Pusan National University
Computer Science and Engineering
Technical Report 2023-15

가상현실 사용자 모니터링 시스템 개발

저자1 최현호

저자2 이희근

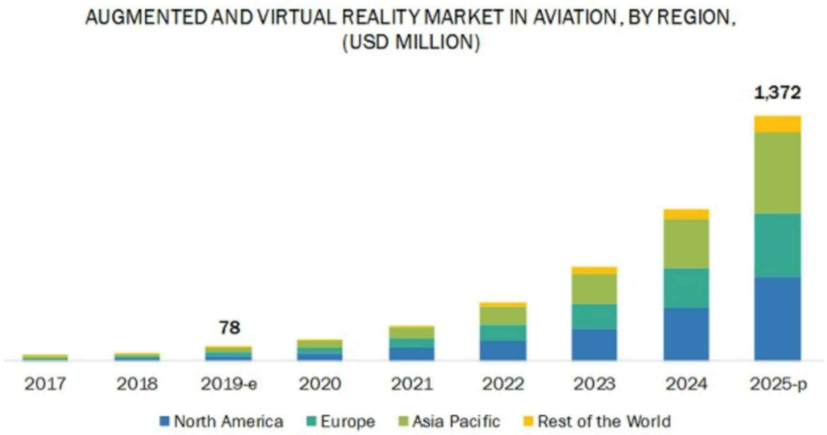
지도교수 이명호

목 차

1. 서론

연구 배경

4차 산업혁명 시대가 도래하였고, 5G 기술이 점차 보급되면서 클라우드 컴퓨팅, 인공지능(AI) 등 차세대 정보기술과 VR/AR 기술의 심화와 융합에 따른 혼합 기술 체계가 형성되었다. VR/AR 기술은 등장 초반에는 군대용 에뮬레이터 제품 개발로 시작되어, 20세기부터 애니메이션 분야로 확장되었다. 이후 동영상, 소셜 미디어, 교육 등 활용 분야가 확대되었고, 특히 2017년 이후에는 다양한 VR 상품이 널리 보급되어 많은 업종에 응용되고 있다.



현재로서는 모바일 게임과 영상 분야가 VR의 주요 소비 시장이며, 가상현실 기술은 이미 교육, 문화예술, 의료 등의 분야로 까지 그 영향력을 확장하고 있다. 특히 메타버스 기술이 발전하면서 몰입감 있는 인터랙티브 체험에 대한 수요가 확대되었고, 따라서 VR/AR 기술이 더욱 활발하게 응용될 것으로 보인다.

기존 문제점

VR 환경에서는 사용자의 감각과 행동을 직접적으로 다루기 때문에, VR 환경에서의 사용자 입력과 그에 따른 오브젝트의 반응 등 전체 상호작용을 기록하여 데이터화하는 작업이 중요해졌다. 특히 VR 환경에서의 사용자 반응 연구를 위해서는 웨어러블 센서를 통한 사용자 반응 연구와 VR 환경 내의 모든 이벤트를 기록(log)하고 다시 재생(replay)하는 과정이 필수적이다.

그러나 VR 환경에서의 사용자 경험 평가 툴들은 VR 기술의 성장속도를 따라잡지 못하

고 있다. VR 콘텐츠는 사용자의 감각과 행동에 밀접하게 반응한다. 하지만 기존 사용자 경험 평가 툴들은 2D 환경 콘텐츠에 맞춰져 있어, VR 환경에서 사용하기에는 적절치 않다.

현재 Unity asset store에서 제공하고 있는 툴은 화면 녹화 및 영상 속도 제어 기능정도만 제공하거나, 2D 모니터를 위한 리플레이 툴이기 때문에 VR 환경에서 사용하기엔 적절치 않다. 그리고 VR 환경에서 발생하는 이벤트와 해당 이벤트로 인해 발생한 다른 이벤트나 현상들을 추적하기 어렵고 인과관계를 명확하게 알 수 없다. 이러한 점으로 인해 사용자들은 VR 경험을 개선하는 것이 어렵고, 개선하기 위한 대안이 필요하다.

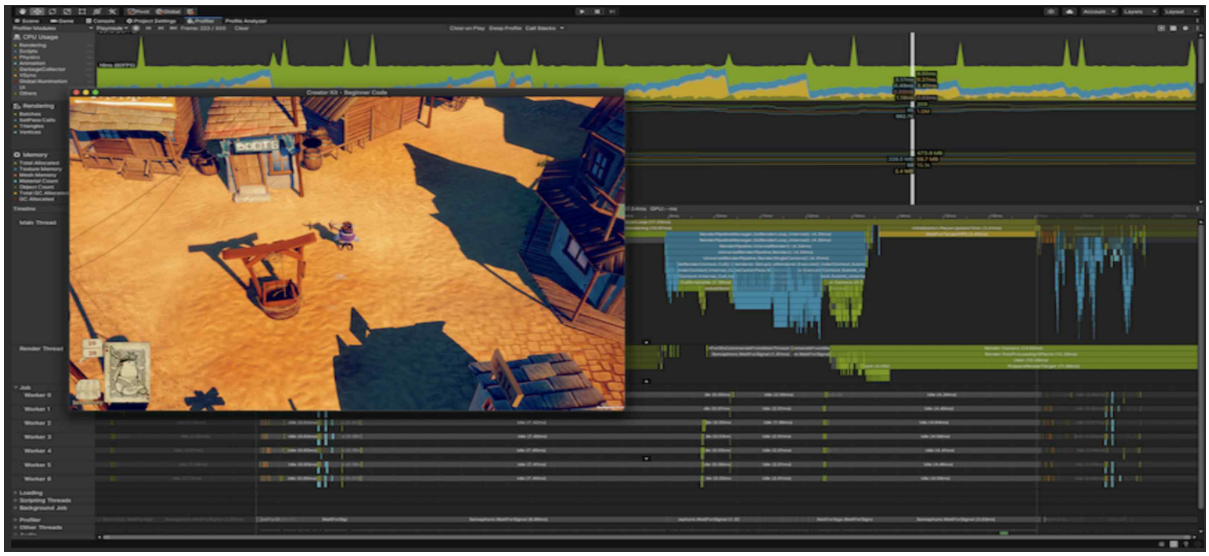


그림 Unity Engine Profiler

Unity 개발자용 툴인 Unity Profiler 또한 GPU, CPU, 메모리, 렌더러, 오디오 등의 콘텐츠 성능 데이터에 초점이 맞춰져 있어, 사용자 경험 평가를 위한 툴로는 적합하지 않다.

연구 목표

본 과제에서는 VR 환경에서 콘텐츠 내의 사용자와 사용자를 둘러싼 모든 이벤트 데이터의 기록(Log) & 다시 재생(Replay) 툴 설계를 목적으로 한다. 이러한 툴은 해당 시점을 VR 환경에서 재현하고, 특정 이벤트와 현상들을 유기적으로 연결하고, 데이터를 활용할 수 있도록 만든다. 기존 VR 리플레이 툴의 한계점 파악과 보완을 중점으로 연구할 것이

며, 이를 통해 VR 컨텐츠 개발자와 사용자 간 상호작용을 더욱 편리하게 할 것이다. 사용자들에게 더욱 편안한 VR 사용 환경을 제공하고자 한다.

2. 연구 배경

Unity

리플레이 엔진의 작업 환경을 Unity로 설정하였다. Unity 2019.4 버전부터 제공된 XR (VR + AR) Interaction Toolkit에서는 VR 헤드셋에 상관 없이 작동 할 수 있는 벤더리스한 작업 환경을 제공하고 있으며, 기존 Update문에서 계속 상태값을 체크하던 입력 방식에서 Action Base로 입력을 받도록 변경 되어 더욱 편리하게 VR 개발을 할 수 있다. 이러한 장점들 때문에 XR Interaction Toolkit을 활용헤 Replay Engine을 개발하였다. 그리고 착용 벤더는 대중적으로 가장 많이 사용되는 meta의 오큘러스 퀘스트2를 사용하였다. 해당 VR 환경에서 사용자의 상태, VR Controller의 상태 그리고 VR 환경상의 다른 오브젝트들의 상태를 녹화하였다.

Intel RealSense

여러 Depth Camera 중 RealSense 카메라를 선택하였다. 이미지 데이터 뿐만 아니라 깊이 데이터도 저장할 수 있기 때문에 사용자의 행동 정보를 모두 저장하도록 하였다. 또한 녹화에 그치지 않고 녹화된 정보를 이용해 스켈레톤 데이터를 추출 할 수 있도록 하였고 Replay Engine에 표현될 수 있도록 하였다.

3. 연구 내용

Replay Engine을 통해 기존 VR 환경에서 사용자의 상태를 녹화함과 동시에 리얼 센스 매니저를 통해 실제 사용자의 반응 또한 측정하여 개발자들의 사용자 반응 측정을 더 용이하게 만들 생각으로 연구를 진행하였다.

Replay Engine 설정

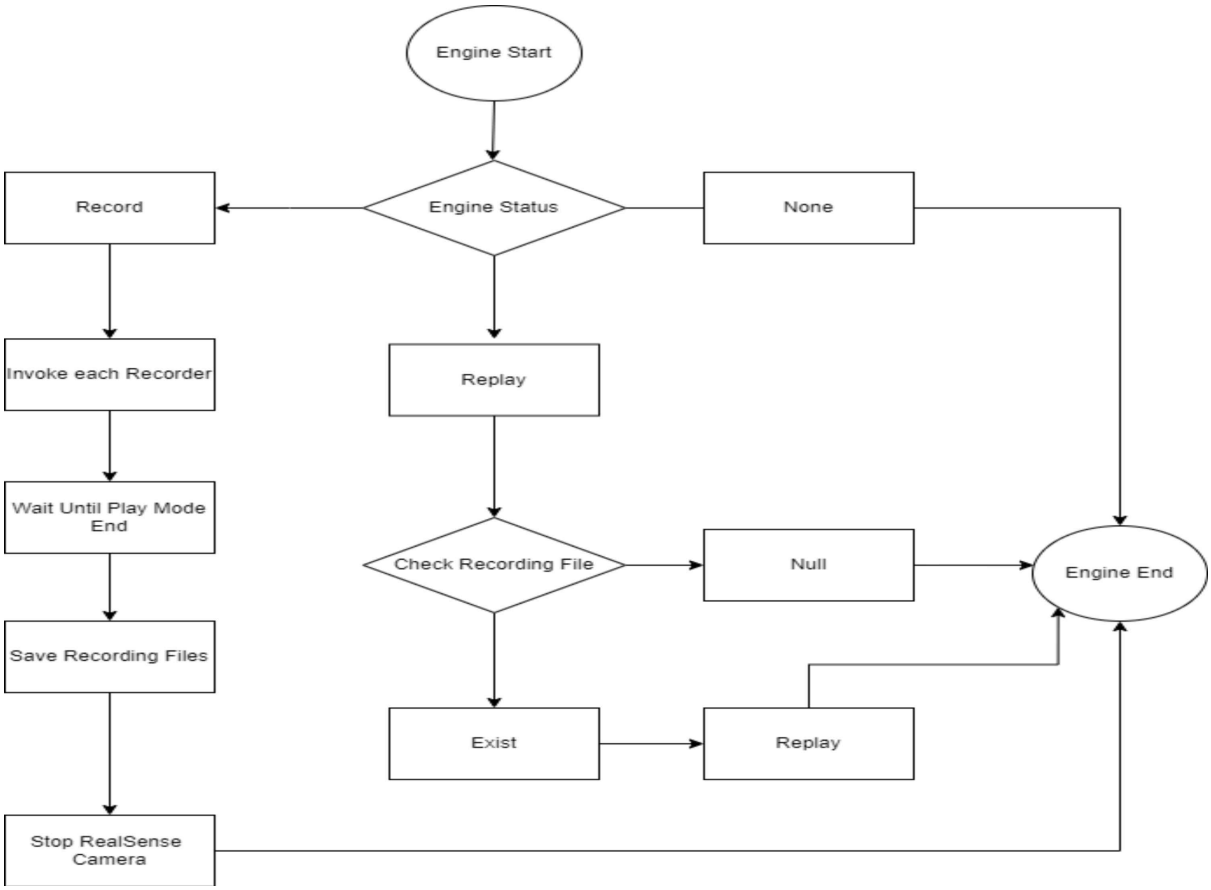


그림 2 Replay Engine 플로우 차트

Unity 상에서 Replay Engine에서 개발자는 XR Origin(XR 환경 상에서 사용자의 중심으로 설정된 오브젝트) 및 Controller 그리고 각각의 Object에 우리가 만든 Recorder파일을 부착시키도록 하였다. 각각의 오브젝트에 부착된 Recorder Script를 ReplayManager Script를 통해 중앙 관리 하도록 만들었다. Unity Editor 상에서 Play 모드로 들어갈 때, Replay Manager의 상태값을 Recording, Replaying, None 이렇게 세 가지 상태로 들어 갈 수 있도록 하였고 Recording 상태일 때는 등록된 오브젝트들의 상태를 ScriptableObject라는 유니티의 데이터 저장 형식을 사용하여 로깅하고, Replay 상태일 때는 로깅된 데이터를 시간에 따라 동기화 시키는 방식으로 리플레이 엔진을 구현하였다.

3.1.1. XR Origin 및 Controller Recorder

Recording 상태일 때는 녹화 시점으로부터 현재 시간, XR Origin의 position, rotation, scale 값 그리고 Origin의 child object인 Main Camera의 Position, Roation 값과 각

Controller 들의 position 값을 저장하였다. XR Origin은 Controller 및 Camera의 부모 오브젝트이기 때문에 다른 오브젝트들의 위치 및 회전값을 XR Origin에 대한 상대 좌표로 설정해 주었고, 개발자들의 다른 프로젝트에서 특별한 상황이 없는 이상 XR Origin은 원점으로 놓기 때문에 XR Origin의 값은 World 좌표계를 기준으로 저장하였다. 녹화 시간을 기준으로 값들을 가져와 동기화 문제를 걱정했으나, 프로젝트가 그렇게 크지 않아 시간 차이는 없었다. XR Controller의 녹화는 XR Interaction Toolkit에서 제공하는 XR Controller Recorder Script를 사용하였다. Controller Recorder는 World 좌표계를 기준으로 위치 및 회전 값이 저장되기 때문에 녹화시의 좌표계와 리플레이 좌표계가 맞지 않는 문제가 있어 리플레이시에는 부모 좌표계를 원점으로 바꿔주었다.

3.1.2. Object Recorder

Object Recorder에서는 ObjectRecorder Script가 붙은 오브젝트의 자식 오브젝트들을 DFS 방식으로 하나하나 가져오고 가져온 자식 오브젝트들의 Transform 값을 로깅하였다. Object Recorder를 처음에는 Scene이 로딩될 때, Scene 산하의 모든 오브젝트들을 가져와 Recording 시킬 생각이었으나, 그렇게 구현할 시 상태가 변하지 않는 오브젝트들을 계속 로깅하여 메모리가 낭비된다는 문제점과 계층구조가 꼬여서 하나의 Recording File에 저장되는 Data가 너무 많아지는 문제가 발생하였다. 그래서 개별 오브젝트에 사용자가 Recorder Script를 붙여서 사용할 수 있도록 설계하였다. Object별로 Recorder를 붙였을 때, 휴머노이드의 경우 Unity Asset Store의 Unity Chan 기준 267개의 자식 Transform Data를 로깅했고, 거미 오브젝트에서는 87개, 간단한 머미 오브젝트는 37개의 자식 트랜스폼 Data를 로깅했다. 로깅하는 데이터가 너무 많아 리플레이시 메모리를 많이 먹지 않을까 걱정했지만, 기존 애니메이션 녹화 방식과 동일해 부하가 많이 걸리지는 않았다. 리플레이 시에는 한 프레임에 동작을 수정하는 스크립트가 리플레이 스크립트와 애니메이터 두개가 작동하기 때문에 에니메이터를 SetActive 값을 false로 바꿔주고 동작하게 하였다.

RealSense 카메라로 녹화 및 스켈레톤 데이터 추출

RealSense 카메라를 통해 게임 중인 플레이어를 녹화한다. 그 후 녹화한 파일을 다시 불러옴으로써 스켈레톤 데이터를 추출하고 해당 데이터를 통해 Unity 상에 표시하고 게임 오브젝트에 값을 넣도록 해주었다.

3.1.3. 저장 방식

RealSense 카메라를 실행하고 값을 저장할 때는 ROS에서 사용하는 데이터 로깅 형식인 bag 파일 형식을 사용하였다.

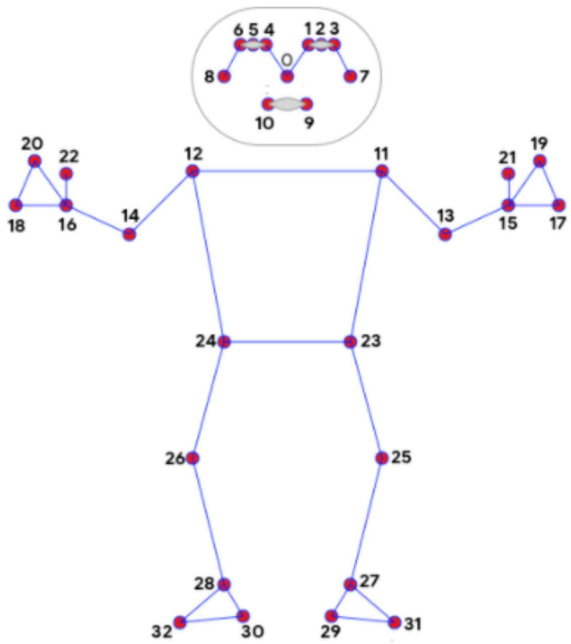
AVI 파일은 높은 프레임 속도에서는 파일의 크기가 커지게 되고, 일반 RGB 데이터가 아닌 깊이, 적외선, 모션 데이터와 같은 다양한 유형의 데이터가 손실 될 수 있다는 단점이 있다. 만약, 이런 추가 데이터를 손실없이 저장하고 싶다면 몇 가지 작업이 추가로 요구된다. 하지만 bag 파일 같은 경우에는 다양한 데이터 스트림과 해당 데이터 스트림의 타임스탬프를 저장하여 효율적이고 원본 데이터를 그대로 보존하기 때문에 데이터의 무결성이 유지된다는 장점이 있어 bag 파일 형식을 채택하였다.

3.1.4. 스켈레톤 데이터 추출

처음에는 Nuitrack 이라는 툴을 사용하려고 하였으나, 값이 제법 나가는 툴이라 사용하지 않고 Google에서 제공해주는 MediaPipe 라는 툴을 사용하여 스켈레톤 데이터를 사용하였다.

MediaPipe는 Google에서 개발한 오픈소스 머신러닝 솔루션 라이브러리로, 다양한 비전 태스크를 위한 프레임워크를 제공합니다. 그 중 하나는 실시간으로 사람의 포즈를 추정하는 것이다. MediaPipe의 Pose 모델을 사용하면 이미지나 비디오에서 스켈레톤 데이터를 추출할 수 있다.

스켈레톤 데이터를 추출하게 되면 다음 사진의 33개의 조인트의 위치가 정규화 된 좌표로 x,y 값이 저장되게 된다.



0. nose
1. left_eye_inner
2. left_eye
3. left_eye_outer
4. right_eye_inner
5. right_eye
6. right_eye_outer
7. left_ear
8. right_ear
9. mouth_left
10. mouth_right
11. left_shoulder
12. right_shoulder
13. left_elbow
14. right_elbow
15. left_wrist
16. right_wrist
17. left_pinky
18. right_pinky
19. left_index
20. right_index
21. left_thumb
22. right_thumb
23. left_hip
24. right_hip
25. left_knee
26. right_knee
27. left_ankle
28. right_ankle
29. left_heel
30. right_heel
31. left_foot_index
32. right_foot_index

그림 . 33 pose landmarks

MediaPipe에서도 깊이 데이터와 비슷한 z 값이 있긴 하지만, 이는 depth data를 이용한 것이 아니라 추정 값으로 정확하지 않다는 단점이 있다. 이를 위해 MediaPipe를 이용해 얻은 각 조인트의 x,y 값을 이용해 이미지에서의 조인트 위치를 찾아낸 후 깊이 데이터를 사용하는 것으로 하였다.

실제로 MediaPipe에서 얻은 z 값을 통해 만든 스켈레톤 형상과 RealSense 카메라의 깊이 데이터를 통해 만든 스켈레톤 형상의 자세가 차이가 있었다. 바른 자세로 촬영을 하였으나 RealSense 카메라의 깊이 데이터로는 바른 자세로 서 있는 것으로 스켈레톤 형상이 만들어졌던 반면, MediaPipe에서 얻은 z 값을 통해 만든 스켈레톤은 상체가 앞으로 굽어 있는 형상을 취하고 있었다.

하지만, 특정 자세로 인해 조인트에 해당하는 깊이 데이터가 측정되지 않는다면 RealSense 카메라로는 스켈레톤의 형상이 완전히 표시가 되지 않는 단점이 있으나 MediaPipe 에서는 추정 값을 통해 정확하지는 않지만 스켈레톤의 형상이 표시가 된다는 차이점이 있다.

3.1.5. Filtering(후처리)

스켈레톤 데이터의 x,y 값에는 크게 문제가 없었으나, z 값이 특정 문제로 인해 측정이 되지 않는 경우가 발생하고, 깊이 데이터로 스켈레톤 형상을 만들면 떨림이 너무 심하게 나타나는 현상이 발생했다,

처음에는 k-NN을 통해 MediaPipe의 z값과 RealSense의 깊이 데이터의 상관관계를 학습시키고 RealSense 카메라에서 측정되지 않은 부분을 보완하려 하였다. 하지만 MediaPipe도 추정치일 뿐만 아니라, RealSense 카메라의 깊이 데이터도 체크가 되지 않는 부분도 있어 떨림 현상이 더욱 심하게 발생하였다. 이로 인해 RealSense 카메라 자체의 노이즈의 문제라고 판단하여, 칼만 필터를 적용해보았으나 위와 크게 바뀌는 것이 없었다.

추후, 깊이 데이터에서 에지부분의 깊이 데이터는 0으로 측정되기 때문에(측정불가) 이를 보정해주고, 노이즈도 어느 정도 발생하는 것을 억제시켜야 함을 발견하였고 이를 해결하기 위해 RealSense SDK에서 제공해주는 필터들을 사용하였다.

먼저, Spatial Filter (공간 필터)는 깊이 이미지 내의 노이즈를 감소시키고, 세부 사항을 보존하면서 경계를 부드럽게 만들어준다. 이 필터는 현재 픽셀과 인접한 픽셀들의 값을 사용하여 깊이 값을 평균화하는 방식으로 작동한다.

Temporal Filter (시간 필터)는 연속적인 프레임 간의 깊이 값을 평균화하여 동적 노이즈를 감소시키고 깊이 변화를 부드럽게 만들어준다. 이전 프레임의 정보를 사용하여 현재 프레임의 깊이 값을 평균화하며, 이는 깊이 값의 변동성을 감소시키는 원리로 작동한다.

Holes Filling Filter (홀 채우기 필터)는 깊이 이미지 내의 누락된 값을 보완하기 위해 사용된다.인접한 픽셀의 값을 사용하여 깊이 이미지 내의 누락된 값을 채우는 방식으로 작동한다.

그리고 OpenCV에서 제공해주는 median filter 또한 사용해주었다. 이는 픽셀과 그 주변 픽셀들의 값 중 중앙값(median)을 선택하여 픽셀 값을 대체하는 방식으로 'Salt-and-Pepper' 노이즈 제거 같이 극단적인 노이즈(예: 밝은 흰색 또는 어두운 검은색 점)를 효과적으로 제거하는 데 특히 효과적이고 작은 커널 크기에서는 에지도 어느 정도 보존할 수 있다는 장점이 있다.

이렇게 필터링하여 얻은 깊이 데이터를 선형보간을 이용하여 최대 30fps 밖에 지원되지 않는 RealSense의 깊이 데이터를 60fps로 자연스럽게 표현되도록 하였다. 원래 IK(Inverse Kinematic)을 통해 자연스러운 스켈레톤의 움직임을 보장하려 하였으나 실패하였다. IK는 스켈레톤에서 각 조인트의 부모-자식관계가 적용되어 있어야 하지만 스켈레톤 간의 부모-자식 관계를 표현하기 위해서는 척추 중심이 있어야 하지만 MediaPipe는 몸통이 직사각형(중심이 없음)으로 되어있고, 해당 부분의 작업자의 Unity 사용 미숙으로

인해 이를 구현에 실패함으로써 적용시키지는 못 하였다.

3.1.6. Unity에 표현

Replay를 하게 되면 MediaPipe를 통해 얻은 x, y 값과 RealSense 카메라의 가로, 세로 화소 수를 곱해주고 RealSense를 통해 얻은 깊이 데이터에 일정 계수를 곱해줌으로써 최대한 실제 데이터와 유사하게 표현해줄 수 있도록 하였다.

테스트 환경 구성

Replay Engine의 성능을 측정할 수 있도록 간단한 테스트 환경을 구성하였다.

4. 연구 결과 분석 및 평가

Unity Engine상에서 프레임마다 상태를 녹화하게 되면 60 fps를 기준으로 각각의 오브젝트에서 초당 60개의 상태 데이터가 저장된다. 그렇기 때문에 성능적인 저하가 우려되었고 우리가 만든 테스트 환경에서 성능차이를 테스트 하였다.

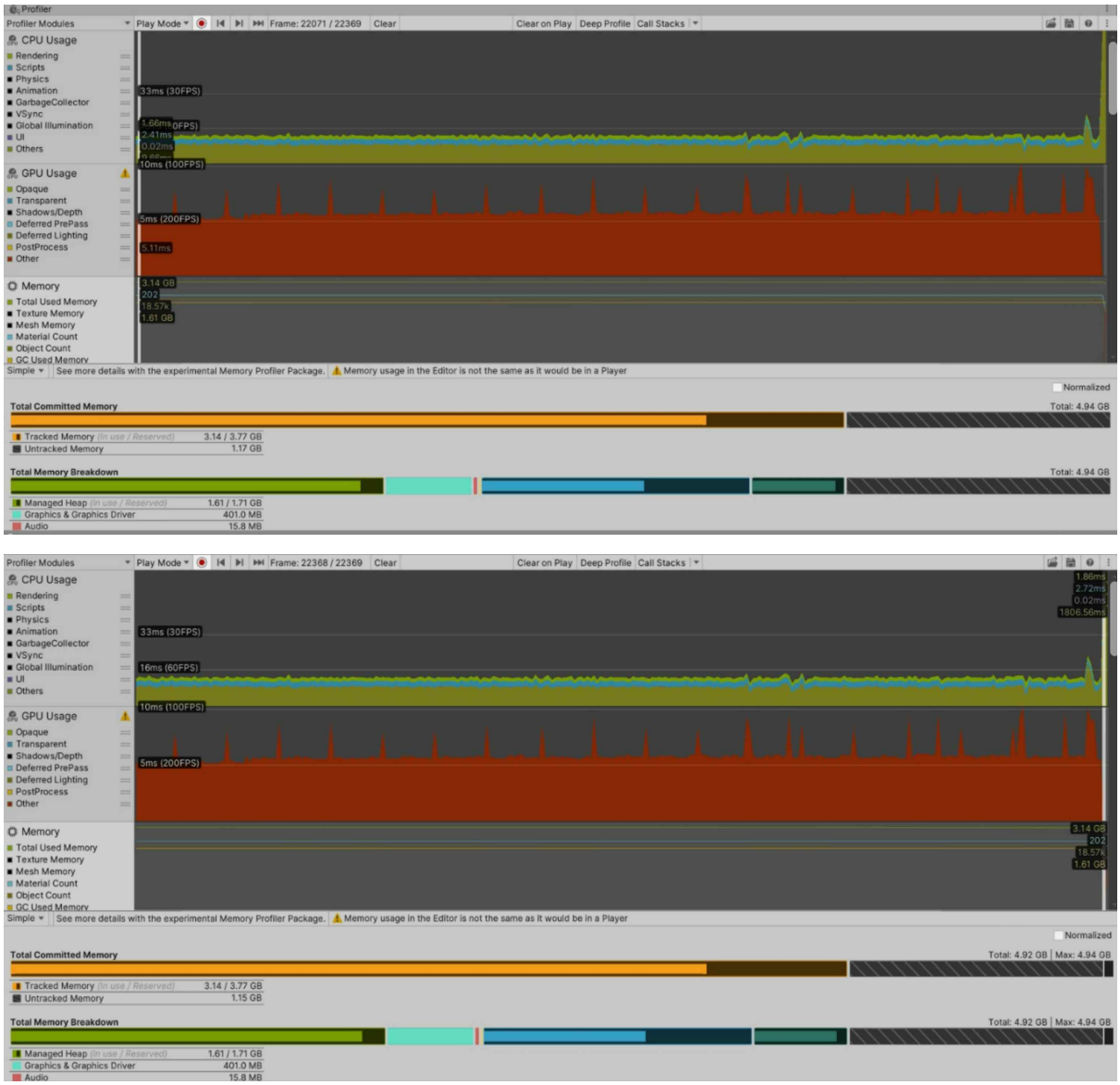


그림 4 리플레이 엔진 미사용시 CPU, GPU, 메모리 성능

리플레이 엔진을 미사용했을 때는 플레이 모드 시 평균적으로 3.14GB의 메모리를 사용하였고, 예약된 메모리 까지 합치면 3.77GB를 사용하였다. 미추적된 메모리 1.17GB와 합치면 4.94GB를 사용하였다.

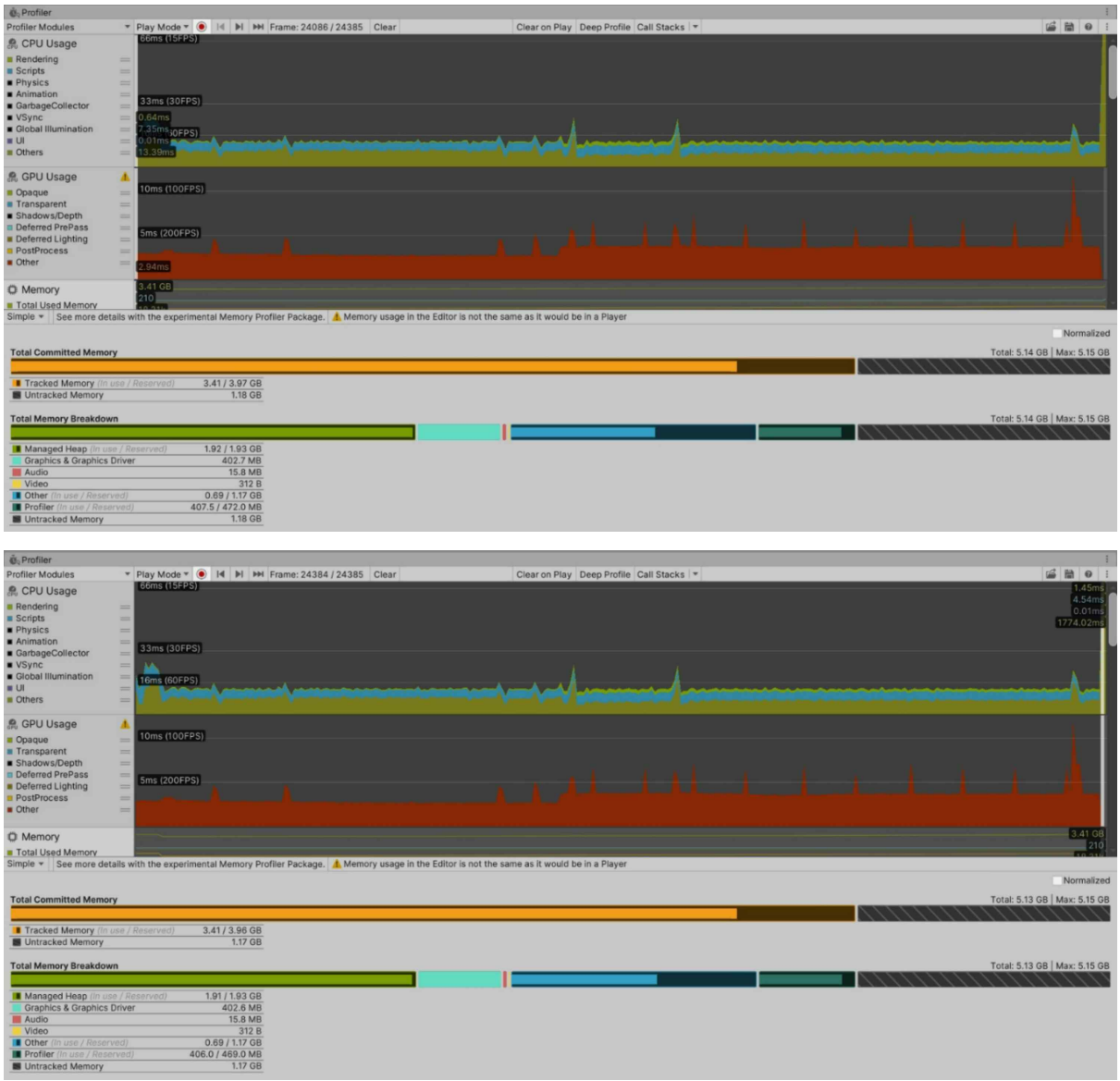


그림 5 리플레이 엔진 레코딩 모드일 시 CPU, GPU, 메모리 성능(리얼센스 미사용)

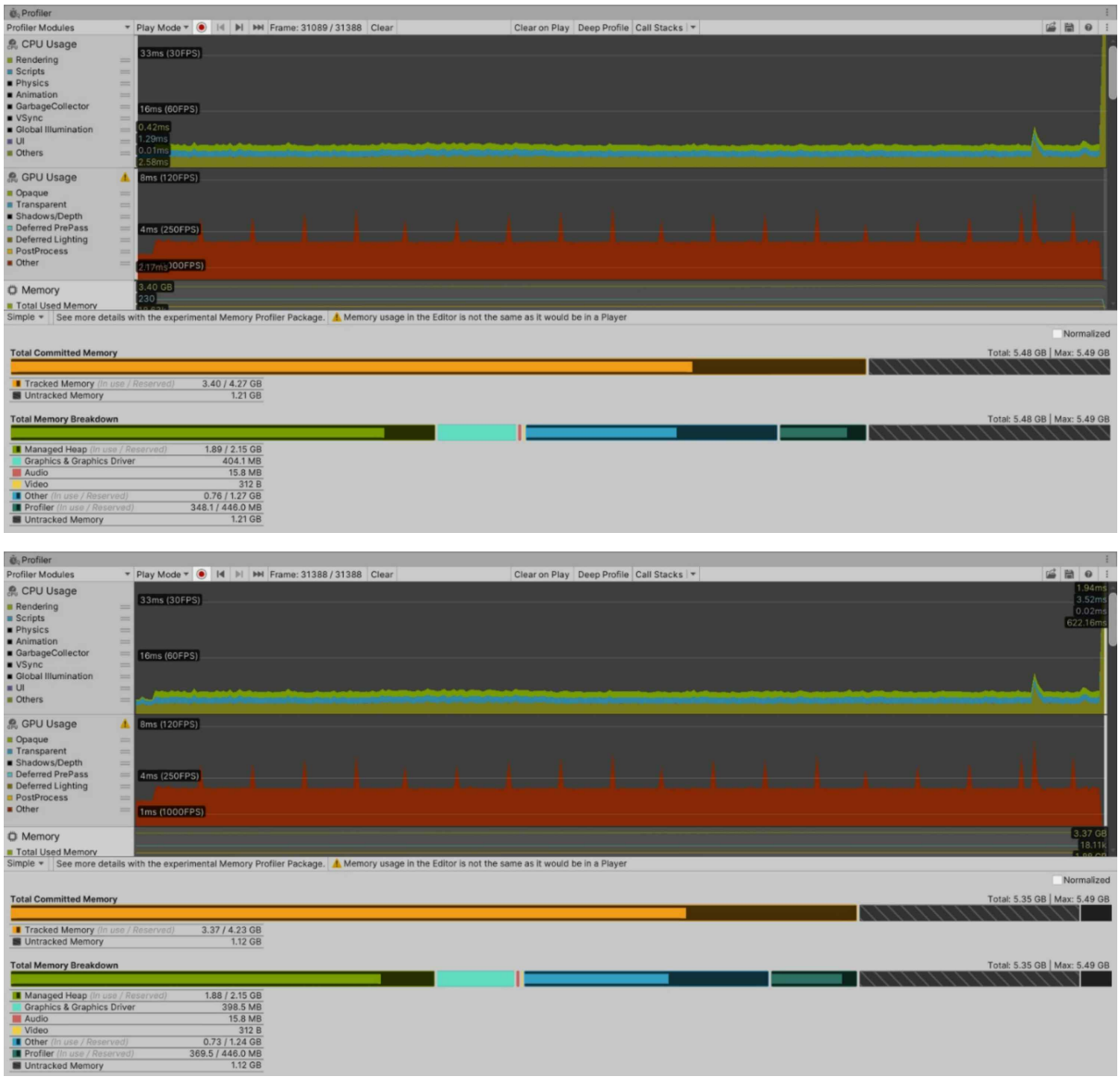


그림 6 리플레이 엔진 리플레이 모드일 시 CPU, GPU, 메모리 성능(리얼센스 미사용)

리플레이 엔진이 주는 성능 부하를 측정하기 위해 리얼센스 매니저를 끄고, 성능 변화를 측정하였다. 레코드 모드일 시는 플레이시 평균 사용 메모리가 3.41GB, 예약 메모리는 3.97GB, 미추적 메모리는 1.18GB로 전체 5.14GB를 사용하였다. 리플레이 시에도 그와 동일한 메모리가 사용되었다. 레코드와 리플레이시의 메모리 변화가 없는 것으로 보아 데이터를 읽고 쓰는 과정에서의 메모리 차이는 거의 없는 것으로 보인다. 기존 유니티의 애니메이터와 비슷한 원리로 동작하게 만들었으나, 리플레이 엔진 사용시 메모리가 0.2GB 더 사용된 것으로 보아 최적화가 덜 된것으로 유추된다. 작업 환경 상에서 26개의 오브젝트가 존재하였는데 각 오브젝트 당 평균적으로 0.007GB 정도의 부하를 더 주는 것 같다. 실제 게임에서는 26개 보다는 훨씬 많은 수의 오브젝트가 사용되므로 최적화를 더 시킬 필요가 있어 보인다.

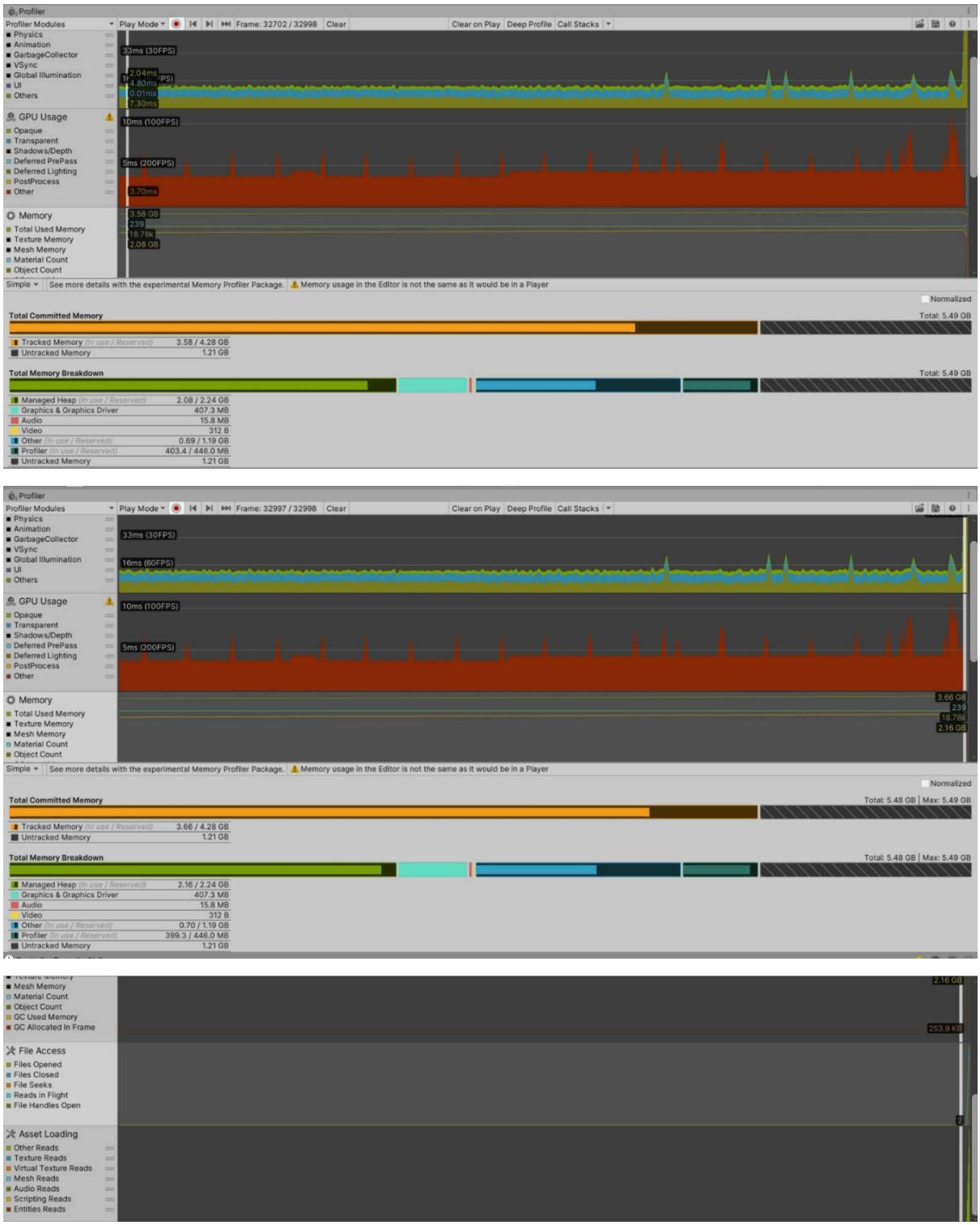


그림 7 리얼센스 작동 시 레코딩 성능(마지막 사진은 File Access가 후처리시 확 튀는 모습)

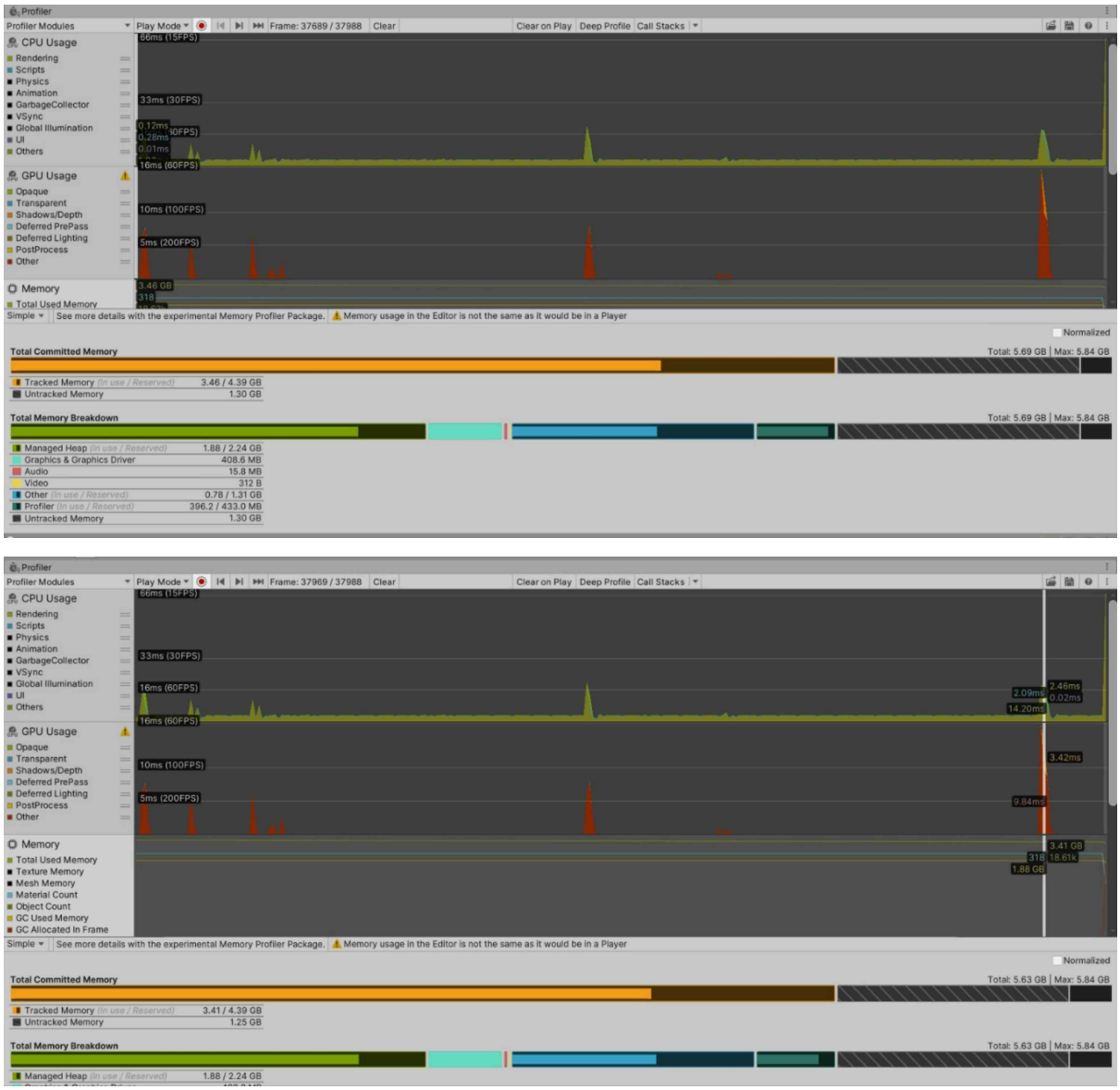


그림 8 리얼센스 작동시 리플레이 성능

리얼 센스를 작동시킨 상태에서 사용자 반응 측정까지 같이 진행 시킨 결과 레코딩 시 총 메모리 사용량은 5.48GB로 늘어났고, 리플레이시는 5.68GB로 늘어났다. 레코딩과 리플레이 시의 차이가 큰 것은 리플레이시 파일접근을 하고 코루틴을 사용해서 계속해서 사용자 조인트 구체들의 상태를 변화시킨 구현 방법 때문인 것으로 보인다. 사용자 데이터는 VR 환경에서 오직 하나뿐이므로 이러한 성능차이가 규모가 큰 VR 환경이라고 해도 더 늘어날 것으로 보이지는 않는다. 시간이 더 주어진다면 레코딩 후처리시 데이터를 재가공해 리플레이에서 발생하는 메모리 소요를 줄일 수 있을 것으로 보인다. 레코딩 시에 문제점은 레코딩이 끝난 시점에 프로세스를 따로 파서 진행했던 리얼센스 파싱 데이터가

한번에 들어오면서 File Access가 확 튀고 그만큼 후처리 시간이 길어진다는 점이다. 그리고 리얼센스 녹화시간이 길어질수록 bag파일의 용량도 많이 커지는 문제점이 있다.

5. 시연 계획

1. RealSense 카메라 1대, Oculus Quest2 1대, 해당 연구과제의 Unity XR Interaction Toolkit(Unity Version 3.15f1), 노트북 1대를 준비.
2. 노트북에 OpenCV, Intel RealSense SDK를 설치하고, python 패키지 중 OpenCV, pyrealsense2, numpy를 설치해준다.
3. Unity Package를 실행시키기 전에, RealSense 카메라와 Oculus Quest2를 노트북에 연결시켜준다.
4. Unity에서 Recording Prefab을 Scene에 로드한후 recording mode로 설정해 준 뒤 실험용 가상환경을 실행시켜준다.

   (recording mode로 실행해야 RealSense 카메라가 자동으로 실행되며 녹화가 시작된다.)
5. 가상 환경 플레이가 끝난 후, 플레이를 종료시키면 OnDisable(유니티 플레이가 꺼질때 자동으로 실행되는 함수)에서 레코딩 파일을 후처리 해준다.

   (종료시켜전 뒤, 바로 스켈레톤 데이터를 추출하고 후처리를 실시하기 때문에 시간이 약간 소요되니 기다려야 한다.)
6. Package에서 replay mode로 변경하여 실행하게 되면 스켈레톤 데이터가 정상적으로 생성되고, Oculus Quest2에 입력했던 값도 정상적으로 리플레이 되는 것을 알 수 있다.

6. 결론 및 향후 연구 방향

초기의 목적은 유니티 환경상에서 XR 사용자와 관련된 환경의 모든 상태값을 로깅하는 것이었으나 유니티 작업능력 부족으로 Transform 데이터 정도밖에 저장하지 못했다. XR 환경이라는 일반적인 환경 말고 특정 VR 게임이나 환경에서 상태 데이터를 아는 상태였다면 구현할 수 있을 것으로 보였다. 대신 각 Transform을 다 파싱함으로써 리플레이시

에 내부 데이터를 제외한 VR 환경은 그대로 재구성할 수 있었다.

그리고 RealSense 카메라를 통한 작업에서도 녹화를 할 수 있는 툴은 빠르게 만들었으나 스켈레톤 데이터 작업 미숙과 Unity 환경 사용 미숙으로 인한 문제점들이 있었다. 대표적으로 필터링에 대한 부분과 IK 미적용 부분이 있겠다. 그럼에도 SDK에서 제공되고 OpenCV의 median 필터를 활용하는 등 여러가지 필터를 사용하였고 떨림 현상과 에지 부분의 깊이 데이터를 보완하는 등 어느 정도 필터링에 대한 효과가 있었음을 확인하였다.

사용자 반응 측정 부분에서는 조원이 중간에 그만두는 바람에 기존에 계획했었던 심박수 측정 모델을 완성하지 못한 부분이 아쉬움으로 남는다. 향후 연구 기간이 더 주어진다면 사용자 정의 데이터를 파싱하고 리플레이 할 수 있는 방법과 사용자 반응 측정 모델에 갤럭시 워치를 연동한 심박수 측정 기능을 추가 할 생각이다.

또한 RealSense 카메라에서 측정이 되지 않는 부분(몸통에 의해 가려진 팔 등) 또한 유추할 수 있도록 할 것이며, skeleton에서 직접 Joint를 더 만들어 부모-자식 관계를 쉽게 표현할 수 있도록 하여 IK를 실현하고 더욱 자연스러운 행동을 표현하도록 할 것이다.

7. 피드백 반영 사항

중간 보고서에서 실험용 가상환경인 좀비 게임에 관한 내용이 너무 많아 개발 주제에 혼돈이 생긴다는 피드백이 있어, 최종 보고서에는 해당 내용을 삭제 했습니다. 구성원별 역할 분담에 관한은 최현호 조원이 Unity 환경상에서 일어나는 상황을 녹화하는 툴을 만들고, 이희근 조원이 리얼센스를 통해 사용자 행동 데이터를 로깅하는것으로 역할을 분담했습니다. 숫자 및 이미지를 통해 어떤 연구를 하였고 고민하였는가에 대해 나타내라는 피드백에 대해서는 실험 결과에 녹화 툴을 사용하기 전후 메모리 사용량과 리얼센스를 연결하기 전후 메모리 사용량을 비교해 저희 녹화 툴을 실제 사용할 시 개발 환경에 얼마만큼의 영향을 주는가를 측정하려고 했습니다.

8. 구성원별 역할 및 개발 일정

이름	역할
최현호	VR 헤드셋 전용 레코딩 툴 구현 Unity 환경상의 오브젝트 레코딩 툴 구현 실험용 VR 환경 재구성.
이희근	RealSense 녹화 및 필터링 툴 개발 MediaPipe로 스켈레톤 데이터 추출 및 RealSense Depth data 추출 스켈레톤 데이터 Unity 환경에 표현
김자원	자퇴
공통	보고서 작성 발표 및 시연 준비

9. 참고 문헌

https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.3/manual/index.html
- Unity XR Interaction Toolkit

https://dev.intelrealsense.com/docs/post-processing-filters
- Intel realsense document

https://github.com/google/mediapipe/blob/master/docs/solutions/pose.md
- MediaPipe Model Pose