# 멀티쓰레드 게임 엔진 아키텍처

멀티코어 프로세서가 피씨용으로 사용된 지 일년을 훌쩍 넘긴 데다 엑스박스 360 은 이미 수 백만 달러가 판매되었음에도 멀티코어 플랫폼용의 게임엔진 개발에 관한 지식과 개발은 여전히 부족한 실정이다. 이 글에서는 아키텍처 수준에서 게임엔진의 병렬성에 대한 관점을 제공하고자 한다.

갭 앤 레이크[1]가 보여준 바와 같이 각각의 구성요소 수준에 있는 멀티쓰레딩을 살펴보는 것보다 전체 게임루프에 병렬성을 추가하는 것이 더 나을 것 같다. 이제 살펴볼 상이한 세 종류의 모델은 기본 게임루프에 사용되는 쓰레드 지원형 아키텍처 모델로써 이들을 확장성, 예상 성능 같은 특성에 비교할 것이다.

주로 두 가지 방식으로 프로그램을 동시적 발생 부분으로 분해할 수 있다. 그 두 방식이란 함수 병렬성, 즉 프로그램을 동시적 발생 작업으로 나누는 방식과 데이터 병렬성, 즉 병렬로 동일 작업들을 수행할 수 있는 데이터 세트를 찾으려는 방식이다. 비교되는 세 모델 가운데 두 가지는 함수 병렬적이고 한 가지는 데이터 병렬적이다.

## 동기적 함수 병렬 모델

게임루프에 병렬성을 포함시키는 한 가지 방법은 기존 루프에서 병렬 작업들을 찾는 것이다. 병렬 작업들 간의 통신 필요성을 최소화하기 위해 이 작업들은 서로 독립적인 것이 더 낫다. 이런 예는 애니메이션을 계산하면서 물리작업을 수행하는 것이다. 그림 1 은 이 기법을 이용해 병렬화 한 게임루프를 나타낸다.

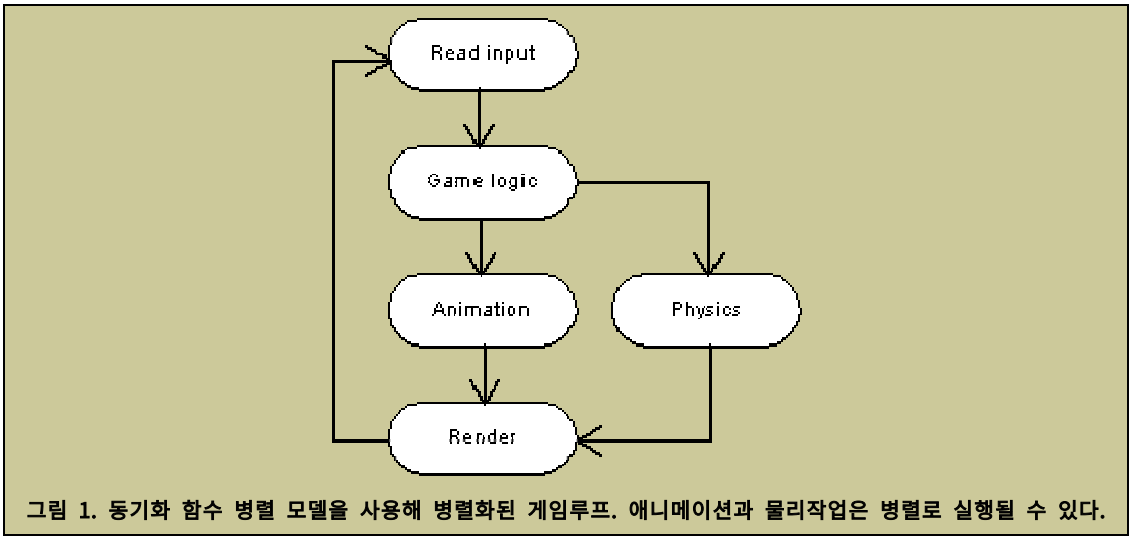


**그림 1. 동기화 함수 병렬 모델을 사용해 병렬화된 게임루프. 애니메이션과 물리작업은 병렬로 실행될 수 있다.**

코스타[2]가 제시한 방법은 이런 종류의 아키텍처 확장을 자동화하는 것이다. 말하자면 기능성을 작은 작업들로 나누고 종전의 작업(그림 1 의 그래프 같은)들로 이뤄진 그래프를 구축한 후 하나의 프레임워크에 이러한 작업 종속형 그래프를 적용한다. 프레임워크는 돌아갈 적절한 작업들의 스케줄을 짜면서 사용 가능한 프로세서 코어의 양을 고려한다. 그러한 프레임워크를 사용하는 대신 당신이 지원하기로 계획한 각 코어의 양에 맞는 당신의 코드를 재 작성할 수 있다.

두 가지의 함수 병렬 모델에 주로 관심이 가는 부분은 이 두 모델의 지원 가능한 코어의 수에 상한이 있다는 점이다. 이것은 엔진에서 찾을 가능성이 있는 병렬작업이 얼마인지에 대한 한도이다. 의미 있는 작업의 수는 지극히 적은 작업을 쓰레드하면 부정적 결과를 나올 수 있다는 사실로 인해 감소한다. 동기화 함수 병렬 모델에는 또 하나의 한계가 있다. 병렬 작업들은 서로에 대한 의존성이 거의 없어야 한다는 점이다. 예를 들어 만약 렌더링이 물리작업의 물체좌표계를 필요로 한다면 렌더링 작업과 병렬로 물리작업을 실행시키는 것은 현명하지 않다.

이 모델의 예상 성능은 게임 루프에 있는 최장의 실행경로 길이에서 직접 확인할 수 있다. 이 실행경로의 길이는 루프에 있는 병렬성의 양에 직접 연결된다. 이 모델은 일반적으로 세 가지 모델의 최소 병렬성을 허용하기 때문에 모델의 성능을 통한 예상은 동일할 수 있다.

동기화 함수 병렬 모델은 병렬로 실행되는 작업들 간에 연결성이 거의 없다고 가정하기 때문에 기존의 구성요소는 많은 변화를 요구하지 않는다. 예를 들어 물리 구성요소 갱신의 실행이 사운드 믹서와 동시에 실행될 수 있는 작업이라면 구성요소도 작동에 특수한 지원을 요하지 않는다.

## 비동기 함수 병렬 모델

갭과 레이크가 제안한 대체함수 병렬 모델은 이 모델이 게임루프를 포함하지 않는다는데 그 차이가 있다. 대신 게임을 진행시키는 작업들은 자체 속도로 갱신하며 사용 가능한 최근 정보를 사용한다. 예를 들어 렌더링 작업이 물리작업의 종료를 기다리지 않지만 가장 마지막에 완료된 물리 갱신만을 사용할 것이다. 이러한 방법을 사용함으로써 상호 의존적인 작업들을 효율적으로 병렬화할 수 있다. 그림 2 는 비동기 함수 병렬 모델의 예를 보여준다.

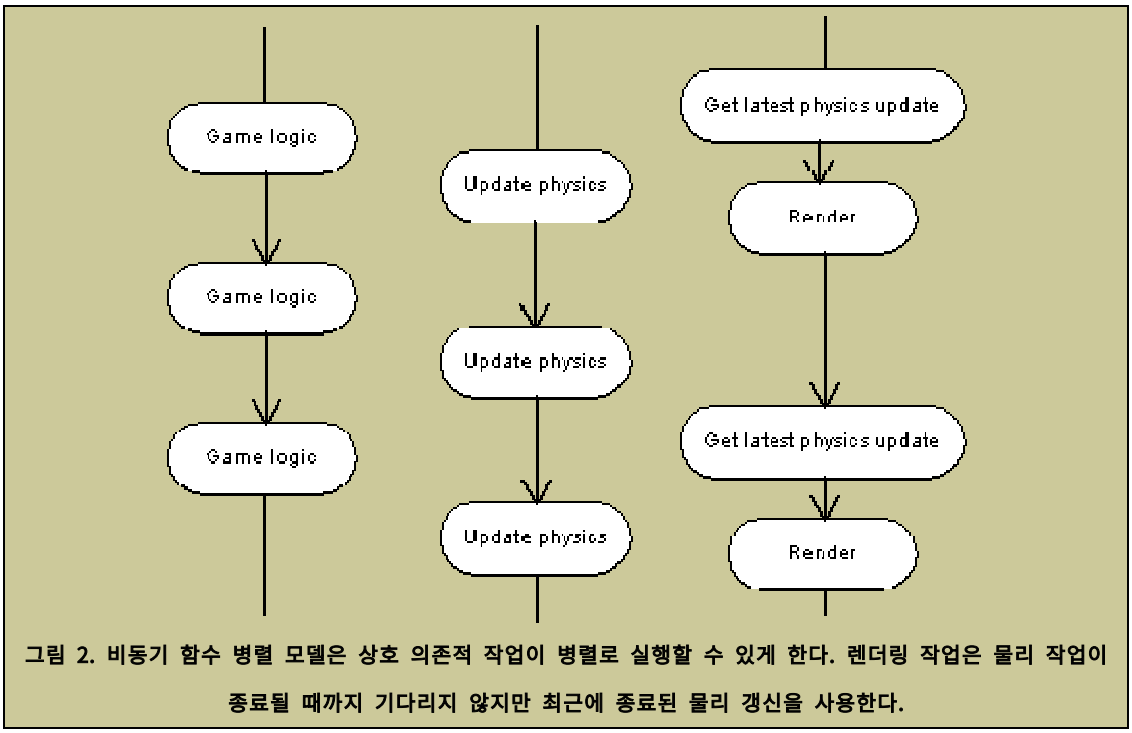


**그림 2. 비동기 함수 병렬 모델은 상호 의존적 작업이 병렬로 실행할 수 있게 한다. 렌더링 작업은 물리 작업이 종료될 때까지 기다리지 않지만 최근에 종료된 물리 갱신을 사용한다.**

동기화 모델의 경우처럼 비동기 함수 병렬 모델의 확장성은 엔진으로부터 찾을 수 있는 작업의 수에 따라 제한된다. 다행히도 사용 가능한 최근 정보를 사용함으로써 쓰레드 간의 통신은 쓰레드가 진정 의존적이어야 할 필요성을 효과적으로 감소시킨다. 따라서 물리작업은 쉽게 렌더링 작업과 동시에 실행될 수 있다. 렌더링 작업은 각 객체에 따른 좌표계를 얻기 위해 종전의 물리 갱신을 사용한다. 이를 토대로 비동기 모델은 늘어난 작업 양을 지원할 수 있고 따라서 동기화 모델의 경우보다 프로세서 코어의 양이 많아질 수 있게 된다.

갭과 레이크가 지적한 것처럼 이 모델의 작업 간 통신은 다른 모델에는 없는 시간성과 관련해 복잡한 문제를 드러낸다. 동시에 실행되는 세 작업, 즉 입력 작업, 게임 객체를 이동하기 위해 입력을 사용하는 물리 작업 및 객체를 그리기 위해 물리결과를 사용하는 렌더링 작업이 있다고 가정하자. 입력작업이 완료된 후 바로 물리 작업이 시작되고 이 작업이 완료된 후 렌더링 작업이 시작되는 것이 최적의 상황이다. 최악의 경우 렌더링이 시작된 후 물리작업이 완료되고 물리 작업이 시작된 후 입력 작업이 완료된다. 이렇게 되면 결과적으로 이 최적의 시나리오의 두 시간대 중 입력 대 표현 시간이 되며 이 시간은 각 프레임에서 최적 및 최악의 시나리오 사이에서 변동된다. 갭 및 레이크가 제안한 치유책은 다른 것보다 더 자주 실행 될 일부 작업들을 조절하는 것으로써 입력작업이 물리작업보다 두 번 이상 실행되게 하는 것 같은 방식이다. 이 방식이 문제를 경감시킬 수는 있겠지만 제거하지는 못할 것이다.

비동기 모델은 동시적 작업 간에 이뤄지는 동기화가 거의 혹은 아주 없다고 가정하기 때문에 이 모델의 성능은 프로그램의 순차적 부분만큼 제한적이지는 않다. 따라서 주요 성능 한계는 유용한 병렬 작업을 충분히 찾아낼 수 있는 능력에서 온다. 염두에 둬야 할 사항은 이 작업들의 균형이 적절해야 한다는 점이다. 말하자면 하나의 큰 작업과 몇 개의 자잘한 작업들이 있으면 성능 병목 현상을 의미할 수도 있다.

비동기 모델은 서로 직접적으로 연결되지 않은 작업들에 많이 의존할 뿐 아니라 가장 마지막으로 사용할 수 있는 정보를 사용한 통신에도 의존하기 때문에 동시적 구성요소가 이 모델을 기반으로 기능 할 수 있도록 변화가 요구되기도 한다. 적어도 각 구성요소는 최신의 상태 갱신을 알아보기 위해 쓰레드에 안전한 방식을 필요로 한다. 그러한 변화는 이뤄지기 쉽기 마련이며 해당 구성요소에 추가적 래퍼로써 추가되기까지 한다.

## 데이터 병렬 모델

병렬 작업의 탐색 외에 병렬로 동일한 작업을 수행할 수 있는 유사한 데이터 집합도 찾을 수 있다. 게임 엔진과 함께 그러한 병렬 데이터는 게임에 있는 객체들일 수 있다. 예를 들어 비행 시뮬레이션에서 모든 비행기들을 두 개의 쓰레드로 분리하기로 결정할 수 있다. 각 쓰레드는 절반의 비행기 시뮬레이션을 담당할 것이다. 엔진이 논리 프로세서 코어만큼 많은 쓰레드를 사용하는 것이 최적의 경우다.

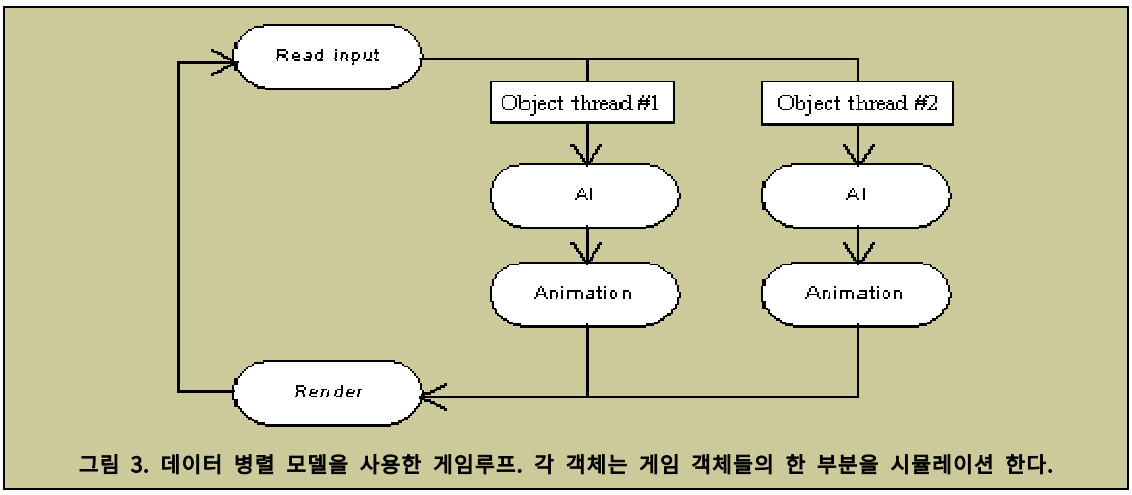


**그림 3. 데이터 병렬 모델을 사용한 게임루프. 각 객체는 게임 객체들의 한 부분을 시뮬레이션 한다.**

중요한 문제는 이 객체들을 어떻게 쓰레드들로 나누느냐 하는 점이다. 먼저 고려해 볼 수 있는 경우는 쓰레드들의 균형을 적절히 맞춤으로써 각 프로세서 코어는 꽉 찬 용량에 익숙해지도록 하는 것이다. 두 번째로 고려해볼 수 있는 것은 상이한 쓰레드에 있는 두 개의 객체가 상호 작용해야 할 때 어떤 일이 일어나는가 하는 것이다. 동기화 기본요소를 사용한 통신은 잠재적으로 병렬화의 양을 줄일 수 있다. 따라서 권고할 만한 실행계획은 비동기화 모델의 경우처럼 마지막으로 알려진 갱신을 사용함으로써 동반되는 메시지 교환을

사용하는 것이다. 쓰레드 간의 통신은 서로 가장 상호 작용을 많이 할 것 같은 객체들을 집단화함으로써 감소될 수 있다. 객체들은 그 이웃들과 접촉할 가능성이 많고 따라서 영역별로 객체를 집단으로 묶는 것이 한 가지 전략이 일 수 있다.

데이터 병렬 모델의 확장성은 훌륭하다. 객체 쓰레드의 양은 시스템이 실행하는 코어의 양에 자동으로 정해지고 게임루프에서 유일하게 병렬화하지 않는 부분들은 게임 객체를 직접 다루지 않는 것들이다(그림 3 의 입력 읽기와 작업 렌더). 함수병렬 모델은 소수의 코어들에서 대부분을 얻을 수 있지만 데이터병렬성은 수 십 개의 코어로 미래의 프로세서를 충분히 활용하는데 요구된다.

데이터 병렬모델의 성능은 게임엔진의 부분이 데이터에 의해 얼마나 크게 병렬화될 수 있는가에 직접 관련된다. 프로세서 코어의 양이 늘수록 엔진의 데이터 병렬부분이 실행시간은 줄어든다. 다행히도 이들은 게임엔진의 성능 과중 부분이기도 하다. 엔진이 게임루프 대부분에 맞게 데이터 병렬성을 사용할 수 있다면 데이터병렬 모델은 세 모델 가운데 최고의 성능을 제공할 것이다.

이 모델의 최대 약점은 데이터 병렬성을 지원하는 구성요소 구비가 요구된다는 점이다. 예를 들어 물리 구성요소는 여러 번의 물리 업데이트를 실행하고 분산된 이러한 쓰레드에 있는 객체로 충돌을 정확히 계산하는데 요구된다. 잠재적 해결책은 게임루프의 데이터 병렬부분에서 물리 계산을 따로 떼어두고 물리 구성요소의 내부 병렬화를 사용하는 것이다. 기타 다른 구성요소는 광범위한 변화를 요하지 않을 것이다. 예를 들어 그래픽 객체의 애니메이션을 계산하는 구성요소에 동시발생 쓰레드들 간의 상호작용이 없으면 병렬화를 위한 어떤 지원도 요구되지 않는다.

## 결론

동기화 함수 병렬 모델은 특별히 엔진구성요소의 변화를 요하지 않고 일반적 게임루프가 향상된 것이기 때문에 기존의 게임엔진에 어느 정도의 병렬성을 추가하는데 매우 적절하다. 이 모델의 향후 사용은 적절치 않다. 왜냐하면 확장 지원이 미약하고 병렬성의 정도가 낮기 때문이다.

비동기 함수 병렬 모델은 새로운 게임 엔진용으로 추천할 만한데 가능한 병렬성의 정도가 높고 기존의 구성 요소들이 아주 약간의 변화만을 필요로 한다는 사실 때문이다. 이 모델은 상대적으로 적은 수의 코어가 든 멀티코어 프로세서 생성을 겨냥한 게임 엔진에 선택하면 좋다. 유일한 단점이라면 최악의 경우 쓰레드 타이밍의 영향을 최소화할 수 있도록 쓰레드 실행시간을 조절해야 한다는 점이다. 이 타이밍 변동이 실제로 게임 플레이에 어떻게 영향을 미치는지에 대해서는 연구가 더 필요하다.

데이터 병렬 모델은 향후에 생각해봐야 할 것임이 분명하다. 프로세서코어의 양이 함수병렬 모델이 사용할 수 있는 작업의 수를 넘어 증가할 때 요구된다. 이런 종류의 데이터 병렬성 지원을 위해 맞춤 구성요소를 코딩하는 문제에 비교할 때 현재 사용하기 위해 확장성을 증가시킨다고 충분한 이점이 제공되지 않는다.

최근 경향은 병렬화의 내부 형식을 사용하는 엔진 요소를 생성하는 쪽으로 가는 것 같다. 이렇게 하면 엔진 개발자는 쓰레딩 문제를 걱정할 필요가 없지만 프로그램의 큰 부분이 순차적으로 일어날 수 있고 그 결과 성능은 엉망이 된다. 본 기사에서 제시된 견해는 지금까지 게임의 일부만이 아니고 전반적 게임 루프의 병렬화가 가능하다는 것이었다. 여기서 제시된 모델들은 특수 게임 엔진 아키텍처를 개발하는데 있어 훌륭한 출발점이 될 수 있다.

## 관련 글 읽기

[1] Gabb H., Lake A. 2005. Threading 3D Game Engine Basics. http://www.gamasutra.com/features/20051117/gabb_01.shtml

[2] Costa S. 2004. Game Engineering for a Multiprocessor architecture. MSc Dissertation Computer Games Technology. Liverpool John Moores University.

Wilson K. 2006. Managing Concurrency: Latent Futures, Parallel Lives. http://www.gamearchitect.net/Articles/ManagingConcurrency1.html