다다익선: Arm Mobile Studio와 Unity 통합

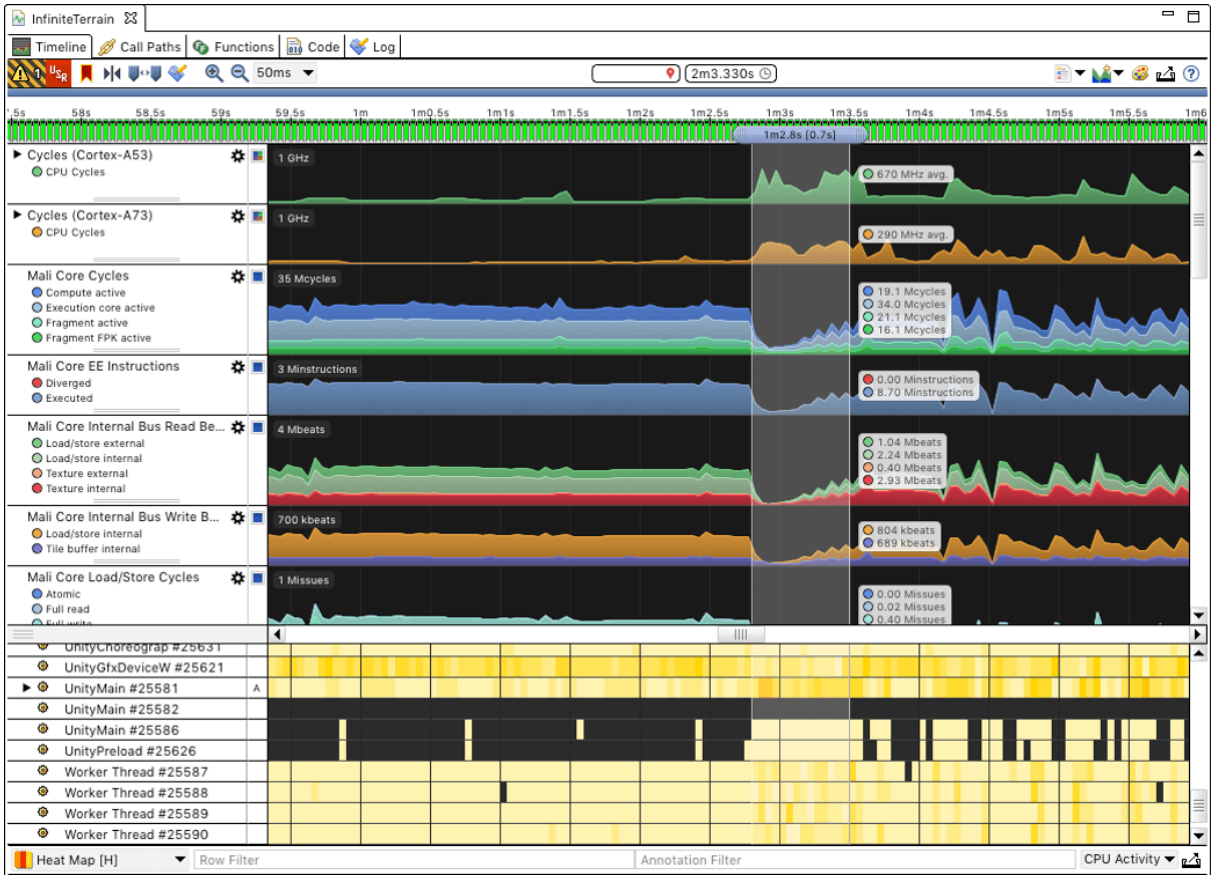
모바일 게임 개발자는 최신 고성능 프리미엄 스마트폰부터 보급형이나 구형 장치에 이르는, 다양한 장치에서 작동하는 콘텐츠를 만들고자 분투합니다. 하지만 모바일 게임 콘텐츠가 점점 복잡해지면서 개발자는 프레임률을 안정시키고 전력소비를 줄이는데 필요한 통찰력을 제공해 줄 수있는 좋은 품질의 도구를 사용해야 합니다.

이 블로그에서는 새로운 Arm Mobile Studio 도구 모음으로 Android 성능 분석을 지원하는 방법과, Arm Mobile Studio를 게임 엔진과 결합해 훨씬 강력한 성능 분석 기능을 만드는 방법을 설명합니다. Arm Mobile Studio의 Starter Edition은 무료로 제공되며, 이 블로그에서 사용한 샘플 코드의 소스 코드는 github에서 확인하실 수 있습니다.

구체적인 설명을 위해, Arm Mobile Studio에서 가장 용도가 다양하며 상세한 성능 분석 도구인 스트림라인(Streamline)을 집중적으로 살펴보겠습니다. 스트림라인을 Unity와 통합하는 방법을 설명하여, 게임의 다양한 요소가 모바일 장치의 Arm Cortex-A CPU 또는 Arm Mali GPU 리소스를 어떻게 사용하는지 보여드리겠습니다.

스트림라인 소개

스트림라인은 Android 장치의 여러 소스에서 샘플 기반 및 이벤트 기반 성능 데이터를 수집하고, 종합한 결과를 다양한 방식으로 표시합니다. 이 블로그에서는 타임라임(Timeline) 뷰를 집중적으로 살펴봅니다. 화면 상위 반에는 수집한 시스템 성능 카운터가 표시되고, 하위 반에는 같은 타임라인에 관한 다양한 유형의 정보를 표시할 수 있습니다. 여기서는 히트 맵(Heat Map)이 표시되는데, 프로파일링한 응용 프로그램 스레드에서의 연산 활동 분포를 확인할 수 있습니다.

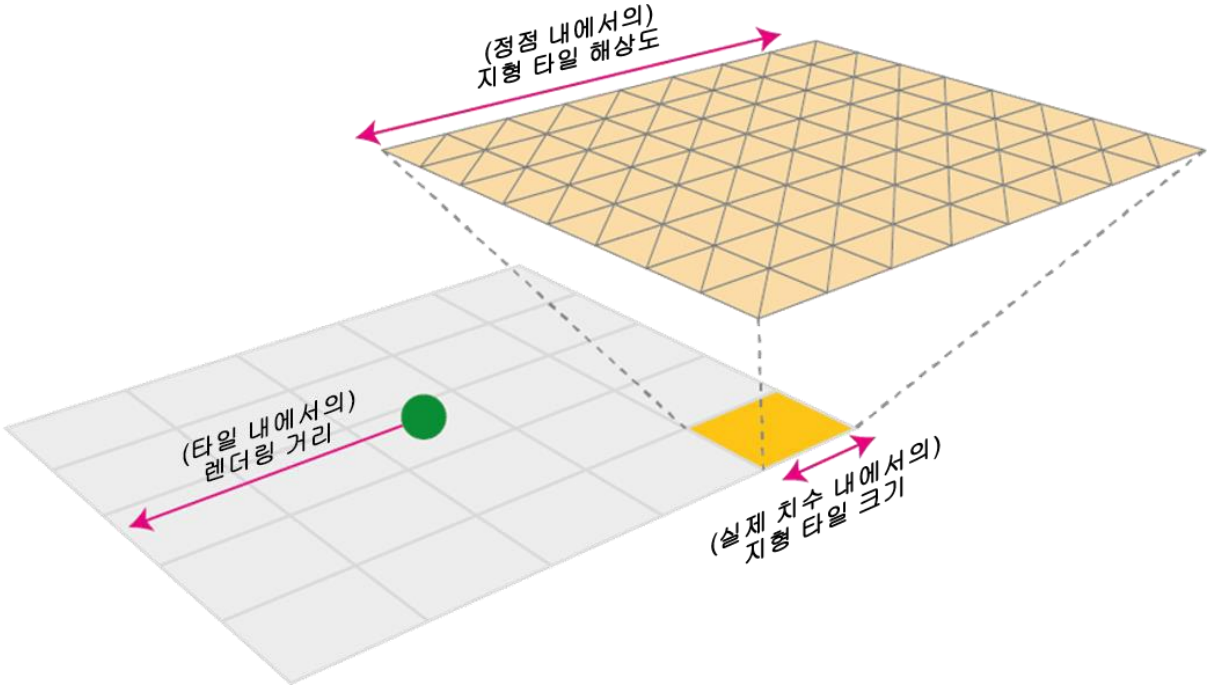


분석 대상은 무엇입니까?

연산으로 생성된 지형을 플라이스루방식으로 진행되는 아주 간단한 Unity 콘텐츠를 분석합니다. 카메라가 마구 움직이며, 새로운 지형 타일이 필요하면 즉석에서 바로 생성됩니다. 카메라에서 너무 멀리 있는 타일은 삭제되며, 따라서 장면이 일단 작성되면 시간이 지나도 복잡성이 크게 달라지지 않습니다. 간혹 카메라가 느리게 움직이면 새로운 지형 생성 속도가 급감했다가 카메라가 빨라지면 지형 생성 속도가 다시 상승합니다. 타일의 크기를 확인할 수 있도록 각 타일의 가장자리는 어두운 색으로 렌더링했습니다.

지형 타일 생성은 대량의 연산이 필요하기 때문에 메인 Unity 스레드를 방해하지 않고 백그라운드 스레드를 보낼 수 있는 Unity Job Scheduler를 이용했습니다. 이 기능을 이용하면 사용자는 새 지형이 생성될 때마다 멈추지 않는, 안정적인 프레임률의 영상을 볼 수 있습니다.

데모는 4가지 장면을 실행하도록 구성했는데, 각 장면은 똑같아 보이지만 서로 다른 지형 타일을 생성합니다. 아래 그림에서 알 수 있듯이 지형은 크기가 고정된 다양한 지형 블록으로 구성됩니다. 각 블록은 해상도가 고정된 여러 메시(녹색 지형용 하나, 노란색 지형용 하나, 물용 하나)로 구성됩니다. 렌더링 거리 (Render Distance)는 플레이어 주위에서 생성될 타일 수를 제어합니다.



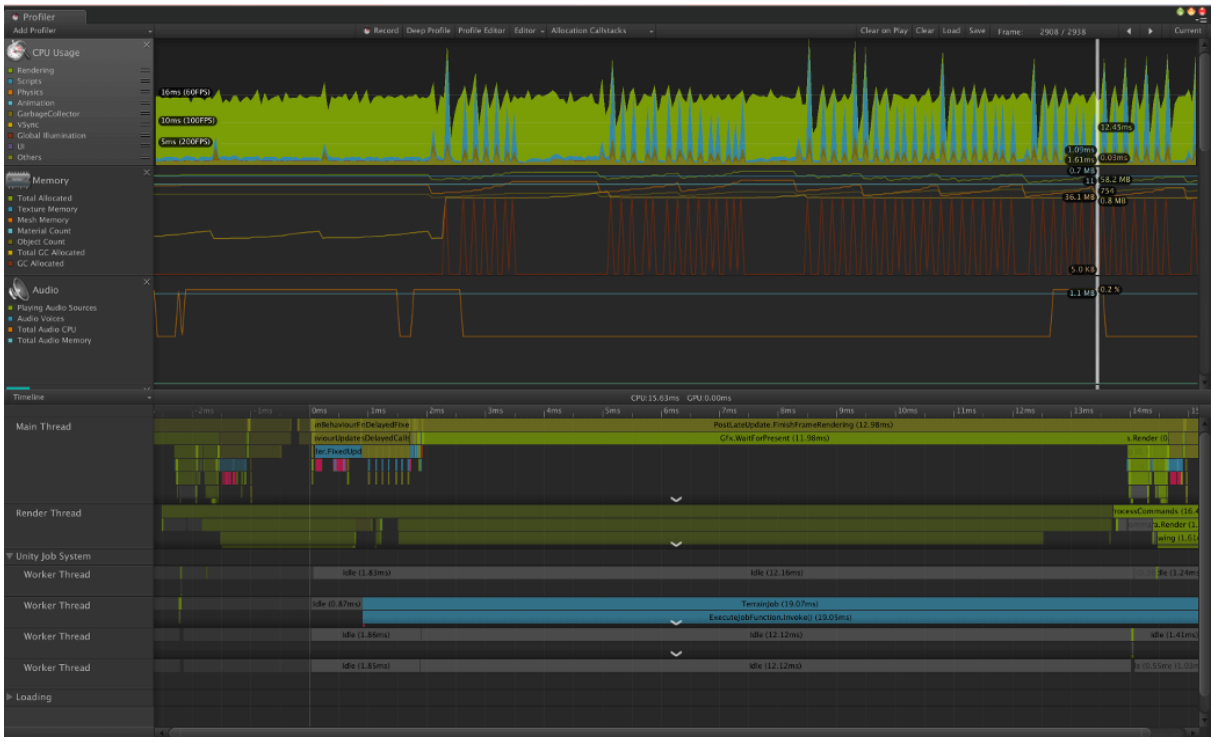
장면 4개는 다음과 같이 구성됩니다.

장면	렌더링 거리	지형 타일 크기	지형 해상도	동시 지형 생성 수
1	3	20x20	32x32	8
2	3	20x20	32x32	1
3	6	10x10	16x16	8
4	6	10x10	16x16	1

2017년 출시된 Huawei P10 휴대폰에서 프로파일링 작업을 수행하겠습니다.
이 장치에는 HiSilicon Kirin 960 칩이 있으며, 이 칩은 고성능 Arm Cortex-A73 CPU 코어 4개, 고효율 Arm Cortex-A53 CPU 코어 4개, Arm Mali-G71 MP8 GPU로 구성됩니다.

Unity에서의 프로파일링

Unity에는 자체 프로파일러가 있으며, 이 프로파일러는 Android 장치에서 아주 잘 작동합니다.



Unity의 프로파일러는 작업을 언제 예약했는지는 아주 잘 보여주지만, 플랫폼의 실제 리소스(이 블로그에서는 CPU와 GPU)와 사용 현황을 자세히 보여주지는 않습니다. 60FPS를 달성한 것 같지만, 이를 위해 모든 CPU 코어를 한계까지 사용하며 배터리 소모가 극심할지도 모릅니다. 바로 이 지점에서 스트림라인이 필요합니다.
이 블로그의 나머지 부분에서는 스트림라인이 어떤 데이터를 캡처하고 제공하는지, 그리고 스트림라인의 *주석(annotation)* 기능을 이용해 Unity 게임의 상위 수준 상황 정보를 스트림라인으로 전달하여 데이터를 쉽게 해석하는 방법을 설명하겠습니다.

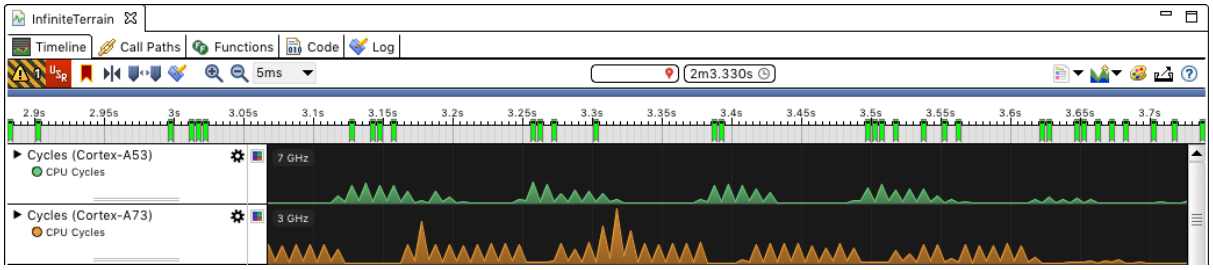
스트림라인을 이용한 Unity 프로파일링 – 기능

주석을 Unity 게임에 삽입하는 방법을 확인하기 전에, 스트림라임의 3가지 주석 기능을 사용하도록 게임을 수정하면 예제 콘텐츠가 어떤 결과를 보여주는지 살펴보겠습니다.

- **마커(Markers)**는 가장 단순한 형태의 주석으로, 스트림라인의 타임라인 뷰 상단에 라벨이 표시되는 단일 시점입니다.
- **채널(Channels)**은 각 스레드 옆에 별도의 정보 행을 표시하는 더욱 체계적인 주석입니다. 주석은 채널 안에 배치할 수 있으며, 마커와 달리 각 주석은 일정한 시간 범위에 적용됩니다.
- **사용자 지정 활동 맵(Custom Activity Maps)**은 가장 발전된 형태의 주석으로, 복잡한 종속항목이 있기도 하는 전역적(교차 스레드) 활동을 표시하는 장치입니다. 각각의 사용자 지정 활동 맵은 스트림라인 UI 하위 반에 자체 뷰로 표시됩니다.

(나중에 자세히 이야기할) 프로파일 수집이 끝나면 스트림라인에서 프로파일이 열려 어떤 일이 진행 중인지 확인할 수 있습니다.

콘텐츠를 살펴볼 때 우리의 주의를 가장 먼저 끄는 것은 (타임라인 상단에 녹색으로 표시되는) 마커로, 여기서는 각 프레임 시작 지점을 표시합니다.



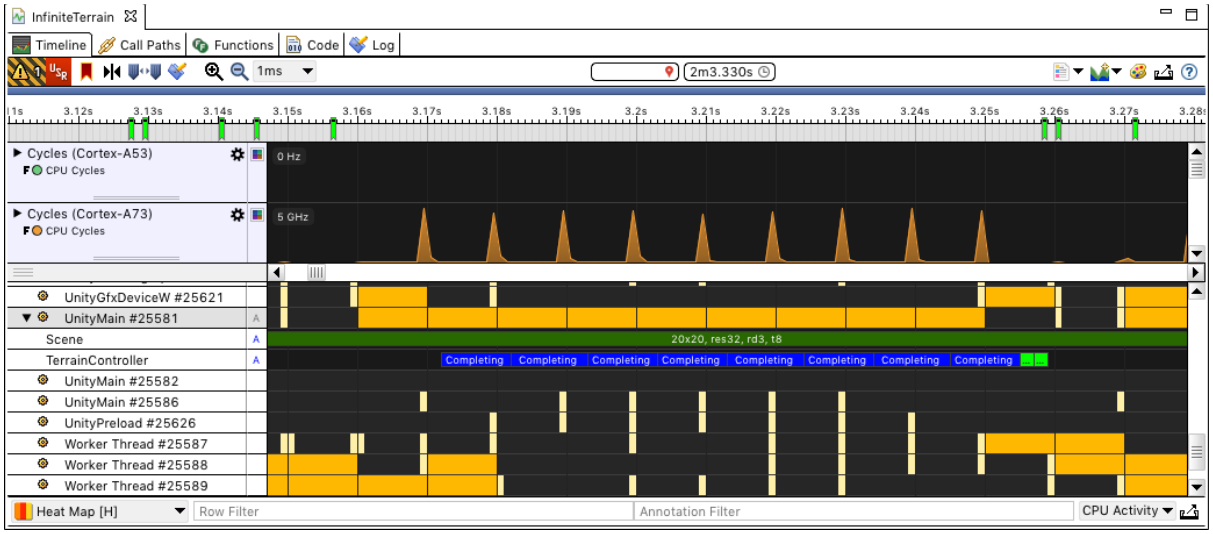
프레임률이 원하는 만큼 규칙적이지 않으며, 모든 CPU 코어에서 상당히 급작스러운 활동이 나타남을 알 수 있습니다. 프레임률이 느려졌다가 회복되며, 가끔 정지하기도 하죠. 지형 생성에서 예상한 것과 거의 같은 결과입니다. 조금 더 자세하게 살펴볼까요?

스트림라인의 타임라인 뷰는 2가지 영역으로 나뉩니다. 상단 뷰는 계량적 그래프를 표시하며 하위 반 뷰는 다양한 요소를 표시할 수 있습니다. 그중 하나는 히트 맵으로, 작업의 시스템 내 분산을 확인하고 상단 타임라인에 필터를 적용해 특정 프로세스나 스레드에 적용되는 작업만 표시할 수 있습니다. 히트 맵을 살펴보면서 먼저 *UnityMain* 스레드를 선택한 다음 *Worker Thread* 스레드를 모두 선택하면, 메인 Unity 스레드와 작업 스케줄러 내 스레드에서의 CPU 활동 분포를 확인할 수 있습니다.

UnityMain 스레드의 CPU 프로파일로, Cortex-A73 CPU에서 활동이 급증함을 확인할 수 있습니다.

모든 *Worker Thread* 스레드의 CPU 프로파일로, Cortex-A73 및 Cortex-A53 CPU 모두에서 활동이 소규모로 증가함을 알 수 있습니다.

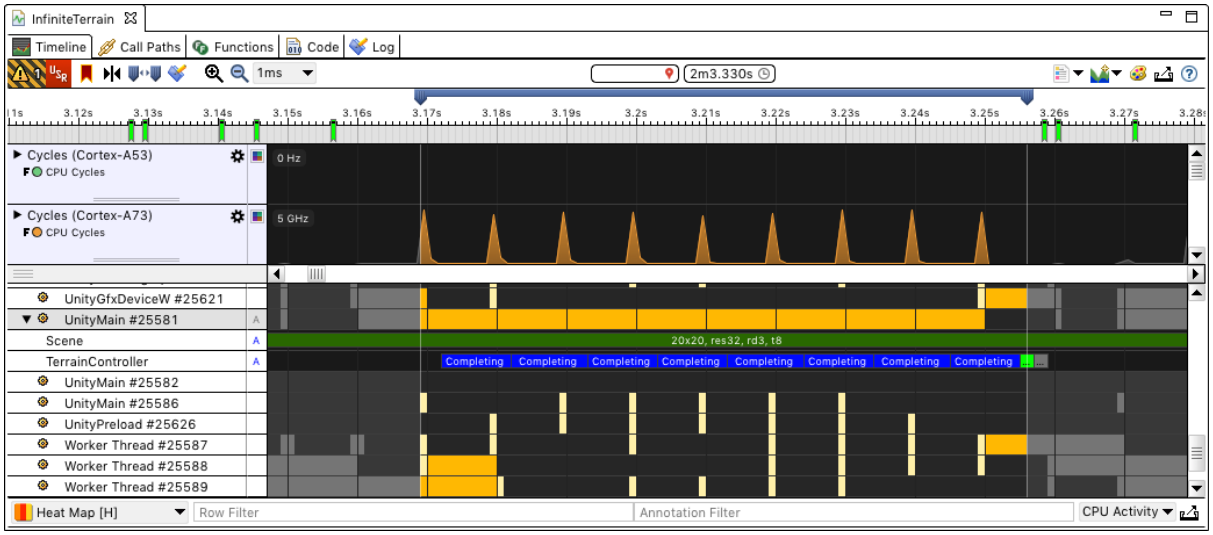
메인 스레드를 살펴보겠습니다. 왼쪽 스크린샷을 자세히 살펴보면, 검사 중인 *UnityMain* 스레드 옆에 'A' 마커가 있습니다. 스트림라인 주석 채널이 존재한다는 뜻이죠. 타임라인을 조금 확대하고, *UnityMain* 스레드를 확장해 무슨 일이 진행 중인지 확인해보겠습니다.



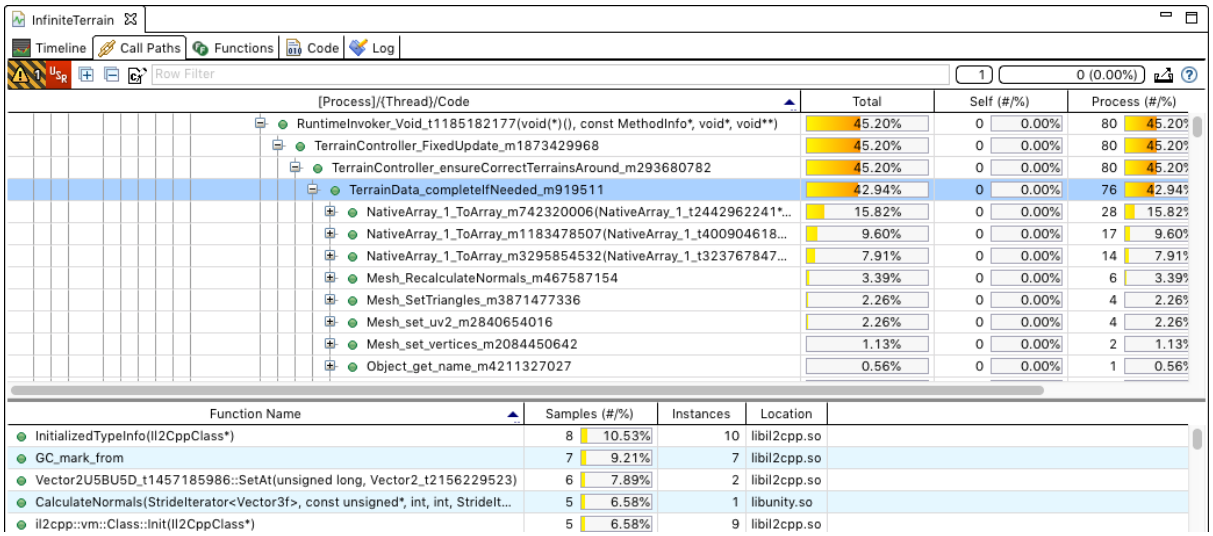
*Scene*과 *TerrainController* 행은 스트림라인 *채널로*, 게임에 삽입된 주석이 생성했습니다. *Scene* 채널은 현재 실행 중인 장면을 표시합니다. 20x20, 32x32 버전으로 렌더링 거리는 3이고 동시 실행 가능 스레드가 8개임을 알 수 있습니다.

TerrainController 채널은 메인 Unity 스레드에서 실행 중인 코드의 특히 흥미로운 부분을 표시하는 데 사용합니다. 파란색 블록은 지형(Terrain) 작업 완료 시 실행하는 코드를 표시합니다. 녹색 블록은 새 지형 생성이 예약된 지점을 표시합니다. 여기서는 모든 메인 스레드 활동은 본질적으로 작업을 완료하고 최종 메쉬를 생성해 장면에 삽입해야 할 때 수행해야 하는 작업에 기초함을 알 수 있습니다.

특정 스레드는 물론, 특정 기간을 집중적으로 분석할 수도 있습니다. 스트림라인의 *캘리퍼스 (calipers)*를 이용하면 분석할 특정 기간을 표시할 수 있습니다. 여기서는 지형 완료와 관련된 활동 집중 기간의 시작과 끝을 선택했습니다(캘리퍼스는 타임라인 뷰 상단에서 설정됩니다).



이제 호출 경로(Call Paths) 뷰로 이동하면, 캘리퍼스가 선택한 기간에 시간을 어디에 쓰는지를 명확하게 확인할 수 있습니다. [Unity용 IL2CPP 스크립팅 백엔드](#)를 사용했기 때문에, 기본 모노(Mono) 런타임을 사용할 때보다 훨씬 많은 정보가 제공됩니다. 여기서는 자세히 확인하지 않겠지만, 심층적으로 탐색해 볼 정보가 가득합니다.

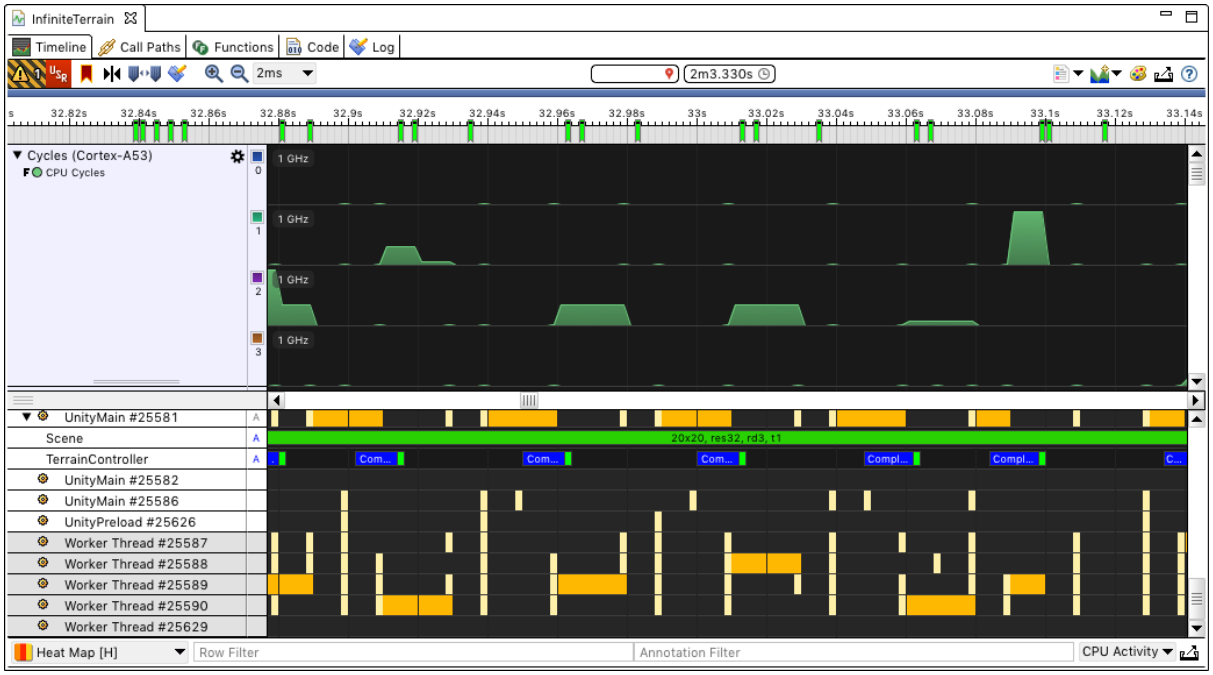


작업자 스레드는 무엇일까요?

작업자 스레드 스레드만 표시하도록 필터를 적용하면, 작업을 8개까지 동시에 실행하도록 요청했음을 감안할 때 예상했던 결과가 표시됩니다. 이 스크린샷에서는 개별 코어의 활용 상태를 알 수 있도록 Cortex-A53 클러스터를 확장했습니다.

TerrainController 채널에는 새 지형이 예약 중임을 나타내는 녹색 블록이 있고, 모든 코어에서 격렬한 활동이 진행 중이며, TerrainController의 파란색 활동은 생성된 지형들을 메인 스레드에서 한번에 처리하는 활동을 보여줍니다(UnityMain 스레드를 선택하지 않았기에 메인 스레드 활동은 그래프에 표시되지 않습니다).

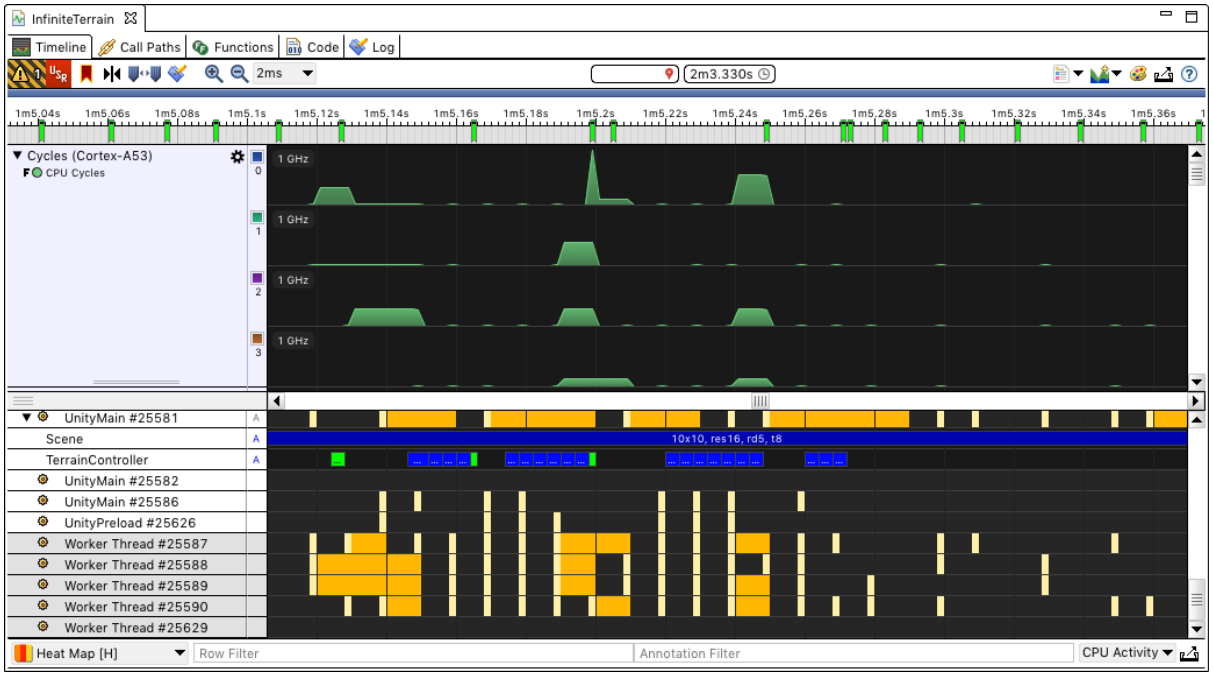
두 번째 장면의 활동과 비교해보면 재미있을 것입니다. 여기서는 지형 타일의 복잡성은 같지만, 한 번에 타일 하나만 예약할 수 있습니다.



여기서는 몇 가지 주목할 점이 있습니다.

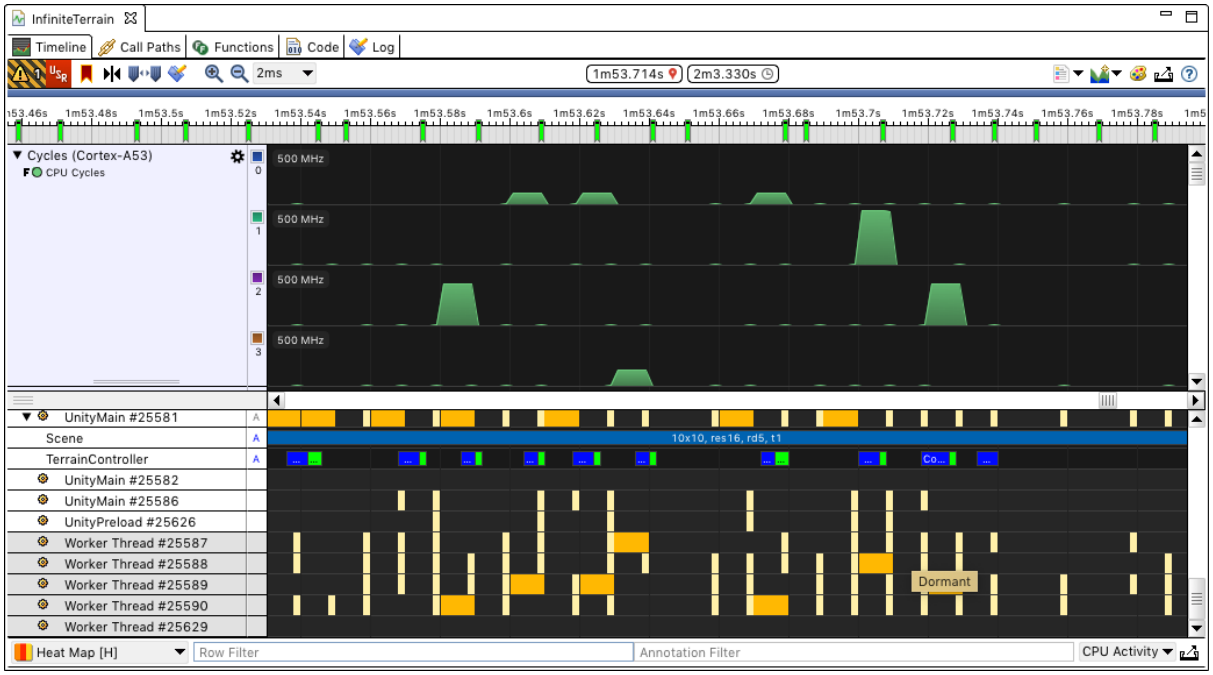
1. CPU 활동이 훨씬 낮습니다. 대부분의 코어가 거의 언제나 유휴 상태거나 유휴 상태에 가깝습니다.
2. 프레임률은 완벽하지는 않지만 훨씬 더 부드럽습니다. 한 번에 프레임 하나만 완료하기 때문에, 우리를 방해하던 메인 스레드에서 발생한 많은 활동이 줄어듭니다.

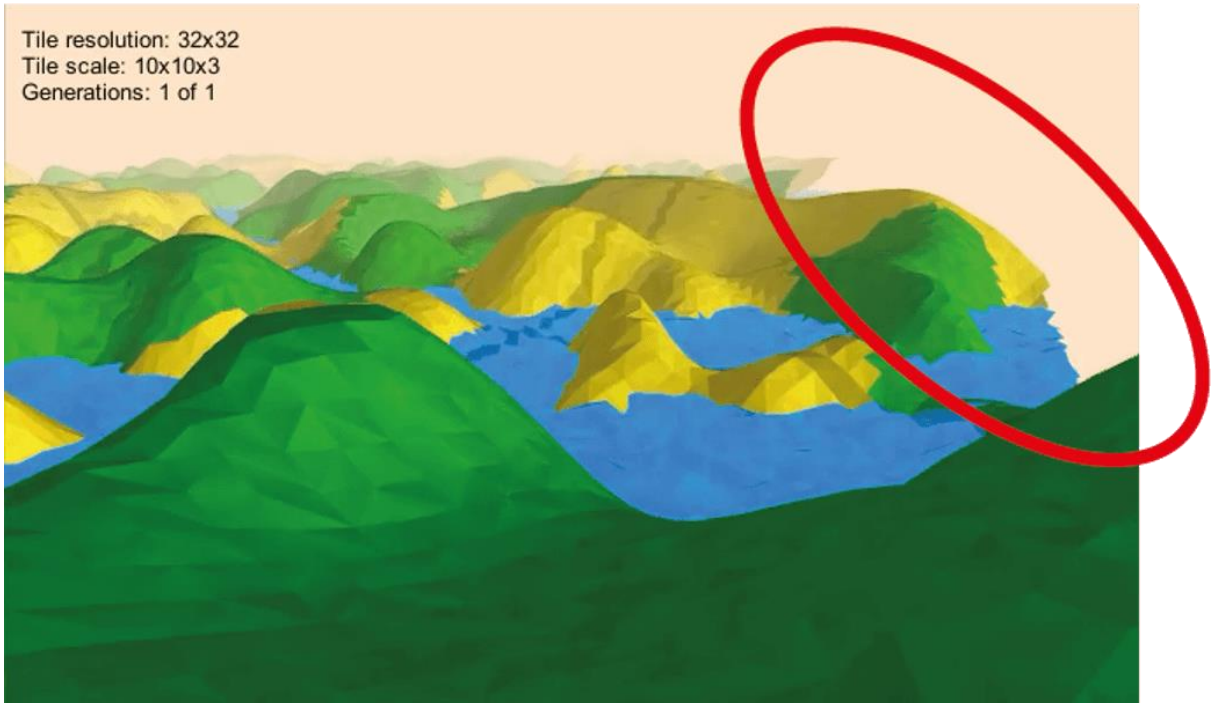
프로필을 더 작은 타일을 사용하는 3번째 장면과 비교해 보겠습니다.



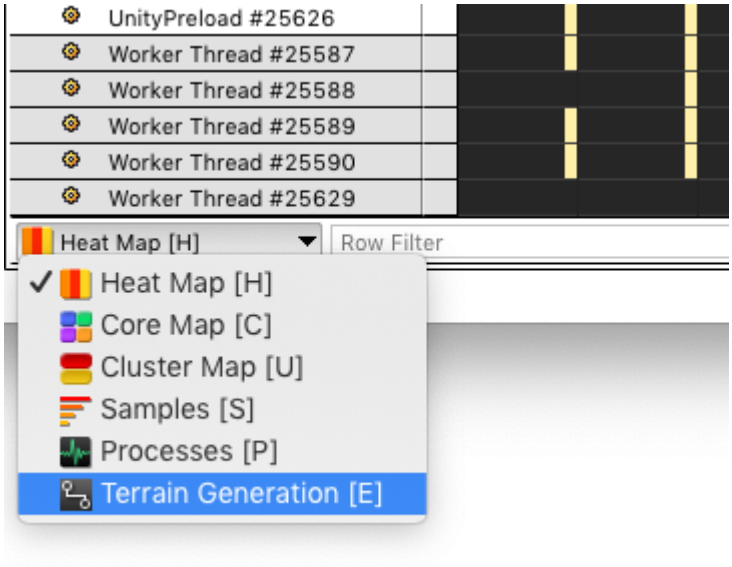
보시다시피, CPU 활동이 훨씬 낮으며 메인 스레드의 파란색 완료 작업 블록이 상당히 짧아 첫 번째 장면과 비교했을 때 프레임률이 훨씬 더 부드럽습니다(하지만 당연히 전체 작업 수가 많으며, 따라서 지형 생성 속도가 지형 위를 이동하는 카메라 속도를 따라잡을 수 있게 해야 했습니다).

타일이 작고 한 번에 지형 생성을 하나만 실행하는 4번째 장면에서는 전반적으로 가장 우수한 프레임률이 보이지만, 지형 생성이 카메라를 따라잡을 수 있는 속도로 진행되도록 신경 써야 했습니다. 동영상 원본에서도 카메라가 4번째 장면을 빠르게 지나갈 때 간혹 카메라 속도를 따라잡지 못함을 알 수 있습니다.

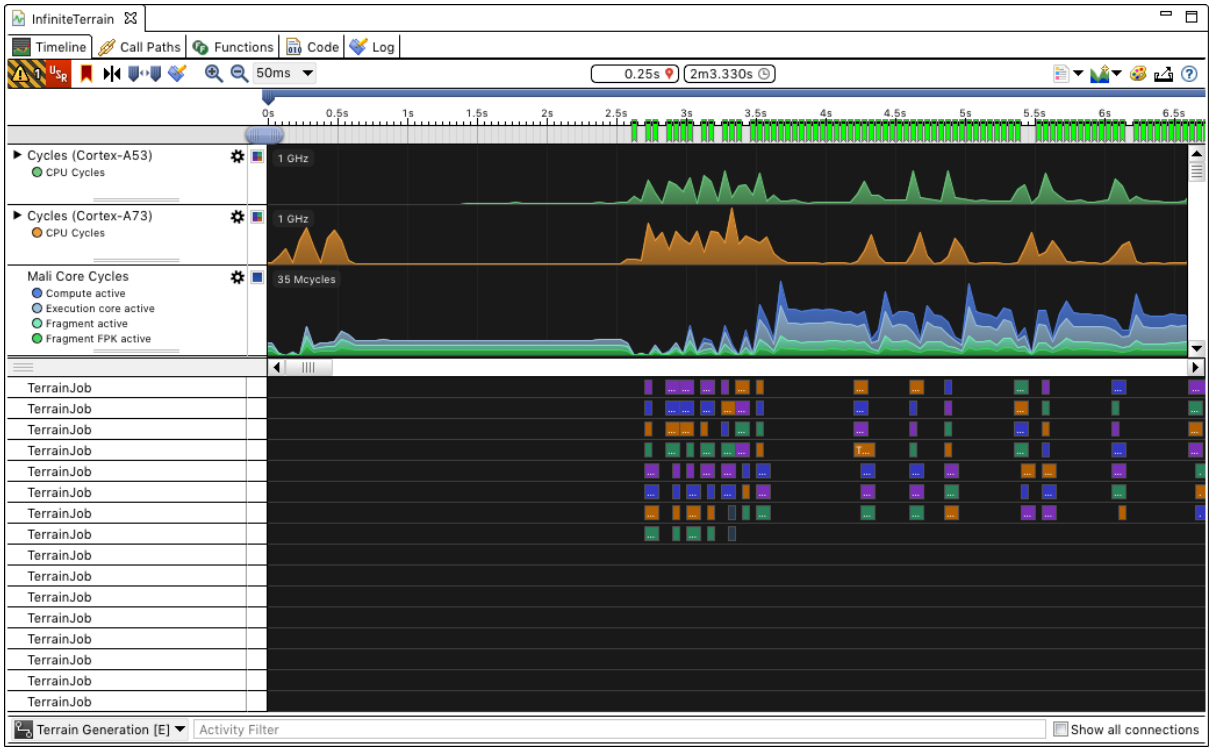




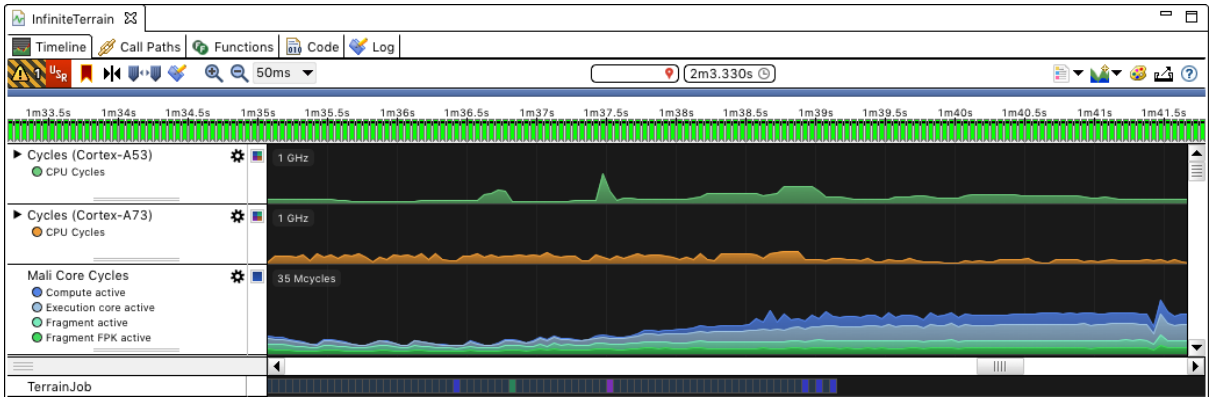
마지막으로, *사용자 지정 활동 맵*을 이용하면 작업자 스레드의 타일 생성 수행에 관한 훨씬 많은 정보를 얻을 수 있습니다. 각각의 사용자 지정 활동 맵은 지금까지 히트 맵을 표시하는 데 사용한, 화면 왼쪽 아래의 메뉴에서 옵션으로 표시됩니다.



지형 생성(Terrain Generation) 뷰를 선택하면, 각 지형 생성 작업의 시작 및 중단 시점을 보여주는 색상 지정 상자가 표시됩니다. 마우스 커서를 위에 올리면 해당 지형 타일의 세계 좌표와 시작 시점 및 완료까지 걸리는 시간을 알 수 있습니다. 이 스크린샷 역시 Mali GPU에서 일어난 연산 작업을 그래프로 만든 것입니다. 그리고 예상했듯이, 지형을 작성할 때 GPU 활동이 꾸준히 증가합니다. 이 스크린샷은 첫 번째 장면 시작 부분에 집중할 때 찍었는데, 이때는 대형 타일을 동시에 8개까지 생성하고 있었습니다. 메인 스레드가 새 지형 일체를 준비할 때 일시 정지하기 때문에, GPU는 오랫동안 유휴상태가 됩니다.



더 작은 타일을 연속으로 생성하는 4번째 장면으로 넘어가면, GPU 활동이 훨씬 안정적이며 지형 작업이 한 번에 하나만 실행되었음을(그리고 타일 크기가 작아 각 작업 시간이 짧음을) 확인할 수 있습니다.



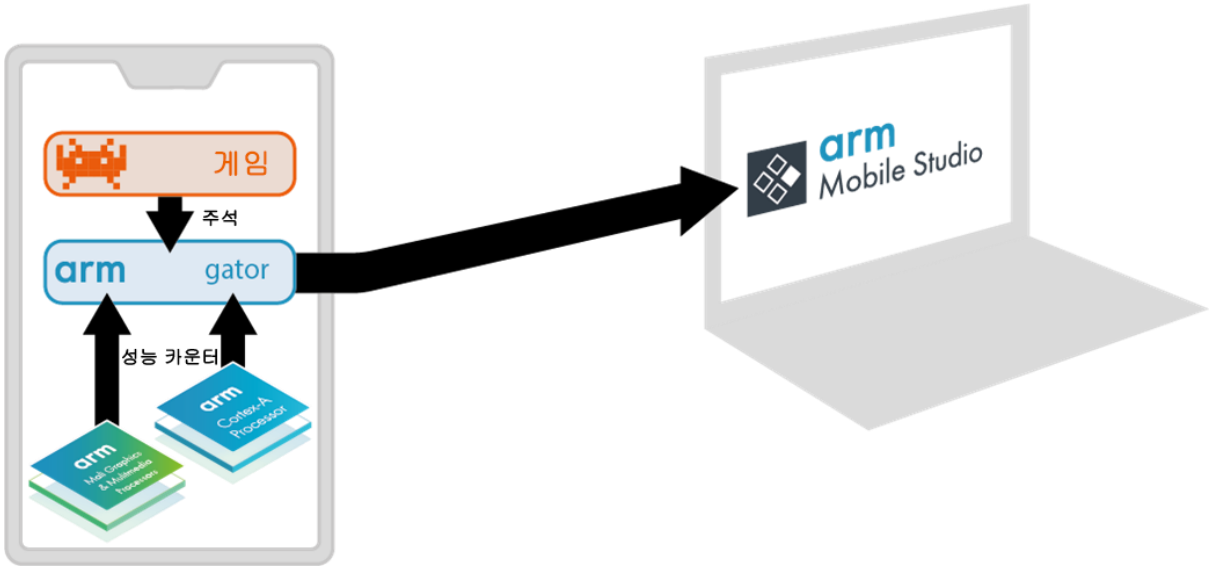
이제까지 스트림라인에서 얻을 수 있는 추가 정보의 극히 일부를 간단히 살펴보았습니다. 게임의 주석을 이용하면 훨씬 수준 높은 상황 정보를 확인할 수 있습니다. 여기서는 다음을 사용했습니다.

- 새 프레임 시작 시점을 알려주는 *마커*
- 실행 중인 장면과 메인 스레드가 실행 중인 작업을 알려주는 *채널*.
- 비동기적으로 예약된 지형 생성 작업을 알려주는 *사용자 지정 활동 맵*.

모두 대단히 멋진 기능이죠. 그렇다면 이러한 기능은 어떻게 작동할까요?

스트림라인 주석 소개

스트림라인 작동 원리를 좀 더 자세히 살펴보겠습니다. Android 응용 프로그램을 분석할 때는 (응용 프로그램에서와 같은 사용자로 실행하는) *게이터 (gator)*라는 별도의 프로세스가 장치에서 실행되어, 여러 하드웨어 소스(Mali GPU와 Arm Cortex-A CPU 등)에서 얻은 프로파일링 정보를 수집하고 종합한 메트릭 스트림을 컴퓨터로 전송합니다. 스트림라인 주석은 응용 프로그램이 자체 마커와 메트릭을 스트림에 직접 삽입하는 기능입니다.



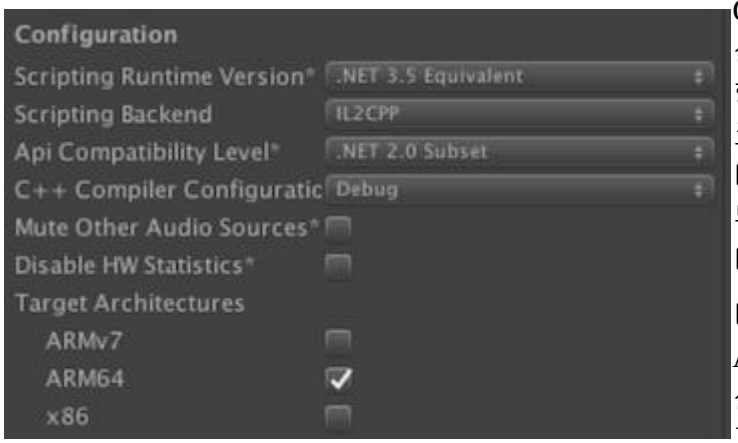
Unity에서 스트림라인 주석을 사용하는 방법

스트림라인 주석은 특정 프로토콜을 사용하며, Arm Mobile Studio의 한 부분으로써 C로 구현된 오픈소스가 제공됩니다. Unity 콘텐츠에서 스트림라인 주석을 쉽게

생성하려면, C 구현에서 몇 가지 C# 래퍼를 사용해야 합니다. 이 예제에서 사용하는 래퍼는 필수 C 구현과 함께 Unity 자산 패키지(Unity Asset Package)로 이용할 수 있습니다. 다운로드해 여러분의 프로젝트에 사용자 지정 자산 패키지로 가져오십시오. 이 패키지는 `Arm` 네임스페이스에 새 메서드를 추가하며, 이를 통해 프로젝트에서 스트림라인 주석을 쉽게 사용할 수 있습니다. API 설명서는 패키지에 포함된 README.md 파일에 있습니다.

최상의 경험을 위한 Unity 프로젝트 설정

Unity에서 Android 빌드를 가장 빠르고 쉽게 분석하려면, 몇 가지 지정된 Android Player 설정을 구성해야 합니다.



*IL2CPP*스크립팅 백엔드(Scripting Backend)로 사용 중인지 확인하고 C++ 컴파일러 구성(Compiler Configuration)을 *디버그(Debug)*로 설정하십시오. 이렇게 하면 성능 향상을 위해 스크립트를 네이티브 코드로 컴파일하며, 스트림라인은 디버그 정보를 확인해 호출 경로 뷰의 함수에 성능 데이터를 매핑할 수 있습니다.

타겟 아키텍쳐(Target Architecture)를 ARM64로 설정합니다(ARMv7이 기본값입니다). 대부분의 현대 모바일 장치는 64비트기 때문에 고품질의 코드 생성을 이용할 수 있습니다.

마커 추가

마커는 주석을 가장 쉽게 사용하는 방법입니다. 제공되는 메서드는 문자열 하나와 선택형 색상을 이용합니다. 예를 들어 녹색 프레임별 마커 방출을 위해, GameObject 중 하나에서 다음 코드를 사용했습니다(Unity 아키텍쳐에 익숙하지 않는 사용자들을 위해서, Update() 매소드는 매 프레임마다 호출이 됩니다).

전체 화면

채널 추가

채널 사용은 그렇게 어렵지 않습니다. 먼저 채널을 만들고 이름을 지정해야 합니다. 그러면 채널(Channel) 객체에서 메서드를 사용해 채널에 주석을 기록할 수 있습니다. 예를 들면 다음과 같습니다.

전체 화면

채널의 주석은 일정 기간에 적용된다는 사실을 명심하십시오. 다음 주석을 시작하기 전에 현재 주석을 끝내고 싶다면 `end()` 메서드를 사용합니다. 예를 들어 메인 스레드에서 지형 완성을 수행하는 TerrainController 부분은 다음과 같이 래핑됩니다.

전체 화면

사용자 지정 활동 맵 사용

사용자 지정 활동 맵(Custom Activity Maps, CAMs)은 채널 상단의 레이어라고 생각하면 됩니다. 트랙을 만들기 전에 CAM의 이름을 지정해야 합니다. 그러면 사용자들은 채널에 주석을 추가할 때처럼 주석을 그 트랙에 추가할 수 있습니다.

이 예제에서는 지형 생성 CAM(Terrain Generation CAM)을 다음과 같이 생성했습니다.

전체 화면

하지만 우리의 사용 사례에서는 문제가 하나 있습니다. Unity 작업 시스템(Unity Job System)에서 작업을 실행하면, 게임의 다른 객체 모델과 거의 상호작용하지 못하는 것입니다(스레드 안전성을 유지하는 데는 도움이 됩니다). 작업에서 우리가 할 수 있는 일은 시작 및 종료 시간을 기억하고, 메인 스레드가 작업을 정리하는 시점을 기억하는 것뿐입니다. 이 시점에서 작업의 활동을 CAM에 등록할 수 있습니다.

C# 래퍼는 작업에서 안전하게 호출할 수 있으며, 현재 시각을 스트림라인 주석에 필요한 형식으로 반환하는 함수를 제공합니다.

전체 화면

메인 스레드로 돌아오면, 사용할 트랙을 선택하고(쉽게 확인할 수 있도록, 우리는 트랙을 풀에서 관리해 중복되지 않게 합니다) 작업을 트랙에 등록할 수 있습니다. 여기서 `job.timings`는 작업이 작성하는, 작업의 시작 시간과 종료 시간을 포함하는 2개 항목 배열입니다.

전체 화면

이제 다 끝났습니다! 이 Unity 패키지(Unity Package)를 개량해 다른 기능을 추가할 예정입니다. 의견이 있다면 언제든 알려주십시오!

스트림라인에서 기본 프로파일 수집

스트림라인에서 첫 번째 프로파일을 수집하려면 몇 가지 단계를 거쳐야 하지만, 설정이 끝나면 나머지 작업은 아주 쉽습니다.

먼저 Arm Mobile Studio Starter Edition의 무료 Windows, Mac 또는 Linux 버전을 다운로드해 설치합니다.

앞에서 스트림라인의 아키텍쳐를 설명할 때 언급한 중요 사항을 떠올려보십시오.

- 모바일 장치에서 게이터를 실행하고, 응용 프로그램에 대한 액세스를 제공해야 합니다.
- 데이터를 장치에서 꺼내 도구로 옮길 수 있어야 합니다.

스트림라인에서는 몇 가지 방법으로 이 작업을 할 수 있지만, 다양한 장치에 안정적으로 액세스할 수 있는 가장 쉬운 방법은 먼저 다음과 같은 중요 정보를 파악하는 것입니다.

- 응용 프로그램이 32비트인지 64비트인지 확인합니다. 위의 지침을 따랐고 Unity 게임을 ARM64 옵션으로 제작했다면 응용 프로그램은 64비트가 됩니다.
- Unity의 Android Player 설정에서 지정한 응용 프로그램의 패키지 이름(Package Name)을 알아야 합니다. 예제 응용 프로그램에서 패키지 이름은 `com.Arm.InfiniteTerrain`입니다.
- 게이터 바이너리를 얻는 방법: Arm Mobile Studio를 설치한 `streamline/bin/arm`(32비트) 또는 `streamline/bin/arm64`(64비트) 폴더에서 확인할 수 있습니다.

이러한 정보를 파악했다면, 다음 방법에 따라 분석을 수행하면 됩니다.

- 응용 프로그램을 장치에 설치했는지 확인합니다.
- Android `adb` 도구를 이용해 모바일 장치의 네트워크 포트를 Arm Mobile Studio를 실행하는 시스템의 로컬 포트로 포워딩합니다.
- `adb`를 이용해 (응용 프로그램에 따라 32비트 또는 64비트 버전의) 게이터를 장치에 푸시하고, 응용 프로그램에서와 같은 패키지 이름으로 게이터를 실행합니다.
- 스트림라인을 시작하고, adb로 트래픽을 포워딩할 때 사용한 로컬 포트에 연결합니다. 이 시점에서 수집에 사용할 성능 카운터를 선택할 수 있습니다.
- 모바일 장치에서 응용 프로그램을 실행합니다. 스트림라인이 들어오는 분석 데이터를 수집하기 시작할 것입니다.

게이터를 한번 실행했다면, 사용자는 게이터를 다시 시작할 필요없이 새로운 버전의 응용프로그램을 설치, 스트림라인 시작/종료, 더 많은 분석을 수행 할 수 있습니다.

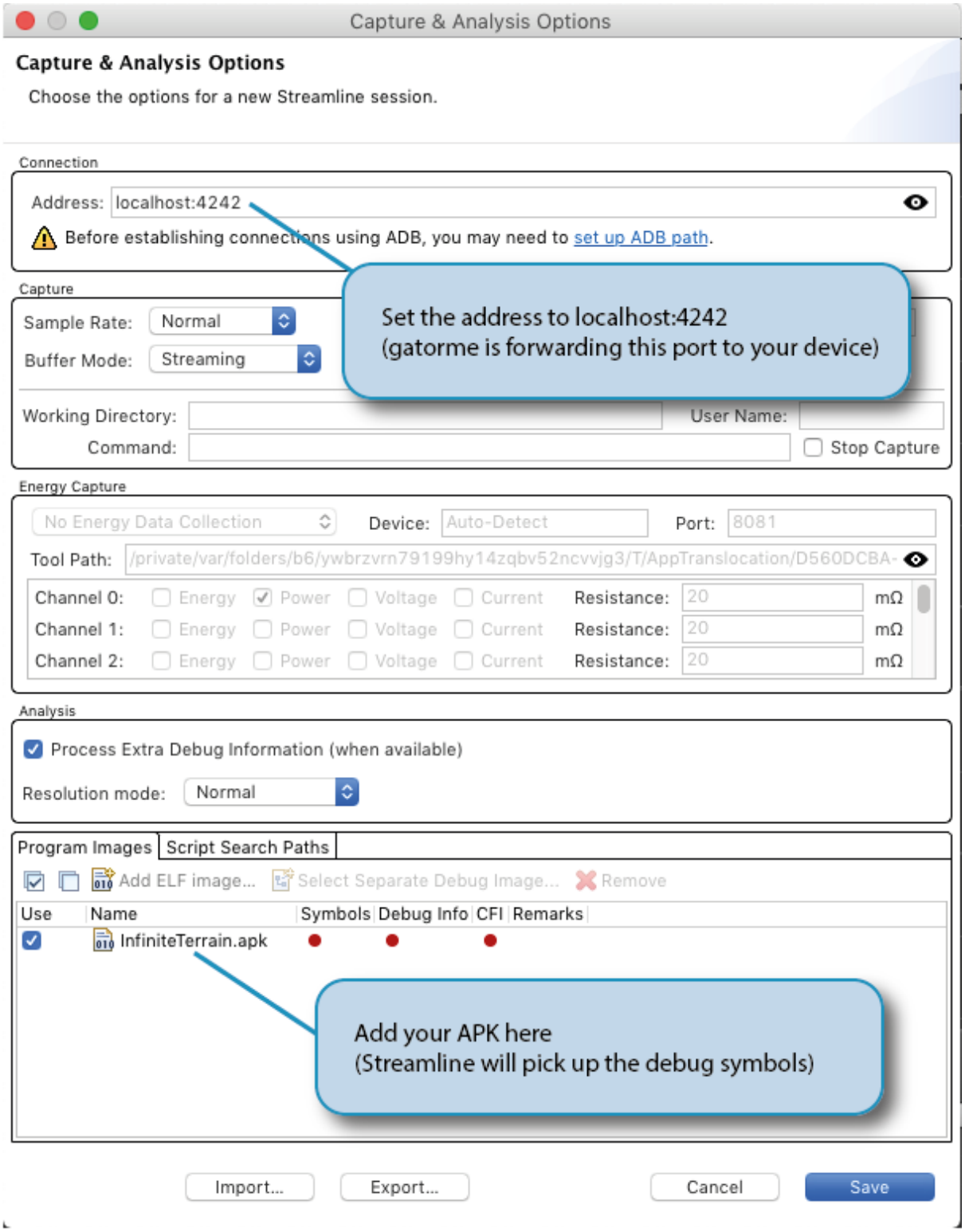
gatorme 스크립트를 사용하면 `게이터`와 `adb`에 대한 환경설정과 실행을 더 쉽게 할 수 있습니다. 사용자는 실행할 게이터 바이너리의 경로, 응용 프로그램의 패키지 이름과 장치에 있는 Mali GPU를 입력하기만 하면 됩니다(Mali GPU 입력은 게이터가 장치를 검사했지만 어떤 GPU를 사용하는지 확인하지 못할 때 유용합니다). 그리고 이 메서드가 다양한 모바일 장치에서 정상적으로 작동하는지, 그리고 프로파일링 작업이 끝나면 게이터가 올바르게 종료되는지 확인하는 단계도 진행됩니다(조만간 `gatorme` 기능을 스트림라인에 바로 적용할 예정입니다!).

gatorme 설명서에서 자세히 설명하지만, 이 예제에서는 APK를 장치에 설치한 후 InfiniteTerrain 콘텐츠를 프로파일링하는 방법을 설명합니다.

먼저 명령 행에서 gatorme를 실행합니다.

전체 화면

이제 스트림라인을 실행해 캡처를 시작할 수 있습니다. 올바른 작업 진행을 위해 몇 가지 항목을 설정해야 합니다.



설정이 끝나면 구축/분석/수정 단계는 쉽게 반복할 수 있습니다. 응용 프로그램을 종료하는 동안 gatorme는 계속 실행하면서, Unity에서 바로 빌드 및 실행을 수행하고 추가 스트림라인 정보를 캡처할 수 있습니다.

이 블로그가 흥미롭고 유익하길 바랍니다. InfiniteTerrain 예제의 모든 소스 코드는 GitHub(Apache 2.0 라이선스)에서 다운로드할 수 있습니다. 소스 코드와 그래픽 자료 일체는 물론 ArmMobileStudio.unitypackage도 이용할 수 있는데, 이것은 Unity 사용자 지정 자산 패키지로 프로젝트에 가져오면 스트림라인 주석을 추가할 수 있습니다.

또한 일부 콘텐츠 분석을 바로 시작하고 싶다면, 사전 빌드한 64비트 Android 개발 빌드인 InfiniteTerrain.apk를 이용하면 됩니다.

마지막으로, Arm Mobile Studio나 Arm의 그래픽 및 게이밍 관련 질문이 있다면 그래픽 및 멀티미디어 포럼(Graphics and Multimedia Forum)에 참여하거나 아래 Arm Mobile Studio 개발자 사이트에서 도구 관련 설명을 읽어주십시오!

Arm Mobile Studio 리소스