. 이를 통해 사용자는 병목 현상이 발생하는 연산 구간을 쉽게 식별할 수 있으며, 연산 리소스의 효율적인 재배치 및 스케줄링 전략을 수립하는 데 실질적인 인사이트를 확보할 수 있다.

[0193] (5) 사용자 인터랙션 및 조건 기반 시뮬레이션 조절 : 사용자는 마우스 또는 HMI(Human-Machine Interface) 상의 포인터 조작을 통해 각 단위큐브에 직접 접근할 수 있으며, 해당 큐브의 고유 식별자(ID), 생성 시점(TSU), 상태 벡터 등의 메타데이터를 실시간으로 열람할 수 있다. 또한, 시뮬레이션 인터페이스에 포함된 슬라이더나 조건 패널을 활용하여 사용자는 중요도 점수(importance score)의 필터링 임계값, 시뮬레이션이 적용될 TSU 범위, AI 기반 알고리즘과 휴리스틱 기반 알고리즘 간의 경로 비교 설정 등을 조정할 수 있다. 이와 더불어, 강화 학습의 policy map 시각화, anchor-to-anchor 흐름의 경로 비교, 사용자 피드백을 반영한 정책 강화 등의 고차원적인 양방향 시뮬레이션 기능도 제공된다. 이러한 기능들은 단순한 시각적 출력 이상의 상호작용성과 분석 기능을 갖추고 있으며, Q-Twin™ 시스템의 실시간 연산 제어 및 AI 학습 최적화에 실질적인 기여를 한다.

표 37

[0194] Q-Twin™ 시각화 및 시뮬레이션 인터페이스의 기술적 효과 요약

기능	효과
연산 흐름 실시간 시각화	병렬 연산 동기화 및 오류 흐름 조기 탐지
경로 탐색 검증	AI 정책 기반 경로 결과와 휴리스틱 결과의 비교 가능
AI 학습 피드백 수집	사용자 조작 기반 보상 설계 및 정책 튜닝 가능
시스템 디버깅 및 검증	병목 현상, 연산 우선순위 불균형 등을 실시간 확인

[0195] Q-Twin™의 시각화 및 시뮬레이션 인터페이스는 단순 UI를 넘어, AI 학습 환경, GPU 연산 흐름 점검, 경로 정책 검증이 가능한 다차원적 인터페이스이자, Q-Block 기반 실시간 시공간 연산 시스템의 완결성을 지원하는 핵심 모듈로 설계된다. 이 모듈은 향후 Unreal Engine, Unity, WebGL 등과의 연동을 통해 메타버스·UAM·디지털 트윈 기반 응용까지 확장될 수 있다.

실시예 1

[0196] Q-Block 기반 GPU 병렬 시뮬레이션 연산 예시(CUDA 코드 기반) : 본 실시예에서는 Q-Twin™ 시스템의 연산 엔진이 GPU 기반 CUDA 환경에서 병렬 처리되는 방식을 설명한다. 각 Q-Block은 $8 \times 8 \times 8$ 의 총 512개 단위큐브로 구성되며, 이를 CUDA threadIdx 기준으로 병렬 처리하기 위해 다음과 같은 연산 구조를 사용한다;

표 38

[0197]

Q-Twin™ 기반 CUDA 커널: 중심 단위큐브 기준 병렬 연산 시뮬레이션 예시

```
// CUDA 커널: 각 단위큐브에 대해 병렬 연산 수행
__global__ void simulateQBlock(UnitCube* cubes, int tsu) {
    int idx = threadIdx.x + blockDim.x * threadIdx.y + blockDim.x * blockDim.y * threadIdx.z;
    if (idx >= 512) return;

    UnitCube cube = cubes[idx];

    // 중심 단위큐브(anchor) 기준 비교
    UnitCube anchor = cubes[ANCHOR_INDEX]; // ANCHOR_INDEX = 4*64 + 4*8 + 4 = 292

    // 상태 유사도 계산
    float similarity = compute_similarity(cube.meta, anchor.meta);

    // 연산 가중치 및 트리거 판단
    float weight = compute_weight(cube.meta, similarity);

    // 결과 저장 (예: 상태 갱신)
    cube.meta.state += weight * tsu;

    // 연산 결과를 다시 저장
    cubes[idx] = cube;
}
```

[0198]

상기 CUDA 연산 흐름은 단위큐브 간의 메타데이터 유사도, TSU 기반 시간 거리, 중심 단위큐브 중심성 등을 고려하여 각 큐브의 시뮬레이션 상태를 병렬로 갱신하며, 실시간 디지털 트윈 연산의 고속성과 확장성을 보장한다. 도 8은 이러한 연산 구조의 시각적 결과를 예시한 것으로, $z=4$ 평면 상의 Q-Block 단면을 기준으로, 좌측에는 각 단위큐브의 초기 상태 분포가, 우측에는 중심 단위큐브(4,4,4)를 기준으로 연산 가중치가 적용된 후의 상태 변화 결과가 각각 도시되어 있다. 각 단위큐브는 중심 단위큐브와의 메타데이터 유사도 및 상대적 거리 등에 따라 상이한 연산 가중치를 부여받으며, 그 결과는 병렬 연산 흐름 내에서 정보 중심성, 연산 파장 확산, 시공간 변화 예측 등의 핵심 연산 기능으로 반영된다.

[0199]

또한, 도 9는 본 발명의 Q-Twin™ 시스템이 Unreal Engine 또는 Unity와 연동되어 실시간 시각화가 수행되는 구조를 예시한 것이다. 해당 도면은 $z=4$ 평면 상의 Q-Block 단면을 기준으로, 좌측에는 각 단위큐브의 정보 밀도 분포를 색상으로 시각화한 결과가, 우측에는 중심 단위큐브(4,4,4)를 기준으로 anchor-to-anchor 경로 상에서 예측된 위험도 점수가 시각적으로 표현되어 있다. 이러한 시각화 구조는 Q-Block 기반 디지털 트윈 연산 결과를 외부 시각화 엔진에서 실시간으로 렌더링하고, 단위큐브 간의 메타데이터 흐름, 연산 집중도, 경로 충돌 가능성 등을 직관적으로 표현할 수 있도록 하며, 디지털 트윈 기반 도시 시뮬레이션, 자율비행체 경로 분석, 메타버스 공간 연산 등 다양한 응용 시나리오에 효과적으로 적용될 수 있다. 이와 같은 구성은 외부 시각화 엔진과의 API 연동을 통해, Q-Block 단위큐브의 연산 결과를 실시간으로 시각화할 수 있게 하며, 메타데이터 기반의 단위큐브 객체를 외부 엔진 내 독립 객체로 바인딩하여 상호작용, 분석, 제어가 가능하도록 한다. 특히, 시각화 엔진에서의 렌더링 객체와 단위큐브 간의 구조적 일치는, 디지털 트윈 기반의 실시간 분석 및 사용자 인터페이스 설계에 있어서 기술적 효과를 극대화한다.

[0200]

이러한 도 8 및 도 9는 단순한 시각화 예시를 넘어, 본 발명이 실현하고자 하는 디지털 트윈 구조의 핵심 원리를 시각적으로 입증하는 사례이다. Q-Twin™ 시스템은 현실 공간에서 수집된 시계열 데이터를 Q-Block 단위로 적층하고, 이들 간의 anchor-to-anchor 연결을 통해 시공간 변화의 흐름을 연산적으로 재현한다. 도 8은 해당 Q-Block 내부 단위큐브들이 중심 단위큐브를 기준으로 연산 가중치를 받아 병렬적으로 상태를 갱신하는 구조를 시각화함으로써, 현실 시점(TSU)에서의 상태 변화가 Q-Twin™ 디지털 트윈 상에서 어떻게 계산되고 반영되는지를 보여주는 예시이다. 이는 단위큐브가 현실 데이터(예: 센서 값, 영상 인식 결과 등)로부터 생성되었을 때, 그 정보들이 GPU 환경에서 병렬로 연산되어 디지털 공간 상의 상태를 어떻게 동적으로 반영하는지를 입증한다. 또한, 도 9는 Unreal/Unity 기반 엔진을 통해 외부 시각화 모듈과 연계된 디지털 트윈의 "실시간 변화 관측 인

터페이스"를 예시한 것으로, 실제 현실 공간에서 수집된 밀도 정보가 단위큐브에 매핑되고, anchor 간 시계열 연결을 통해 시간에 따른 변화 예측 및 경로 분석이 가능함을 보여준다.

표 39

[0201]

Q-Twin™ 디지털 트윈 기술과 도 8, 도 9 시각화 예시 간의 대응 관계 요약

구분	도 8: CUDA 기반 Q-Block 연산 시각화	도 9: Unreal/Unity 기반 트윈 시각화 예시
기반 기술	GPU 병렬 연산 (CUDA), 중심 단위큐브(anchor) 기반 연산 정렬 구조	Unreal Engine 또는 Unity 연동 시각화 구조 (Blueprint, Niagara 등 포함)
시각화 대상	z=4 평면 기준 단위큐브들의 상태 변화 (초기 상태 vs. 연산 후 상태)	단위큐브 밀도 시각화, anchor-to-anchor 위험도 예측 경로 시각화
Q-Twin™과의 연관성	현실 데이터를 중심 단위큐브 기준으로 병렬 처리 및 시계열 반영	현실 센서 기반 상태를 시각적으로 복제하고 미래 상태 변화까지 예측 가능
디지털 트윈 구현 요소	메타데이터 기반 연산 가중치 계산 및 상태 갱신 흐름 시뮬레이션	anchor 중심 시계열 트윈 적층, 실시간 정보 흐름 시각화 및 경로 탐색
응용 가능 분야	고속 상태 갱신 연산 구조 검증, 연산 파장 시뮬레이션	도시 디지털 트윈, 자율비행체 위험도 시각화, 경로 충돌 예측
기술 효과	실시간 연산 성능, 시공간 축적 흐름 기반 연산 최적화	실시간 시각화 + 경로 예측 연계로 고차원 디지털 트윈 구현 가능

[0202]

이는 곧 Q-Twin™이 단순한 3D 모델이 아니라, "현실 공간의 변화 흐름을 시공간 구조화하고 예측 가능한 상태로 재현하는 디지털 트윈 연산 엔진"임을 뒷받침하는 핵심적 시뮬레이션 사례에 해당한다. 결론적으로, 도 8과 도 9는 본 발명의 기술 구성인 중심 단위큐브 중심 연산 구조, anchor-to-anchor 기반 정보 전이, 그리고 GPU 병렬 연산 기반 연산 동기화 기술이 디지털 트윈의 실시간 상태 반영, 예측 연산, 인터페이스 구현 등과 직접적으로 연결되어 있으며, 이러한 구조는 Q-Twin™이 현실 공간의 시계열 변화 정보를 기반으로 정밀한 디지털 복제 및 시뮬레이션을 가능하게 함을 입증한다.

실시예 2

[0203]

Unreal Engine 기반 3D 트윈 시각화 연동 : Q-Twin™ 시스템은 Unreal Engine과의 연동을 통해 실시간 시각화 인터페이스를 제공할 수 있다. 이 연동 구조는 연산 엔진(CUDA 또는 Python)에서 수행된 Q-Block 병렬 연산 결과를 JSON 또는 FlatBuffer와 같은 경량 직렬화 포맷으로 출력한 뒤, Unreal Engine 내 Blueprint 또는 C++ 모듈을 통해 이를 로드하여 3D Q-Block Mesh에 반영하는 방식으로 구현된다. 특히 Niagara Particle System을 활용하면 각 단위큐브의 상태 벡터 값을 입자의 색상 변화, 크기, 운동량 등으로 시각화할 수 있으며, 이를 통해 연산 흐름, 중요도 분포, 에너지 상태 등을 직관적으로 표현할 수 있다. 이 구조는 고속 병렬 연산과 고품질 시각화를 유기적으로 결합함으로써, Q-Twin™ 시스템이 요구하는 실시간 디지털 트윈 시뮬레이션 및 AI 시각화 응용에 최적화된 환경을 제공한다.

표 40

[0204]

Unreal Engine 기반 Q-Block 시각화 연동 예시(C++ 코드 스니펫)

```
FQBlockData LoadedData = LoadFromJSON("QBlock_Results.json");

for (int i = 0; i < 512; ++i) {
    FVector CubePosition = ComputePosition(i);
    float Density = LoadedData.Cubes[i].Density;
    FColor CubeColor = GetColorByDensity(Density);

    UStaticMeshComponent* Cube = CreateCubeMesh(CubePosition, CubeColor);
    Cube->AttachToComponent(RootComponent, FAttachmentTransformRules::KeepWorldTransform);
}
```

[0205] 상기 표는 Q-Twin™ 시스템에서 생성된 Q-Block 병렬 연산 결과를 Unreal Engine 환경에서 시각화하기 위한 핵심 C++ 코드 흐름을 나타낸다. 구체적으로는 JSON 형식으로 저장된 각 단위큐브(UnitCube)의 메타데이터를 Unreal Engine에서 불러와, 위치별로 정렬하고 정보 밀도(Density)에 따라 시각적 속성(색상, 위치 등)을 적용한 후, 3D 공간 내에 정적 메쉬(Cube Mesh)로 구성하는 구조를 보여준다. 이 과정은 실시간 시뮬레이션 인터페이스 또는 분석 환경에서 Q-Block 내부 상태를 직관적으로 확인할 수 있도록 설계되었다. 정보 밀도 값(Density)이 높은 단위큐브는 붉은색 계열로, 밀도가 낮은 큐브는 푸른색 계열로 시각화되며, 시뮬레이션의 시간 흐름에 따라 Q-Block은 수직(상방) 방향으로 TSU(Time Singularity Unit) 단위로 적층된다. 각 중심 단위큐브(anchor cube) 간의 연결(anchor-to-anchor transition)은 선(line) 또는 입자 흐름 형태로 시각화되어, 연산 전이 방향과 경로 선택 흐름을 동적으로 표현할 수 있다. 이러한 구성은 디지털 트윈 내 anchor 기반 연산 흐름을 실시간으로 직관화하는 데 매우 효과적이다.

실시예 3

[0206] *디지털 트윈 재현 시나리오: TSU₀ ~ TSU₁₀₀ 시계열 Q-Block 적층 시뮬레이션:* 본 실시예는 Q-Twin™ 시스템이 구현하는 디지털 트윈 환경에서, 시간 특이점 단위(TSU: Time Singularity Unit)로 적층된 Q-Block 시퀀스를 실시간으로 시각화하고, 각 Q-Block의 중심 단위큐브를 기점으로 연산 파장이 외부로 확산되는 흐름을 연출하는 과정에 관한 것이다.

[0207] 상기 시나리오에서 구성된 시뮬레이션 환경은 Q-Twin™ 시스템이 TSU₀부터 TSU₁₀₀까지 총 101개의 Q-Block을 시계열로 적층하여, 각 시점의 연산 흐름과 메타데이터 상태를 연속적으로 모사할 수 있도록 설계되어 있다. 각 Q-Block은 8×8×8 크기의 단위큐브로 구성되며, 그 중심에는 항상 좌표 (4,4,4)에 위치한 중심 단위큐브가 존재하여 해당 시점의 연산 중심 노드로 기능한다. 매 시점마다 각 단위큐브의 메타데이터는 외부 센서 데이터 또는 시뮬레이션 입력값에 따라 실시간으로 갱신되며, 이 메타데이터를 기반으로 연산 흐름이 조정된다. 특히 anchor-to-anchor 연산 흐름 구성은 시계열 디지털 트윈의 핵심으로 작동한다.

[0208] 이전 시점 Q-Block의 중심 anchor로부터 추출된 중요 메타데이터(density, energy, overlap 등)는 다음 시점의 anchor로 전이되어 연산 시드(seed value)로 사용되며, 이를 통해 연산 우선순위가 초기화되고, 시점 간의 연산 흐름이 연속성을 유지하면서 확장된다. 이 구조는 연산 타이밍 및 연산 트리거의 정합성을 확보함과 동시에, 강화학습 기반의 AI 정책 최적화에도 직접적으로 활용된다.

[0209] 시각화 측면에서는 각 중심 단위큐브를 기점으로 파동형 애니메이션(wave-like expansion)이 생성되며, 병렬 연산의 확산 흐름을 직관적으로 표현한다. 각 단위큐브는 중요도(importance score)에 따라 색상 및 투명도가 조절되며, 중요도가 높은 큐브는 붉은색과 불투명하게, 중요도가 낮은 큐브는 푸른색과 반투명하게 시각화된다. 이로 인해 연산 집중 구간(high-density computation zone)이 자동적으로 강조되며, 병목 구간 탐지 및 리소스 재배치 전략 수립에 실질적인 도움을 제공한다. 각 TSU별 Q-Block은 수직 방향으로 누적되어 3D 타워형 구조를 이루며, 전체 디지털 트윈 시스템의 시공간적 확장을 입체적으로 표현할 수 있다.

[0210] AI 연계 측면에서는 anchor-to-anchor 간 연결이 시공간 그래프의 edge로 구현되어 실시간 탐색 경로로 활용되며, AI는 이 흐름을 강화학습 루프 내에서 학습하고, 정책을 점진적으로 강화한다. 연산 경로 선택, 리소스 집중도 분석, 예상 충돌 회피 등은 모두 이러한 학습된 정책을 통해 실시간으로 갱신되며, 시뮬레이션 성능의 고도화를 유도한다.

[0211] 결론적으로 본 실시예는 Q-Twin™ 시스템이 단일 시점의 연산 처리에 그치지 않고, anchor 기반의 연산 시드 전이를 통해 시계열 디지털 트윈 연산 체계를 정밀하게 구현할 수 있음을 입증하는 대표적인 사례이다. 이는 자율 비행체 항법 경로 시뮬레이션, 스마트시티 센서 데이터 예측, 대규모 정책 검증 환경 등 다양한 고속 연산 기반 응용 분야에서 Q-Twin™의 적용 가능성과 기술적 유효성을 뒷받침하는 근거로 작용한다.

실시예 4

[0212] *재난 대응 훈련 시뮬레이션:* Q-Twin™ 시스템을 활용하여 화재, 붕괴, 연기 확산 등 복합 재난 상황을 가정한 시뮬레이션 환경을 구성하고, 센서로부터 유입되는 실시간 데이터를 단위큐브 메타데이터에 반영함으로써 위험 구역을 자동 식별하고 시각적으로 강조하는 시나리오를 구현한다.

[0213] 다음은 Q-Twin™ 시스템을 활용한 재난 대응 훈련 시뮬레이션의 구체적 실시예이다. 본 시나리오는 실내 공간에

설치된 다양한 IoT 센서 데이터를 실시간으로 Q-Block 구조에 반영함으로써, 위험 구간의 시각화, 예측, 대응 전략 학습까지 포괄하는 디지털 트윈 기반 훈련 환경을 구성한다. 먼저, 시뮬레이션 환경은 훈련 대상 공간을 $8 \times 8 \times 8$ 단위큐브로 이루어진 3차원 격자형 Q-Block으로 모델링하여 구현된다. 각 단위큐브는 온도 센서, 연기 감지기, 진동 센서 등으로부터 실시간으로 유입되는 데이터를 메타데이터로 반영받는다. 구체적으로는 온도(temperature)는 energy 값에, 연기 농도(smoke_density)는 density 값에, 진동 세기(vibration_intensity)는 overlap 값에 각각 매핑된다. 이 연산 흐름의 기준점은 건물 중심 좌표(4,4,4)에 위치한 중심 단위큐브(anchor cube)로부터 시작되며, 위험 정보는 인접 큐브로 확산되면서 파동형 연산 파장을 형성한다. 각 단위큐브는 실시간으로 중요도(importance score)를 재계산받고, 이 점수가 높을수록 연산 우선순위가 높아져 위험 요소가 빠르게 처리된다.

[0214] 시각화 측면에서는, 위험도가 높은 단위큐브가 붉은색과 높은 불투명도로 표현되며, 저위험 영역은 푸른색과 낮은 투명도로 나타나 위험 분포가 직관적으로 구분된다. 연산의 확산은 중심 anchor로부터 외부로 퍼져나가는 파동형 애니메이션으로 실시간 구현되며, 중요도 점수가 일정 임계값(예: 0.85)을 초과하는 큐브는 경고 아이콘 또는 사운드 트리거를 통해 사용자에게 시각적·청각적으로 경고된다. HMI(사용자 인터페이스)를 통해 사용자는 중요도 필터링 임계값 조절, 시점(TSU)의 이동, 경로 알고리즘 비교 등 다양한 조작을 실시간으로 수행할 수 있다.

[0215] AI 연산 구조는 이 시뮬레이션의 핵심 기능 중 하나로, anchor-to-anchor 기반의 연산 예측을 통해 향후 연산 집중 가능성이 높은 영역을 선제적으로 파악하고, AI 강화학습 정책은 탈출 경로, 위험 회피 방향 등을 실시간으로 제안한다. 이러한 정책은 반복 훈련을 통해 성능이 지속적으로 개선되며, 시뮬레이션이 끝난 후에는 위험 확산 패턴을 학습 데이터로 축적하여 더욱 정교한 대응 시나리오 생성을 가능하게 한다.

[0216] 이러한 연산 기반 위험 탐지 흐름은 도 15에 도시된 바와 같이 3차원 시각화 결과로 구체화된다. 도 15는 본 실시예(재난 대응 훈련 시뮬레이션)에서 구성된 Q-Twin™ 시스템 내 단위큐브 수준의 위험도 분포를 시각적으로 표현한 예시로, 센서 기반 메타데이터(연기, 온도, 진동)를 기반으로 실시간 연산된 importance score를 기준으로 각 큐브의 위험 수준을 시각화하였다. 주요 해석 포인트는 다음과 같다; 첫째, 색상 스케일은 각 단위큐브의 중요도 값을 0~1로 정규화한 후, 높은 위험도를 나타내는 큐브는 붉은 계열, 낮은 위험도는 푸른 계열로 시각적으로 구분되도록 구성되었다. 둘째, 위험 구역 강조는 고위험 메타데이터가 복합적으로 감지된 큐브(예: 온도 상승 + 연기 농도 증가 + 진동 이상)의 경우, 시각적으로 더욱 강조되어 실시간 모니터링 중 즉각적인 대응 유도가 가능하다. 셋째, Anchor 기반 확산 구조는 중심 단위큐브(좌표 4,4,4)로부터 연산이 개시되어, 메타데이터 유사도 및 거리 기반으로 주변 큐브로 연산 파장이 확산되는 구조를 재현한 것으로, 병렬 연산의 흐름을 직관적으로 이해할 수 있도록 시각화하였다. 마지막으로, 본 도면은 3D 공간 내 큐브 위치와 연산 상태를 동시에 표현함으로써, 실제 공간 내 위험 분포의 실시간 추적, 리소스 재배치 전략 수립, AI 경로 재계산 등과 같은 복합적 시뮬레이션 기능을 통합적으로 수행할 수 있는 Q-Twin™ 시스템의 실시간성·정밀성·융합성을 종합적으로 보여주는 시각적 결과물이라 할 수 있다.

[0217] 결과적으로, 이 재난 대응 훈련 시나리오는 다음과 같은 기술적 효과를 제공한다; 실시간 위험 예측이 가능하며, 메타데이터 기반 연산으로 위험 밀도와 확산 흐름을 사전에 예측할 수 있고, 시각적 인지 향상을 통해 히트맵, 색상, 애니메이션을 활용한 직관적 경고 체계를 구현하며, 훈련 정책 개선 측면에서 강화학습 기반 AI가 피드백을 반영하여 반복적으로 전략을 고도화할 수 있으며, 응급 경로 탐색 지원 기능을 통해 anchor-to-anchor 연결 구조를 기반으로 최적 탈출 경로 및 대응 시점을 자동 연산하고 제시할 수 있다. 이러한 실시예는 Q-Twin™ 시스템이 단순한 3D 시각화 도구를 넘어서, 실시간 재난 시뮬레이션, 정책 평가, AI 학습 기반 시나리오 자동 생성이 가능한 디지털 트윈 기반의 고차원적 훈련 플랫폼으로 활용될 수 있음을 입증한다.

실시예 5

[0218] AI 예측 시각화 시나리오 - 미래 고밀도 연산 영역의 강조 표현 : 본 시나리오는 Q-Twin™ 시스템의 시계열 예측 기능을 기반으로, AI(강화학습 기반 예측 모델)가 향후 TSU(Time Singularity Unit)에서 연산 집중도가 높을 것으로 판단한 미래 anchor cube를 시각적으로 강조하는 방식으로 구성된다. Q-Block은 TSU₀부터 TSU_n까지 순차적으로 적층되며, AI는 현재까지의 anchor-to-anchor 흐름, 메타데이터 누적 경향, 정보 전이 이력 등을 바탕으로 미래 시점의 연산 밀도 집중 영역을 예측한다. 이 예측 결과는 anchor-to-anchor 경로 상의 다음 anchor 후보 중, importance score의 추세가 가파르게 증가하는 방향성을 기준으로 산출된다. AI 예측 기반 시각화 구성은 다음과 같다;

[0219] Q-Twin™ 시스템에서의 AI 예측 시각화는 시공간 연산 흐름의 미래 예측 결과를 직관적으로 전달하기 위해 다양한 시각적 효과를 통합적으로 적용한다. 먼저, AI가 고밀도 연산 영역으로 예측한 미래 anchor cube는 Glow 또는 Halo 효과를 통해 해당 위치를 강조하여 시각적으로 부각되며, 이와 동시에 해당 anchor cube 주변에는 Particle Emission 애니메이션이 적용되어, 정보 에너지의 누적 상태와 예측된 연산 파동의 확산이 동적으로 표현된다. 또한, 예측 중요도에 따라 점진적으로 진행되는 색상 스펙트럼 - 예를 들어 연한 노랑에서 진한 주황, 최종적으로 붉은색에 이르기까지 - 이 적용되어 예측 신뢰도와 연산 긴급도가 시각적으로 명확히 구분된다. 마지막으로, 예측이 도달하는 미래 TSU(Time Singularity Unit) 값은 해당 큐브 상단 또는 인터페이스의 UI 패널에 TSU index로 실시간 표기됨으로써, 사용자가 예측된 시점을 명확히 인지하고 시뮬레이션의 시간적 맥락을 이해할 수 있도록 설계되어 있다. 이러한 시각화 전략은 Q-Twin™ 시스템의 자율적 예측 기능을 보다 직관적이고 정보적으로 전달함으로써, 시뮬레이션 기반 의사결정 환경의 신뢰성과 반응성을 함께 향상시킨다.

[0220] AI 학습 연계 및 활용 흐름은 다음과 같은 단계로 구성된다; 먼저, 현재 시점까지 누적된 각 anchor cube의 메타데이터(density, energy, overlap, TSU delta 등)를 기반으로 입력 벡터를 구성하고, 이를 강화학습 에이전트(policy network)에 전달하여 미래 TSU에서 연산 밀도가 집중될 가능성이 높은 anchor cube 후보를 예측한다. 에이전트는 사전에 정의된 reward weight를 기준으로 정책 기반의 예측을 수행하며, 이 결과는 Q-Twin™의 시각화 인터페이스와 연동되어 실시간으로 사용자 화면에 반영된다. 이후 사용자는 예측된 anchor 위치 및 중요도 정보를 실제 연산 결과와 비교할 수 있으며, 이를 통해 피드백을 제공하게 된다. 이 피드백은 다시 강화학습 에이전트의 정책 개선에 활용되어, 연산 예측 정확도를 점진적으로 향상시키는 순환적 학습 구조를 형성한다.

[0221] 도 16은 Q-Twin™ 시스템이 강화학습 기반 AI 정책을 통해 미래 TSU에서의 고밀도 연산 가능성이 높은 anchor cube를 예측하고, 이를 시각적으로 강조한 시뮬레이션 결과를 보여주는 대표적 예시이다. 붉은색 계열로 표현된 큐브는 AI가 중요도(importance score)가 높은 영역으로 판단한 단위큐브이며, 중앙 상단의 시안색 큐브는 AI가 미래 anchor 후보로 예측한 핵심 위치를 나타낸다. 각 큐브의 색상 강도는 해당 위치의 연산 밀도 예측값의 신뢰도와 긴급도를 반영하고 있으며, 이로써 사용자는 향후 시점에서 연산 집중 구간이 어디에 형성될지를 직관적으로 인지할 수 있다. 도 16은 단순히 현재 상태를 묘사하는 기존 디지털 트윈의 한계를 넘어서, Q-Twin™ 시스템이 예측 중심의 디지털 트윈 구조로 확장되었음을 보여주는 대표적 사례로, 향후 자원 재배치, 연산 흐름 최적화, 긴급 상황 대응 전략 수립 등에 있어 핵심 기술적 기반이 될 수 있다.

표 41

[0222] Q-Twin™ 기반 AI 예측 시각화의 기술적 효과 및 응용 의의

항목	효과 요약
미래 고밀도 연산 예측	연산 파장 확산 방향을 사전에 예측하여 GPU 자원 할당 및 스톱 예약 가능
직관적 시각화 표현	Glow, Particle, 색상 효과를 통한 실시간 예측 신뢰도 시각화
학습-예측-검증 루프 통합	예측 결과에 대한 사용자 피드백 기반 강화학습 정책 개선 루프 구축
연산 병목 사전 대응	연산 집중 구간의 발생을 미리 파악하여 시뮬레이션 리소스 및 경로 재배치 등 전략 수립 가능

[0223] 결론적으로, 본 실시예는 Q-Twin™ 시스템이 단순히 현재 상태를 정적으로 표현하는 기존 디지털 트윈의 한계를 넘어, 강화학습 기반 AI 정책을 통해 미래 TSU(Time Singularity Unit)에서의 연산 집중 구간을 사전에 예측하고, 이를 Particle 또는 Glow 효과를 통해 직관적으로 시각화함으로써, 디지털 트윈이 갖추어야 할 예측성(predictiveness)과 자율성(autonomy)을 실질적으로 구현하고 있음을 보여주는 대표적 사례라 할 수 있다. 특히 “AI 예측 시각화” 시나리오에서는, 현재까지 축적된 중심 단위큐브(anchor cube)의 메타데이터 시계열을 학습한 AI 에이전트가 다음 TSU 또는 미래 시점에서의 anchor 후보를 추론하고, 그 결과를 Glow 효과, Particle 강조, 색상 변화 등으로 실시간 시각화함으로써, 사용자가 예측 흐름과 실제 결과를 직관적으로 비교하고 피드백까지 제공할 수 있는 인터랙티브 환경을 제공한다.

[0224] 이는 단순히 과거와 현재 상태를 복제하는 시뮬레이션에 머무르지 않고;

[0225] 가. 미래 연산 흐름을 능동적으로 예측하고,

[0226] 나. 그 결과를 시각적으로 전달하며,

[0227] 다. 사용자 피드백을 기반으로 AI 정책을 지속 개선하는 자율적 구조를 구현한 것이다. 이러한 구조는 Q-Twin™이 단순한 시뮬레이션 엔진을 넘어, 고성능 AI 예측 기반 실시간 디지털 트윈 시스템으로 진화할 수 있음을 구체적으로 입증하는 핵심 사례라 할 수 있다. 나아가 이 기술은 연산 병목의 사전 회피, 자원 재배치, 실시간 연산 스케줄 최적화 등에도 적용 가능하여, 산업 현장, 재난 대응, 도심 운영 등 고차원적 의사결정 지원에 있어서도 실질적인 기여가 가능하다.

실시예 6

[0228] 도심지 드론 관제 상황 시나리오 : 본 실시예는 본 발명의 Q-Twin™ 시스템이 도심지 내 드론 운항을 실시간으로 시각화 및 관제하는 상황에 적용되는 구체적인 사례에 관한 것이다. 가상의 도심 지역인 ‘U-City’ 내에서 복수의 드론(UAV)이 동시에 운항 중인 상황을 가정한다. 이때, Q-Twin™ 시스템은 해당 도심의 3차원 공간을 일정한 크기의 단위큐브로 분할하여 Q-Block을 형성하고, 각 단위큐브에 실시간으로 위치 정보, 고도 정보, 기상 정보, 장애물 정보 및 공중권 설정 데이터를 매핑한다. 각 드론은 자신의 현재 위치 및 이동 계획 정보를 시스템에 전송하고, Q-Twin™은 이 정보를 기반으로 anchor-to-anchor 방식의 예상 경로를 산출한다. 산출된 경로상의 단위큐브들에 대해 다음과 같은 실시간 시각화 및 분석이 수행된다;

[0229] 가. 위험도 시각화 : 예상 경로상 단위큐브의 정보 밀도, 과거 충돌 빈도, 인접 UAV 간 최소 거리 등을 종합하여 위험도 점수를 산정하고, 이를 색상(예: 녹색~적색 그라데이션)으로 시각화한다.

[0230] 나. API 바인딩을 통한 시각화 엔진 연동 : Q-Twin™은 Unreal Engine 또는 Unity와 연동되어, 위 정보를 실시간으로 3D 시각화한다. 이때 각 단위큐브는 외부 엔진 내 렌더링 객체로 바인딩되어 있으며, 드론의 이동 경로 및 해당 경로 상의 위험도, 속도, 고도 변화 등이 실시간으로 반영된다.

[0231] 다. 경로 재조정 및 회피 판단 : 특정 단위큐브에 대해 위험도가 임계치를 초과할 경우, 시스템은 대체 anchor-to-anchor 경로를 제안하고, 해당 정보를 즉시 외부 엔진에 반영함으로써 시각적으로 회피 경로를 표현한다.

[0232] 라. 관제자 인터페이스 연동 : 관제자는 시각화된 3D 화면에서 각 드론의 예상 경로, 위험도, 충돌 가능 구역 등을 직관적으로 파악할 수 있으며, 필요 시 특정 단위큐브를 클릭하여 상세 메타데이터를 확인하거나, 수동으로 경로 우선순위를 조정할 수 있다.

[0233] 이와 같은 시스템은 복잡한 도심 항로 내 다수 드론의 충돌 방지, 비행 효율성 제고, 공중권 침해 예방 등에 효과적으로 기여할 수 있으며, 향후 스마트시티 기반 드론 교통 관리 시스템(UAM-TMS)에 직접 적용 가능하다. 한편, 도 9는 본 실시예와 같이 도심지 드론 운항 시나리오에 Q-Twin™ 시스템이 적용된 실제 시각화 결과를 도시한 것으로서, 좌측에는 $z=4$ 평면 상 단위큐브들의 정보 밀도 분포가 색상으로 시각화되어 드론의 경로 상 복잡도와 위험도를 직관적으로 판단할 수 있으며, 우측에는 anchor-to-anchor 방식으로 예측된 경로상 위험도 점수가 시각화되어 향후 충돌 가능성, 경로 중첩 현상, 연산 집중도 등을 사전에 식별하고 회피 경로를 설정하는 데 효과적으로 활용된다.

[0234] 도 17은 본 발명의 Q-Twin™ 시스템이 Unreal Engine 또는 Unity와 연동되어 실시간 시각화를 수행하는 예시를 도시한 것이다. 본 도면은 $z=4$ 평면을 기준으로 생성된 Q-Block의 단위큐브 단면을 3차원 공간상에 시각화한 결과로, 좌측 화면은 각 단위큐브의 정보 밀도(density)를 기반으로 색상으로 표시하고 있으며, 이를 통해 데이터 집중도나 상호작용 빈도 등의 공간적 특성이 한눈에 확인된다. 우측 화면은 중심 단위큐브(anchor cube, 예: (4,4,4))를 기준으로 설정된 anchor-to-anchor 경로 상에 위치한 단위큐브들의 예측된 위험도 점수를 시각화한 것이다. 위험도가 높은 큐브는 강렬한 색상으로 강조되어, 자율비행체 경로 설정, 도심지 충돌 회피, 미래 TSU 상의 연산 집중 예측 등에 활용될 수 있다. 이와 같은 시각화 구조는 Q-Twin™ 시스템이 제공하는 디지털 트윈 연산 결과를 직관적으로 표현하며, 단위큐브 간 연산 흐름, 메타데이터 이동, 시공간상 변화 양상 등을 3D 기반으로 명확하게 분석할 수 있도록 한다. 따라서 본 시스템은 도심 드론 관제, 메타버스 도시 설계, 실시간 사고 예측 등 다양한 시나리오에서 효과적으로 적용 가능하다.

산업상 이용가능성

[0235] 본 발명인 Q-Twin™: Q-Block 기반 디지털 트윈 생성 및 시계열 시뮬레이션 시스템은 시공간 단위의 정보 연산을 정량화·시각화하고, AI 기반 강화학습을 통해 미래 시점의 연산 흐름까지 예측할 수 있도록 설계된 고도화된 디지털 트윈 시스템이다. 이와 같은 구조는 기존 디지털 트윈 기술이 제공하지 못했던 예측성(predictiveness)과 자율성(autonomy)을 실현함으로써, 다양한 산업군에서 높은 수준의 실시간성 및 전략적 의사결정 기반 기술로 활용될 수 있다. 특히 본 발명은 단순히 과거와 현재 상태를 모사하는 것에 그치지 않고,

강화학습 기반의 시계열 학습 구조를 통해 미래 TSU(Time Singularity Unit)에서의 연산 밀도 집중 가능성을 사전에 예측하고, 이를 시각적으로 직관화함으로써 다음과 같은 산업적 활용 가능성을 제공한다;

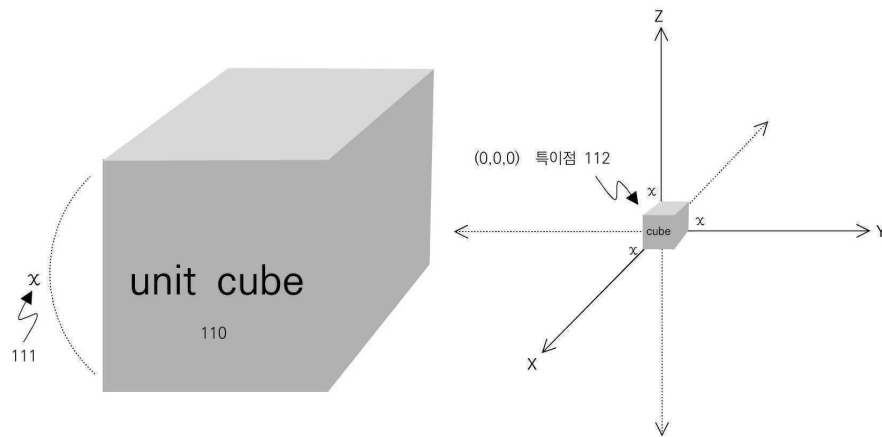
- [0236] 1. 도시 기반 시설 관리 및 스마트 시티 운영
- [0237] Q-Twin™ 시스템은 도시 내 공중권, 교통 흐름, 에너지 소비 패턴 등을 시공간 단위로 디지털화하고, 미래 연산 집중 지점을 예측함으로써 교통 체증, 에너지 피크 시간대, 항공 모빌리티(UAM) 운항 경로 충돌 가능성 등을 사전에 시뮬레이션하고 대응할 수 있다.
- [0238] 2. 자율비행체(UAV, UAM) 항법 및 경로 계획
- [0239] 강화학습 기반 anchor-to-anchor 연산 흐름 예측 기능은, 실시간 비행체 경로 재계산과 잠재적 충돌 회피, 연산 리소스 최적 배치를 위한 정책 시드를 제공함으로써, 자율비행체의 안전성과 항로 효율을 극대화하는 데 기여할 수 있다.
- [0240] 3. 산업 설비 및 공정의 위험 예측 및 자원 재배치
- [0241] 대규모 공정 또는 제조 라인에서는 연산 집중도가 높아지는 지점에서 병목이나 에러가 발생할 수 있다. 본 발명은 이러한 고밀도 연산 예측을 통해 사전 대응적 리소스 스케줄링 및 시뮬레이션 기반 공정 재설계를 가능하게 한다.
- [0242] 4. 재난 대응 및 시뮬레이션 훈련
- [0243] 연기, 온도, 진동 등의 실시간 센서 정보를 메타데이터화하여 시공간 단위 위험 지수를 정량화하고, 이후 연산 흐름을 통해 고위험 확산 경로를 예측함으로써, 소방·재난 대응 훈련의 정밀도 및 실효성을 향상시킬 수 있다.
- [0244] 5. AI 기반 도시 예측 엔진 및 정책 시뮬레이션
- [0245] Q-Twin™은 정책 실험 플랫폼으로서도 활용될 수 있다. 특정 규제 또는 개발 정책이 적용되었을 때 발생할 수 있는 정보 밀도 변화, 교통량, 공중권 이용 흐름 등을 예측하여, 의사결정 과정에서 정량적 근거 기반 정책 시나리오 검증이 가능하다.
- [0246] 결론적으로, 본 발명은 단순 모사의 범주를 넘어, 예측 가능한 미래 시점의 정보 흐름을 학습하고 시각적으로 표현하며, 사용자와의 상호작용을 통해 지속적으로 정책을 강화할 수 있는 진화형 디지털 트윈 기술이다. 이는 고성능 GPU 연산 환경 및 실시간 AI 학습 구조와 결합하여, 시계열 예측 기반 디지털 트윈의 핵심 플랫폼 기술로서 향후 메타버스, UAM, 공중권, 스마트 인프라, 재난 대응, 도시 정책 실험 등 다수의 미래 지향적 산업 영역에서 확장성과 실용성을 동시에 입증할 수 있는 높은 산업적 파급력을 가진다.

부호의 설명

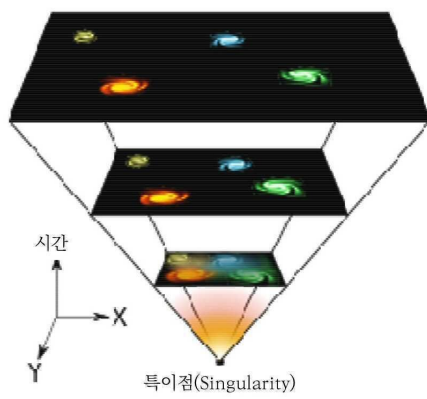
- [0247] 110 단위큐브(Unit Cube)
- 111 일측의 길이
- 112 특이점(Singularity)
- 120 Q-Block
- 121 중심 단위큐브(Anchor Cube)
- 122 TSU(Time Singularity Unit)

도면

도면1

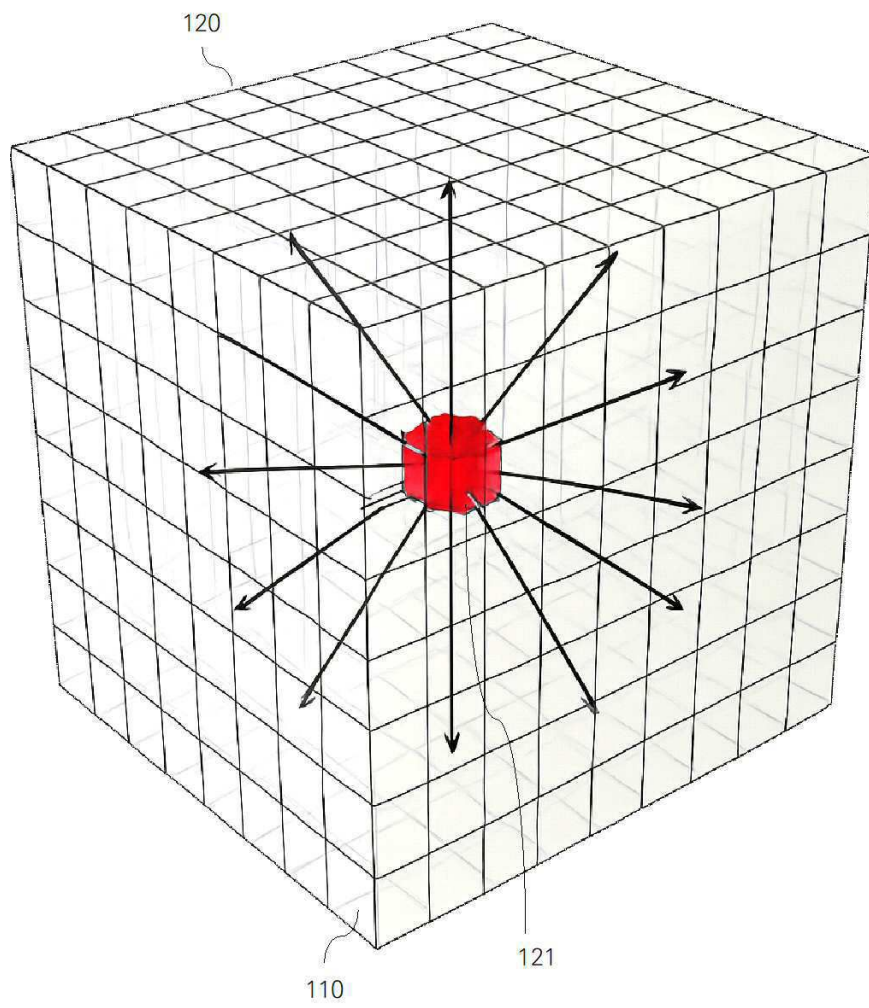


도면2

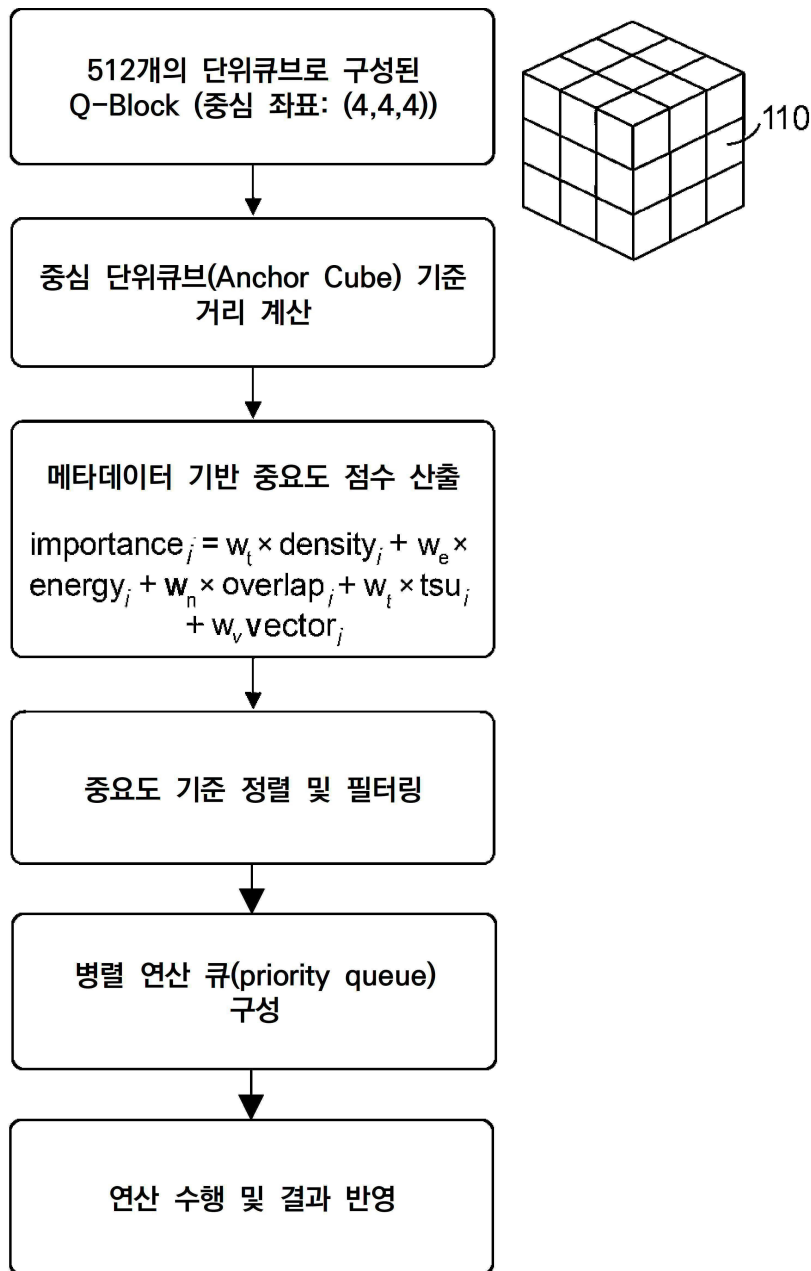


Source: Mrmw, "Universe Expansion", Wikimedia Commons, 16 November 2019(CC BY-SA 4.0)
https://commons.wikimedia.org/wiki/File:Universe_expansion-en.svg

도면3

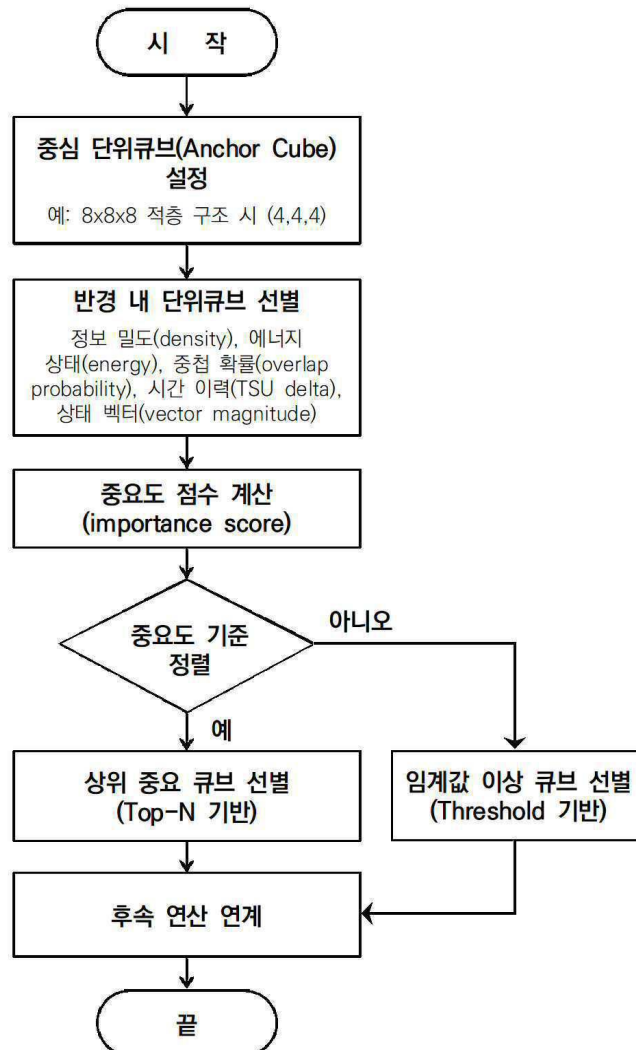


도면4



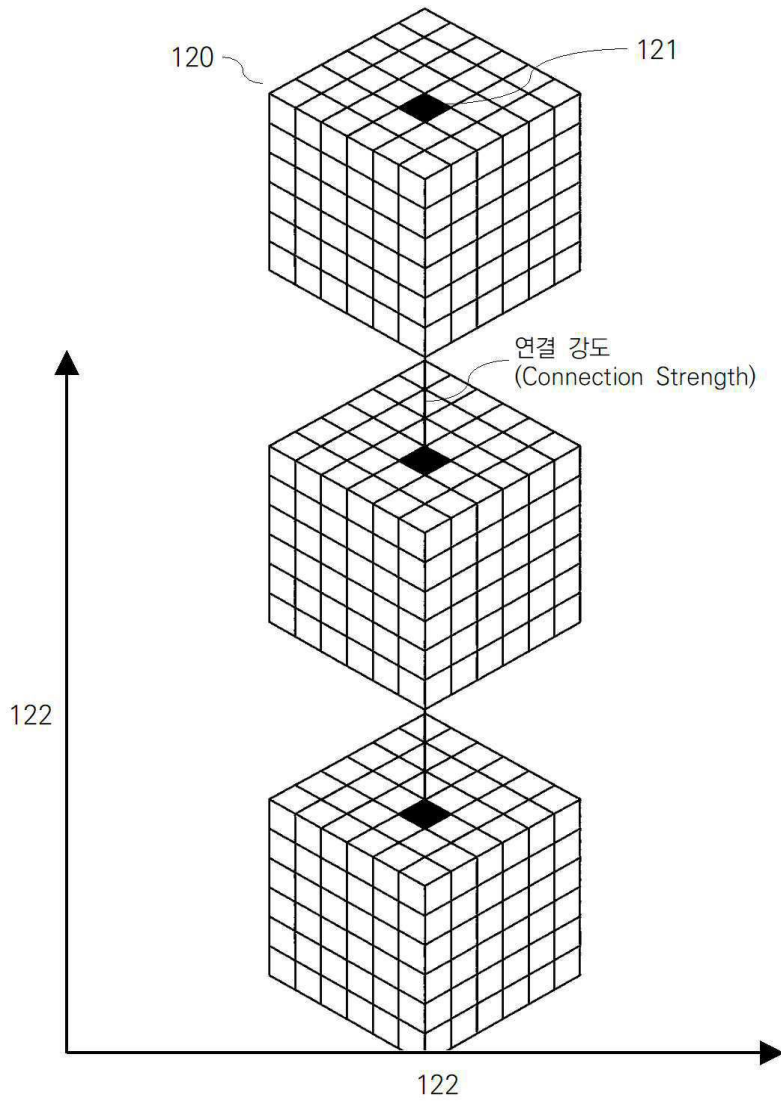
도면5

메타데이터 기반 가중치 필터링 알고리즘 흐름

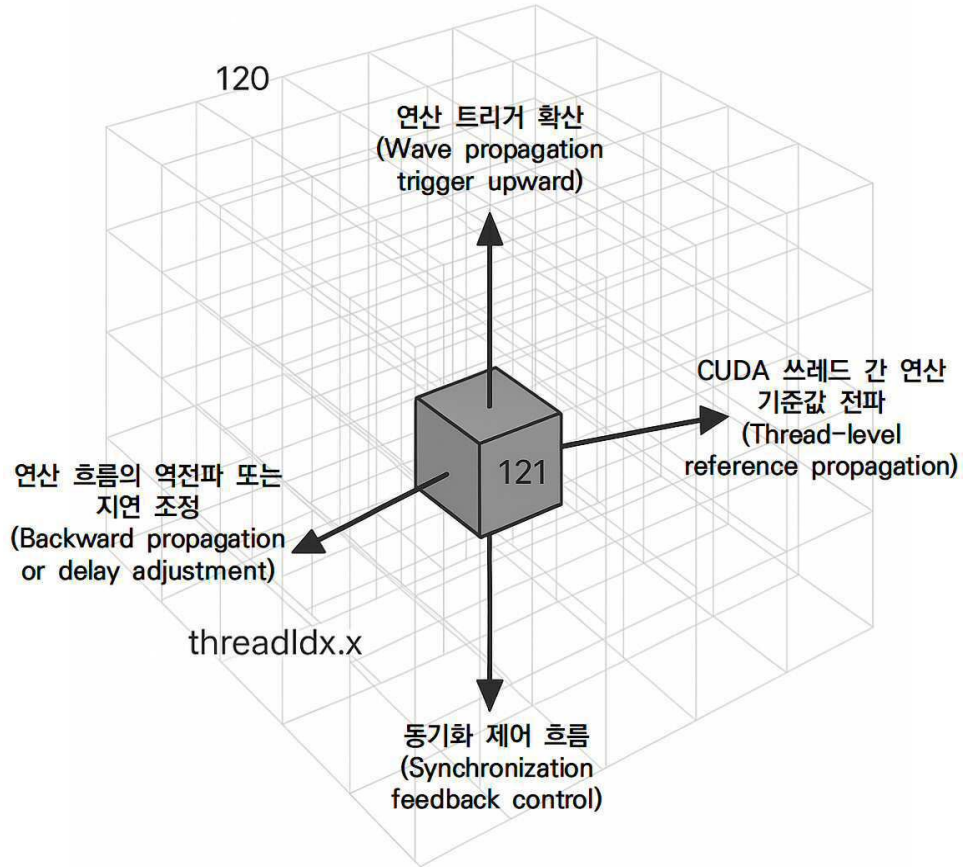


도면6

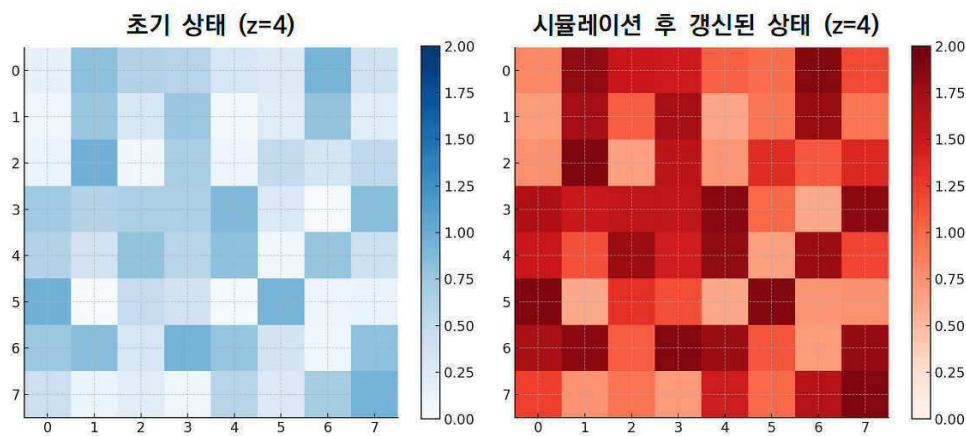
중심 단위큐브 간 연결
(Anchor-to-anchor connection)



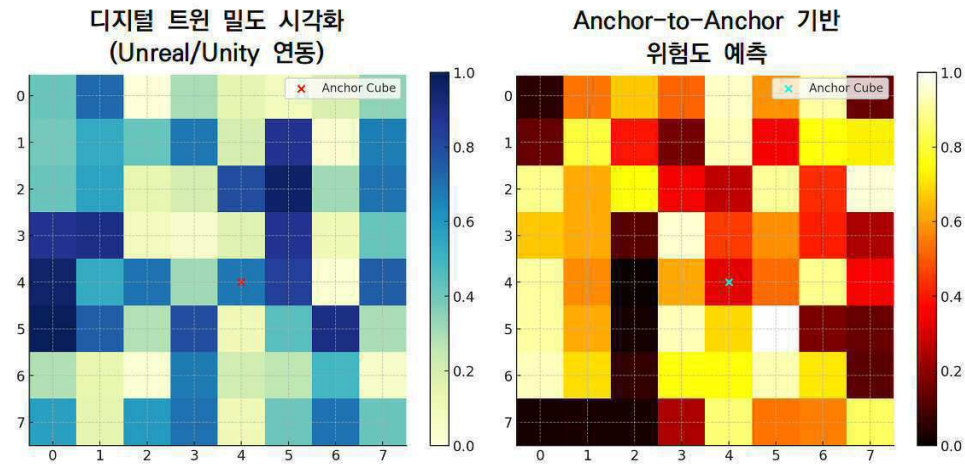
도면7



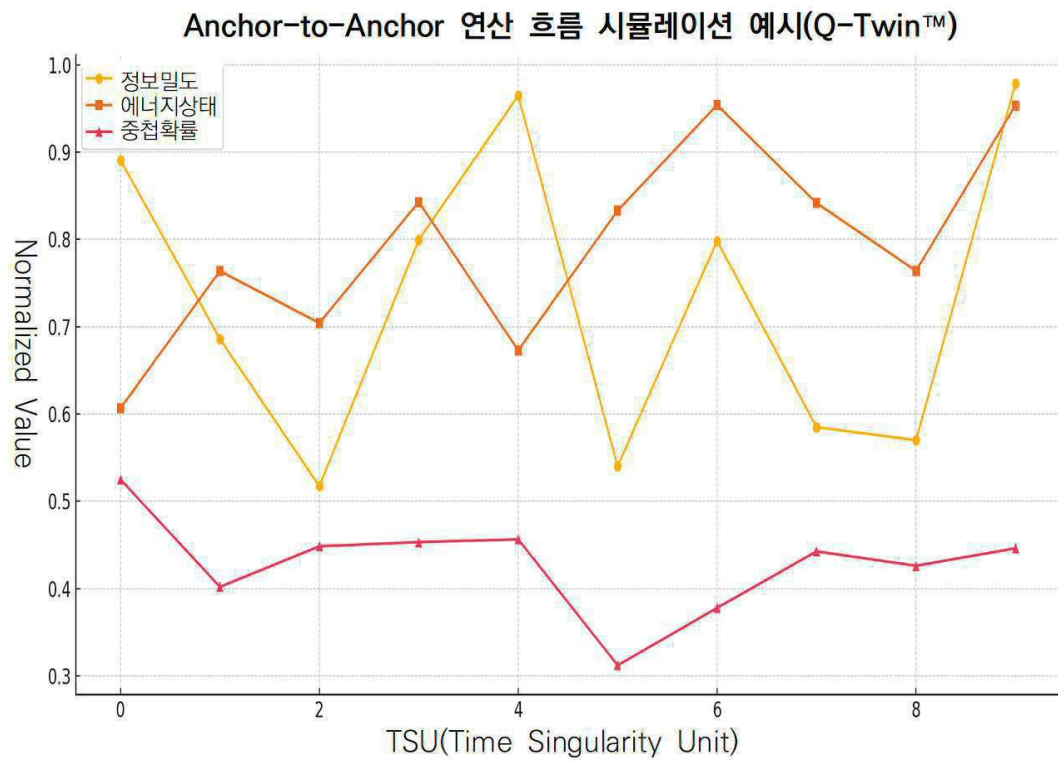
도면8



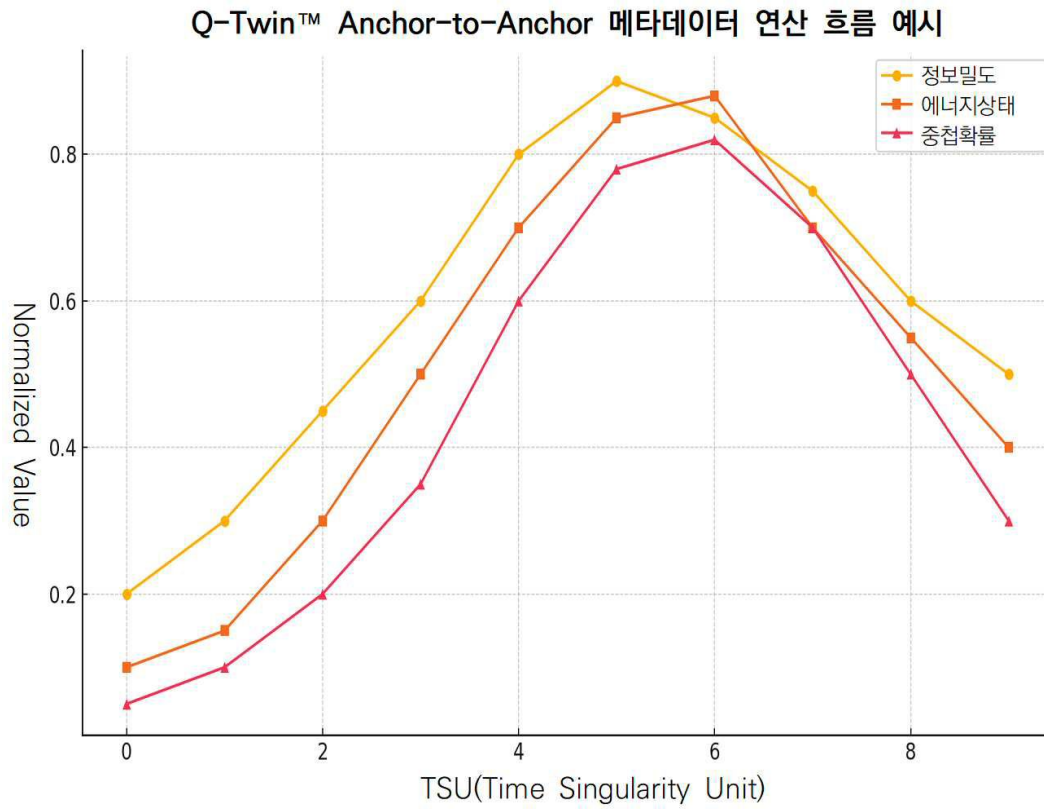
도면9



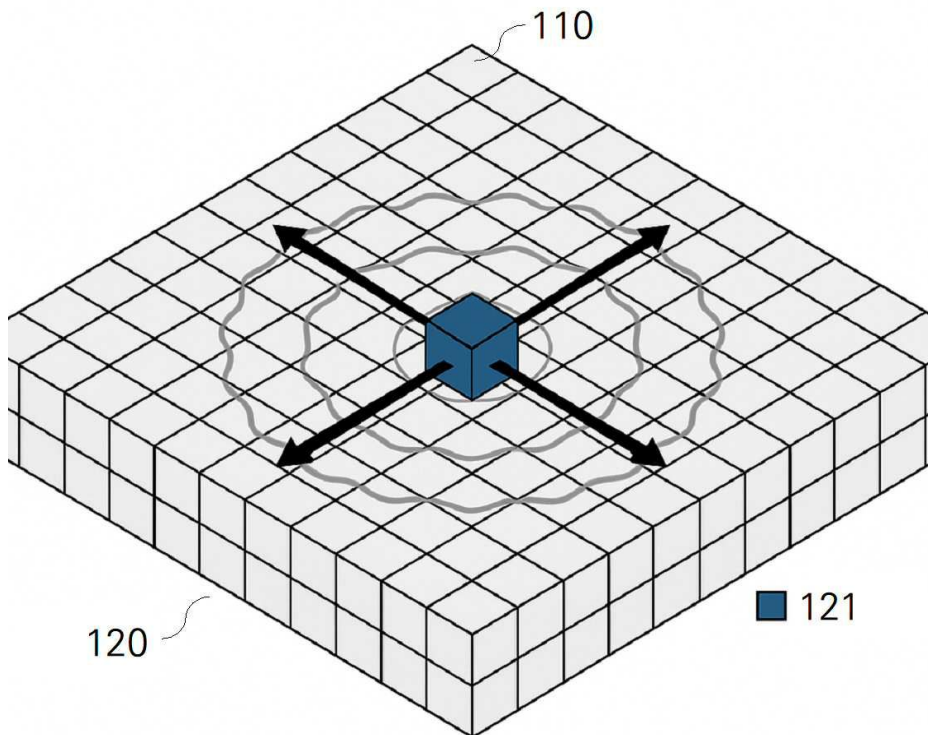
도면10



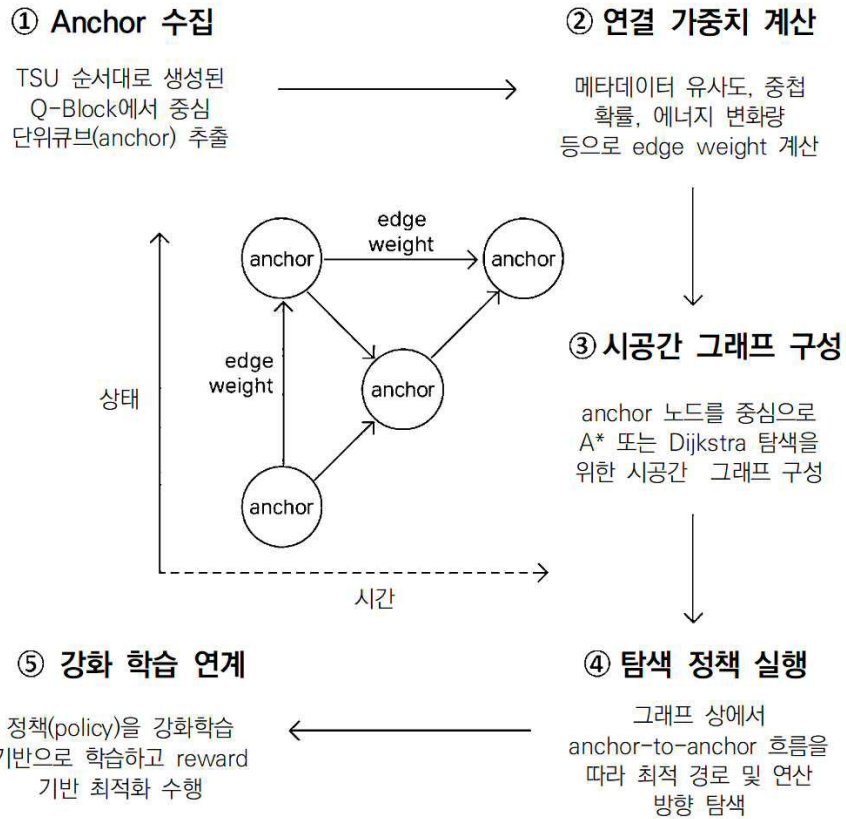
도면11



도면12

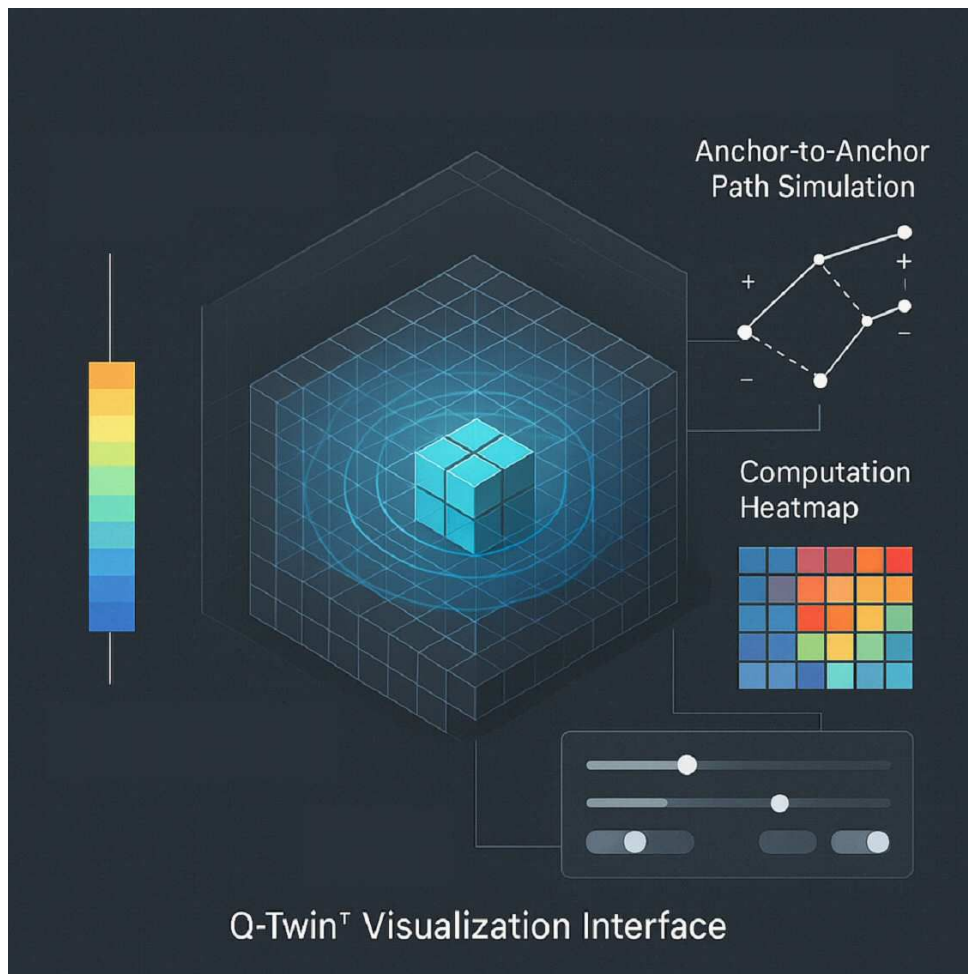


도면13



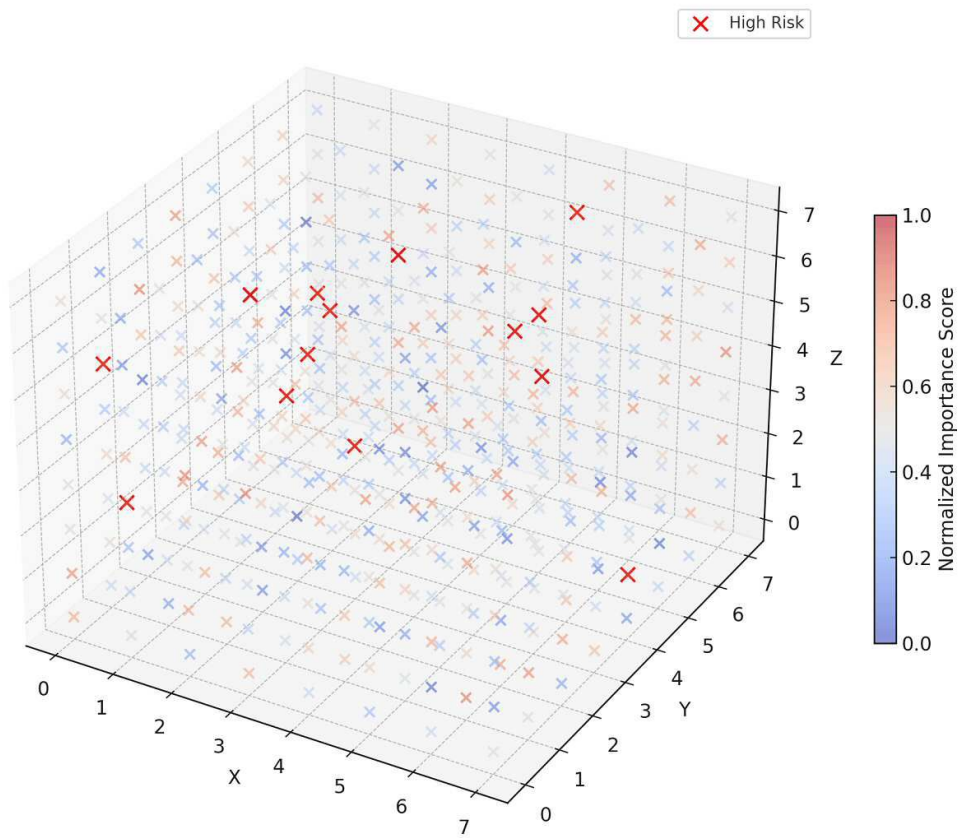
Anchor-to-Anchor 기반 시공간 그래프 탐색 및 정책 학습 흐름도

도면14

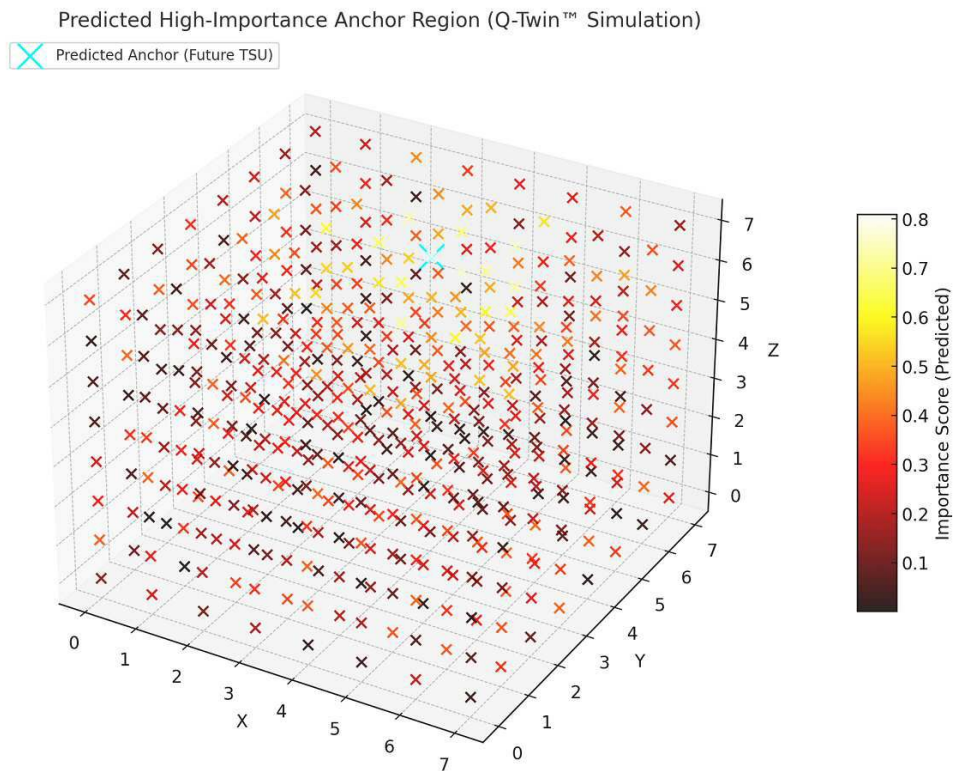


도면15

Q-Twin™ 재난 시뮬레이션 결과 - 단위큐브 위험도 분포



도면16



도면17

