현실과의 상호작용을 결합한 증강현실 게임 제작

2019102225 전세계

요 약

증강현실(AR)을 지원하는 기기의 폭넓은 보급에도 불구하고 AR 기술을 적절히 활용하는 게임 콘텐츠는 많지 않다. 본 프로젝트에서는 AR 만의 장점을 살린 인터랙티브 콘텐츠를 제작하고자 현실과의 상호작용을 결합한 AR 게임을 개발한다. 현실의 동작이 게임 속에 영향을 미치고 주변 환경에 따라 게임 환경이 변화하는 시스템의 개발을 목표로 한다.

1. 서론

1.1. 연구배경

2022 년 현재, 대부분의 스마트폰 사용자들이 증강현실(AR) 기술을 활용할 준비가 되어있다. 시장조사업체 스트래티지 애널리틱스(SA)에 따르면 전 세계 스마트폰 평균 교체 주기는 2020 년 기준 43 개월로 나타났다. 삼성에서 출시한 스마트폰이 ARCore 를 지원하기 시작한 시기가 2016 년 3 월임을 감안할 때, 대중들이 보유한 대부분의 기기가 AR 기술을 지원하는 것으로 추측해볼 수 있다.

이러한 폭넓은 AR 지원 기기의 보급에도 불구하고 AR 을 적절히 활용한 성공적인 콘텐츠를 찾기란 쉽지 않다. 2022 년 9 월 기준 구글플레이 인기차트 상위 100 개의 게임을 조사해봤을 때, AR 을 활용한 게임은 '포켓몬 고(Pokémon GO)'가 유일하다. 그마저도 포켓몬이라는 강력한 IP 를 활용한 AR 선택적(optional) 앱임을 감안하면 AR 의 장점을 살린 성공적인 콘텐츠가 전무하다고 봐도 무방하다.

본 연구에서는 증강현실 콘텐츠의 소외 원인으로 '현실과의 상호작용 부족'을 제시하고 이를 해결하고자 한다. 기존에 출시된 게임들을 보았을 때, 대중들이 AR 게임에 갖는 이미지는 GPS 기반의 반쪽짜리 증강현실일 것이다. 증강현실의 본질이 현실과 가상을 동시에 즐길 때 발생하는 시너지임을 고려하면 기존의 AR 게임들은 상호작용 요소가 부족하다. 현실을 단순히 게임 내 배경으로만 활용하는 기존 게임에서 탈피하여, 현실에서의 동작이 게임 속 세계에 영향을 미치는 상호작용 요소를 적극적으로 활용하는 게임 콘텐츠를 개발하고자 한다.

1.2. 연구목표

증강현실의 두 가지 장점을 살린 게임 개발을 목표로 한다. 하나는 현실과의 상호작용을 접목시키는 것, 다른 하나는 주변 환경에 따른 동적 오브젝트 생성이다.

현실과의 상호작용은 플레이어의 일상적인 행동이 게임 속에 영향을 미치는 것으로 이루어진다. 예를 들어 문을 여는 행동이 게임 속에도 영향을 미쳐, 문밖에서 NPC 를 마주하는 식의 이벤트가 발생한다. 이러한 상호작용의 추가를 통해 현실과 게임의 경계선을 허무는 것을 목표로 한다.

주변 환경에 따른 동적 오브젝트 생성은 게임을 플레이하는 장소에 따라 자연스러운 사용자 경험을 제공하는 것을 목표로 한다. 사용자마다 서로 다른 공간에서 플레이하는 증강현실 게임의 특성을 살려 맥락에 맞는 환경을 제공하고자 한다. 평면, 깊이 인식을 통해 주변 환경을 모델링하여 장식 요소와 오브젝트의 배치를 동적으로 시행하는 방식을 목표로 한다.

추가적으로 다른 사용자와 함께 플레이할 수 있는 멀티플레이 기능을 제공한다. 기존 GPS 기반 증강현실 게임의 멀티플레이 기능과 다르게 정교한 로컬 트래킹 기능을 제공하여 높은 수준의 몰입감을 부여한다. 트래킹 기능은 Cloud Anchor 기술을 통해 각 기기의 좌표계를 동기화하는 것을 목표로 한다.

종합적으로 게임과 현실의 괴리감을 최대한 줄여 몰입감 높은 콘텐츠를 개발하는 것이 목표이다. 일상적인 행동 맥락(context) 속에서 게임을 조작하고 이질감 없는 가상 환경을 구성하여, 플레이어에게 높은 수준의 몰입감을 제공하고자 한다.

2. 관련연구

2.1. 증강현실 소프트웨어 개발 도구

2.1.1. ARCore

ARCore 는 Google 에서 개발한 증강현실 소프트웨어 개발 도구이다. Android, iOS, Unity, Unreal, Web 등의 많은 플랫폼에서의 개발을 지원한다. Hit-test, Placement, Depth, Lighting Estimation 등의 증강현실 어플리케이션을 개발하기 위한 기능들을 제공한다.

본 프로젝트에서는 Android 디바이스를 타겟으로 하기 때문에 ARCore 를 사용하여 개발한다.

2.1.2. ARKit

ARKit 은 Apple 에서 개발한 증강현실 소프트웨어 개발 도구이다. iOS 기반의 디바이스만 지원한다. ARCore 에서 제공하는 기능들을 대부분 지원하며 전후면 카메라 동시 지원, Motion Capture 등의 기능을 추가적으로 제공한다.

ARKit 은 iOS 디바이스만 지원하기 때문에 Android 디바이스를 타겟으로 하는 본 프로젝트에서는 사용할 수 없다.

2.2. 게임 개발 도구

2.2.1. Unity

Unity 는 게임 개발 환경을 제공하는 게임 엔진이다. Android, iOS, Windows, Mac, Linux, Web 등의 거의 모든 플랫폼을 지원하는 것이 특징이다. Unity 는 Component 기반의 모듈형 개발 방식을 채택하고 있기 때문에 다른 게임 엔진에 비하여 높은 생산성과 쉬운 개발 난이도를 갖는다.

AR Foundation 이라는 패키지를 통해 ARCore 와 ARKit 을 Extension 으로 지원한다. ARCore Extension 으로 세션을 구성하여 모든 AR 프로세스를 관리한다.

짧은 기간 내에 개발을 마쳐야 하는 본 프로젝트의 일정을 고려하여 생산성이 높은 Unity 와 AR Foundation 을 사용해 개발한다.

2.2.1. Unreal Engine

Unreal Engine 은 게임 개발 환경을 제공하는 게임 엔진이다. Unreal Engine 은 Lumen, Nanite 등의 최신 3D 기술을 탑재한 UE5 를 출시하면서 주목을 받고 있다. 모든 게임 엔진 중에서 가장 뛰어난 기술력과 성능이 특징이다.

C++를 통해 개발하므로 비교적 생산성이 떨어지며 다른 엔진과 비교했을 때 엔진 자체가 많은 리소스를 요구하는 등의 단점이 있다.

AR Core 를 플러그인(Plugin) 형태로 제공한다.

2.3. 깊이 인식

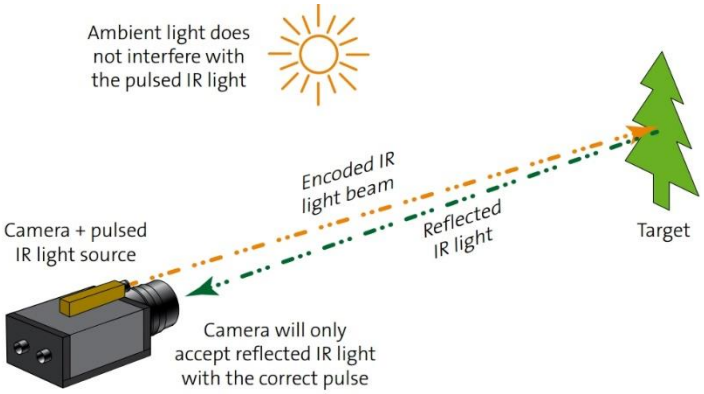
증강현실에서 현실의 장면을 3 차원 정보로 인식하고 정확한 Anchor 를 잡기 위해서는 깊이(Depth) 정보가 필요하다. 현실과의 상호작용, 주변 환경에 따른 동적 오브젝트 생성이 필요한 본 프로젝트에서는 이 깊이 정보를 통해 3 차원 메쉬(mesh)를 재구성할 필요가 있다.

2.3.1. Depth-from-motion

ARCore 에서 제공하는 Depth API 는 Depth-from-motion 알고리즘을 사용한다. 사용자가 움직이면서 동일한 장면이 여러 각도에서 촬영될 때 특징점의 위치 변화를 감지해서 거리를 계산한다. 필요한 경우 머신러닝을 통해서 작은 움직임으로도 정확한 깊이를 계산할 수 있다.

따로 깊이 카메라(Depth Camera)가 없더라도 적용할 수 있는 방식이라는 이점이 있다. 반면 움직임이 적거나 특징점이 없는 환경에서 정확도가 떨어질 수 있다. ARCore 에서는 ToF(Time of Flight)센서 등의 하드웨어가 있는 경우 해당 정보도 활용해서 정확도를 높인다.

2.3.2. Time of Flight



[그림 1] Time of Flight 작동 원리

Time of Flight 는 물체와의 거리를 직접 구하는 방식으로 깊이 정보를 인식한다. 대상에게 빛을 쏘아 반사되어 돌아오는 시간을 측정하여 실제 거리를 계산한다.

애플에서는 ToF 원리를 사용하는 라이다(LiDAR)를 아이폰 12 프로에 처음으로 탑재했다. 삼성에서는 DepthVision Camera 라는 이름으로 s20+, s20u 에 ToF 모듈을 탑재했다.

Time of Flight 는 Depth-from-motion 알고리즘에 비해 더 높은 정확도를 가졌지만 탑재된 기기가 제한적이라는 단점이 있어 많은 사용자를 대상으로 하는 어플리케이션에서는 활용하기 어렵다.

2.4. 기존 증강현실 게임 연구

가장 흥행한 증강현실 게임인 '포켓몬 고(Pokémon GO)'를 출시한 Niantic 사의 게임을 통해 기존 증강현실 게임을 분석하고자 한다. Niantic 에서 2022 년 4 월에 출시한 'AR Voyage'라는 게임은, 당사에서 자체 개발한 증강현실 SDK 인 Niantic Lightship AR 의 핵심 기능을 모두 담은 게임이다.

[그림 2] Niantic 에서 개발한 'AR Voyage' 플레이 장면

깊이 정보로 재구성한 메쉬를 통해 **[그림 2]**의 왼쪽 사진처럼 현실의 오브젝트와 충돌하고 텍스처를 입히는 기능을 제공한다. 현실의 물체에 가려지는 Occlusion 기능과 간단한 이미지 인식 기능(하늘, 풀) 또한 제공한다. 이외에도 본 프로젝트에서 제공하는 멀티플레이어 기능처럼 앵커 호스팅을 통한 로컬 멀티플레이어를 지원한다.

2.5. 기존 연구의 문제점 및 해결 방안

2.5.1. 연구의 문제점

'AR Voyage'로 대표되는 기존 증강현실 게임에서는 현실과의 상호작용을 제공하지 않는다. 현실의 물건과 충돌하는 등의 상호작용을 제공하기는 하지만, 부수적인 효과에 그치는 수준이고 게임 메커니즘적으로 발전한 케이스는 찾아볼 수 없었다. 또한 모든 조작이 화면의 버튼을 누르는 것으로 이루어졌고, 이는 게임의 몰입감을 낮추는 요소로 작용했다.

어떤 장소에서 플레이하든 똑같은 콘텐츠가 제공되는 문제점도 있었다. 주변 환경을 인식하기는 하지만 결국 오브젝트가 허공에 랜덤하게 생성되는 방식으로 게임이 진행되었고, 이는 환경 인식이 무의미하게 이루어진다고 볼 수 있다.

2.5.2. 해결 방안

2.5.2.1. 플레이어의 행동 인식과 상호작용

몰입감을 떨어뜨리는 기존의 버튼 기반 상호작용에서 행동을 통한 상호작용으로 전환할 필요가 있다. 현실에서 이루어지는 플레이어의 몇 가지 행동을 트리거로 하여 상호작용을 제공한다면 훨씬 더 높은 몰입감을 제공할 수 있다. 따로 햅틱 장치를 준비하지 않더라도 자연스러운 촉각 피드백을 줄 수 있는 장점도 있다.

2.5.2.2. 동적 게임 오브젝트 생성

동적 게임 오브젝트 생성을 통해 장소 이동에 따른 새로운 경험을 제공할 필요가 있다. 플레이어가 현실의 모습을 단순한 배경이 아닌 게임 속 환경의 일부라고 느낄 수 있도록 맥락에 맞게 동적으로 오브젝트를 생성한다. 특징적인 평면(천장, 바닥, 벽면) 인식을 통해 장식 요소를 추가하고 기타 오브젝트를 배치한다면 더 높은 수준의 몰입감을 제공할 수 있다.

3. 프로젝트 내용

3.1. 게임 개요

본 프로젝트에서는 앞서 언급한 현실과의 상호작용, 동적 오브젝트 생성 시스템이 포함된 'AR 생존 게임'을 개발한다. 가상의 자연재해가 발생한 세계에서 살아남는 디스토피아 장르의 게임이며, 실내에서 플레이하는 것을 가정한다. 게임의 최종 목표는 한정된 자원을 가지고 실내에서 일정 시간을 버티는 것이다.

플레이 중 랜덤하게 NPC 가 찾아오는 이벤트가 발생하며, 현실의 문을 여는 것으로 상호작용할 수 있다. NPC 는 플레이어에게 도움을 줄 수도, 피해를 끼칠 수도 있다. 한정된 자원 속에서 NPC 를 적절히 활용하는 것이 게임의 핵심 전략(tactic)이 된다.

게임 플레이는 크게 물품 구매 단계, 생존 단계의 두 단계로 구성되며, 각각의 상황에 맞는 오브젝트가 동적으로 생성된다. 예를 들어 물품 구매 단계에서는 벽면을 인식하고 가상의 선반을 배치하여 상점의 모습을 연출하고, 생존 단계에서는 창문에 이미지를 덧입혀 가상의 자연재해 상황을 연출한다.

3.2. 시나리오

3.2.1. 게임 준비 단계

게임 준비 단계에서는 멀티플레이를 위한 설정과 주변 인식을 진행한다. 멀티플레이에는 Cloud Anchor 를 통한 좌표계 동기화가 필요하기 때문에 Host 가 기준이 될 지점에 앵커를 설치한다. Client 들은 해당 지점에 앵커를 resolve 하여 좌표계를 동기화한다. 추가로 문을 여는 동작에 대한 인식을 위해 문 밖에서 조금 떨어진 지점에 앵커를 설치한다.

두 개의 앵커 설치가 끝나면 주변 인식을 진행한다. 플레이어가 방 내부를 360 도로 회전하며 스캔을 하면, 동시에 depth 정보를 받아와 mesh 를 생성하여 저장한다. 이 mesh 정보는 지형 충돌과 오브젝트 배치에 이용된다.

3.2.2. 물품 구매 단계

물품 구매 단계에서는 주어진 재화 내에서 생존에 필요한 물품을 구매한다. 앞선 준비 단계에서 얻은 mesh 정보를 활용해 판매할 물품들을 배치하여 상점의 모습을 연출한다. 물품 구매가 끝나면 은신처의 모습으로 주변이 바뀌며 생존 단계로 넘어간다.

3.2.3. 생존 단계

생존 단계에서는 자연재해가 끝날 때까지 실내에서 한정된 물품을 가지고 생존한다. 앞선 물품 구매 단계에서 얻은 물품들을 방 안에 자유롭게 배치할 수 있다. 생존 단계에서는 NPC 가

방문하는 이벤트가 랜덤하게 발생한다. 노크 소리를 통해 NPC 의 방문을 인지할 수 있으며, 문을 여는 것으로 상호작용을 할 수 있다. 문을 열었는지의 여부는 앞선 준비 단계에서 설치한 앵커와 거리 정보를 통해 판단한다.

플레이어는 배고픔, 건강 상태, 정신력 등의 수치를 관리하며 생존해야 한다. 구매한 물품이나 NPC 이벤트를 통하여 이 수치들을 조절할 수 있고, 기준치에 도달하면 게임오버 된다. 멀티플레이의 경우 다른 플레이어의 상태를 얼굴 증강을 통해 확인할 수 있다.

3.3. 요구사항

3.3.1. Android Device 에 대한 요구사항

- AR foundation 의 ARCore 를 지원하면서 Depth API 를 지원하는 Android 기기.
(https://developers.google.com/ar/devices)
- Android 7.0 이상, API 레벨 24 이상.

3.3.2. 현실과의 상호작용에 대한 요구사항

- 현실과의 상호작용, 특히 문을 여는 동작의 인식을 위해 추가적인 앵커가 필요하다. 문에서 일정 거리 떨어진 지점에 앵커를 설치하여 해당 위치에 NPC 가 생성되고 Occlusion 을 통해 문에 가려지도록 한다.
- 카메라에서 앵커까지의 수평 거리와 깊이 정보를 통해 얻은 NPC 방향으로의 최단 거리를 비교하여 문이 열렸는지 닫혔는지를 파악한다.

3.3.3. 주변 환경 인식과 동적 오브젝트 생성에 대한 요구사항

- 게임 준비 단계에서 주변 환경을 스캔할 때, depth meshing 알고리즘을 통해 mesh 를 생성하여 저장한다.
- 동적 오브젝트 생성을 위한 위치 선정은 mesh 정보와 평면 인식을 동시에 활용한다. 선반과 같은 오브젝트는 평면 인식으로 적절한 높이에 배치하고, 기타 장식 요소는 바닥 평면과 맞닿아 있는 mesh 주변에 랜덤하게 배치하는 것으로 은신처의 모습을 연출한다.

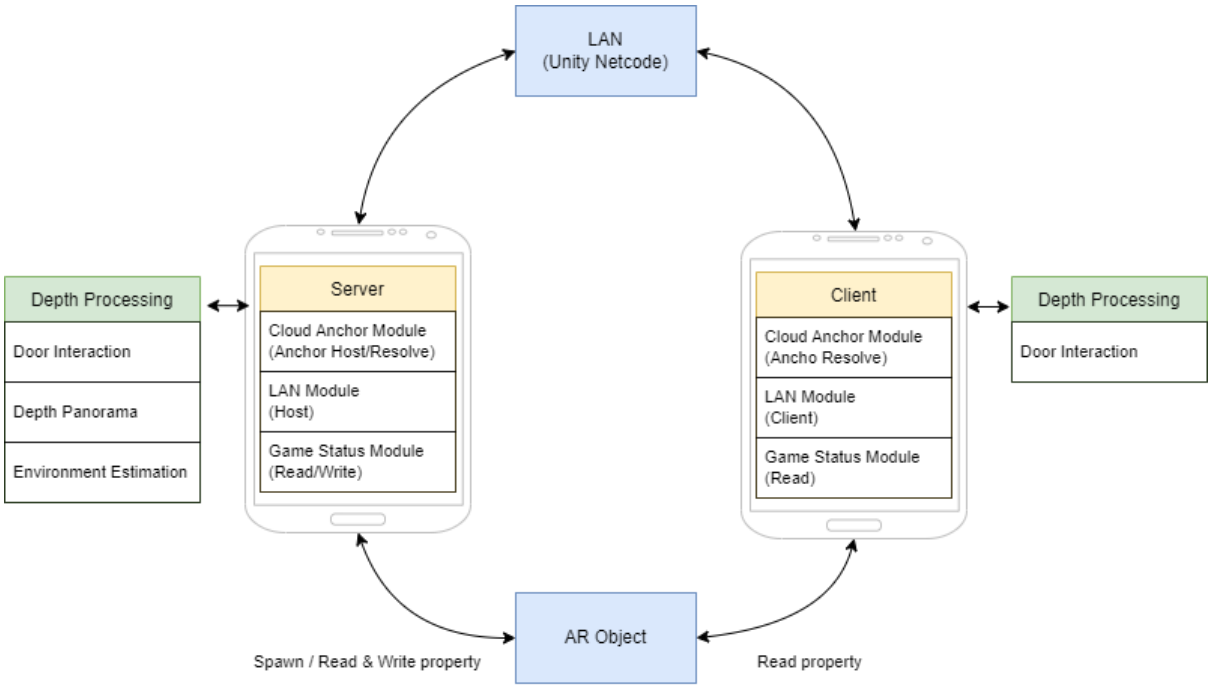
3.3.4. Cloud Anchor 와 통신에 대한 요구사항

- 로컬 멀티플레이를 위한 LAN 연결이 필요하다. 동일한 Wi-Fi 혹은 데이터 테더링을 통해 게임에 접속할 수 있도록 구현한다.
- 별도의 서버를 두지 않고 Host-Client 방식으로 멀티플레이를 구현한다. Host device 에서 앵커를 클라우드에 호스팅하면 Client device 에서 앵커를 resolve 하여 좌표계 동기화를 수행한다.

- Cloud Anchor 를 기준으로 한 device 의 위치를 LAN 을 통해 전달하여 다른 플레이어의 위치를 트래킹한다. 모든 통신은 Cloud Anchor 를 원점으로 한 상대 좌표계를 기준으로 전달하여 각자의 디바이스에서 역계산을 수행하는 것으로 이루어진다.

3.4. 시스템 설계

3.4.1. 시스템 구성도



[그림 3] 시스템 구성도

시스템 구성도는 **[그림 3]**과 같다. 게임 내 통신은 클라이언트-서버 모델로 이루어진다. 서버 플레이어는 서버와 클라이언트를 동시에 수행하는 호스트(Host)의 역할로 게임에 참여한다. 각 디바이스에는 Cloud Anchor 를 관리하는 모듈, LAN 통신을 관리하는 모듈, 게임 상태를 관리하는 모듈이 존재한다. 서버 여부에 따라 모듈을 통해 수행할 수 있는 작업의 권한이 다르며 그 목록은 다음과 같다.

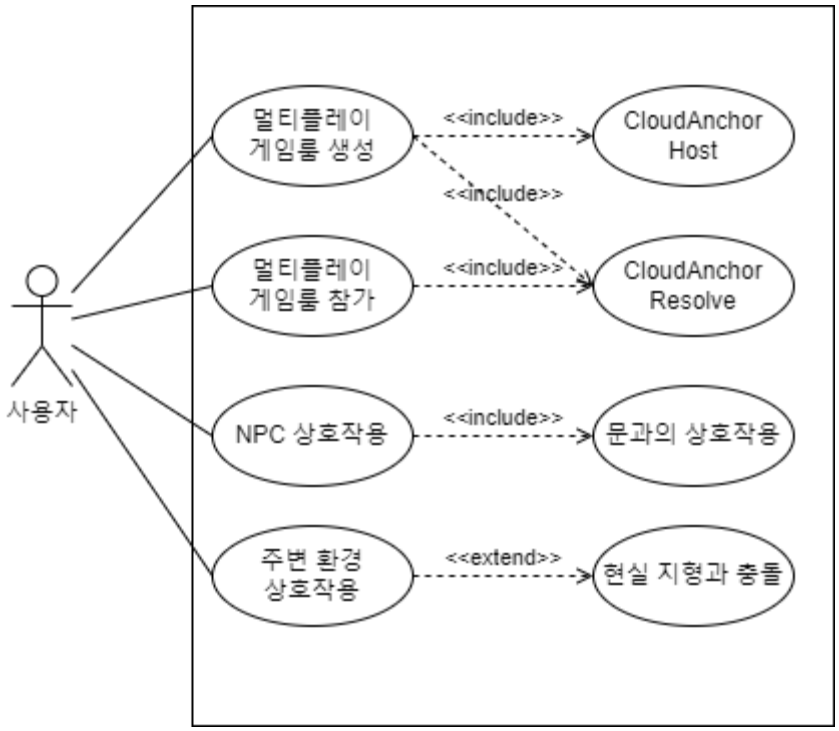
1. 서버에서만 Cloud Anchor 를 Host 할 수 있다.
2. 서버에서만 게임 상태 및 오브젝트 속성을 편집할 수 있다.
3. 서버에서만 AR Object 를 Spawn 할 수 있다.

현실과의 상호작용을 위한 Depth Processing 은 각 디바이스의 로컬 환경에서 개별로 수행한다. 문을 여는 상호작용은 서버와 클라이언트에서 각자 판단하며 상호작용 여부만 LAN 을 통해

전송한다. 주변 환경 추정을 위한 Depth Processing 은 서버 디바이스에서만 수행하고 결과를 클라이언트에 동기화한다.

3.4.2. UML 다이어그램을 통한 시스템 모델링

3.4.2.1. Use Case Diagram

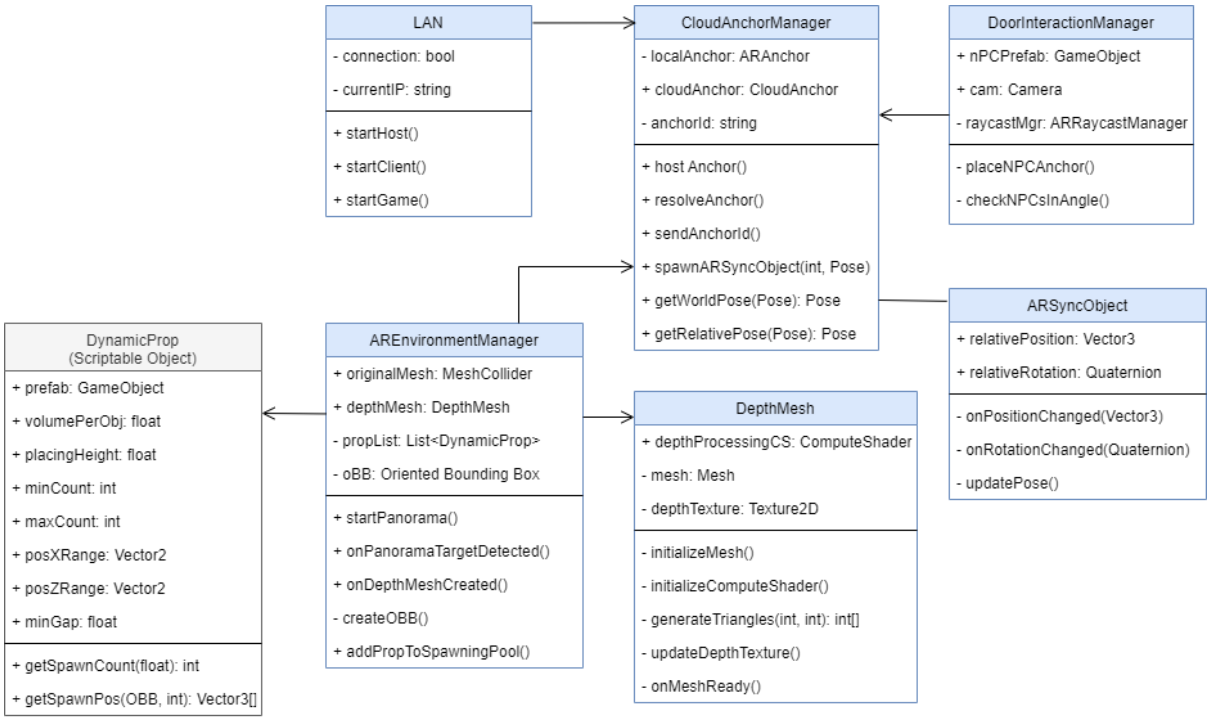


[그림 4] Use Case Diagram

[그림 4]는 게임의 Use Case 를 나타낸다. 사용자는 게임을 시작하고 멀티플레이를 위한 게임룸을 생성하거나 이미 존재하는 게임룸에 참가할 수 있다. 게임룸을 생성하는, 즉 서버역할을 하는 Use Case 에서는 클라우드 앵커를 생성하고 Host 하는 행동이 수반된다. 게임룸에 참가하는 클라이언트 사용자는 Host 된 클라우드 앵커를 받아와 Resolve 하는 행동이 수반된다.

게임 플레이적인 Use Case 로는 크게 NPC 상호작용과 주변 환경 상호작용이 있다. NPC 상호작용은 문을 열었을 때 이루어지므로 문과의 상호작용이 Use Case 에 포함된다. 주변 환경과의 상호작용은 여러 방법으로 이루어질 수 있지만, 대표적으로 현실 지형과의 충돌 기능으로 확장된다.

3.4.2.2. Class Diagram



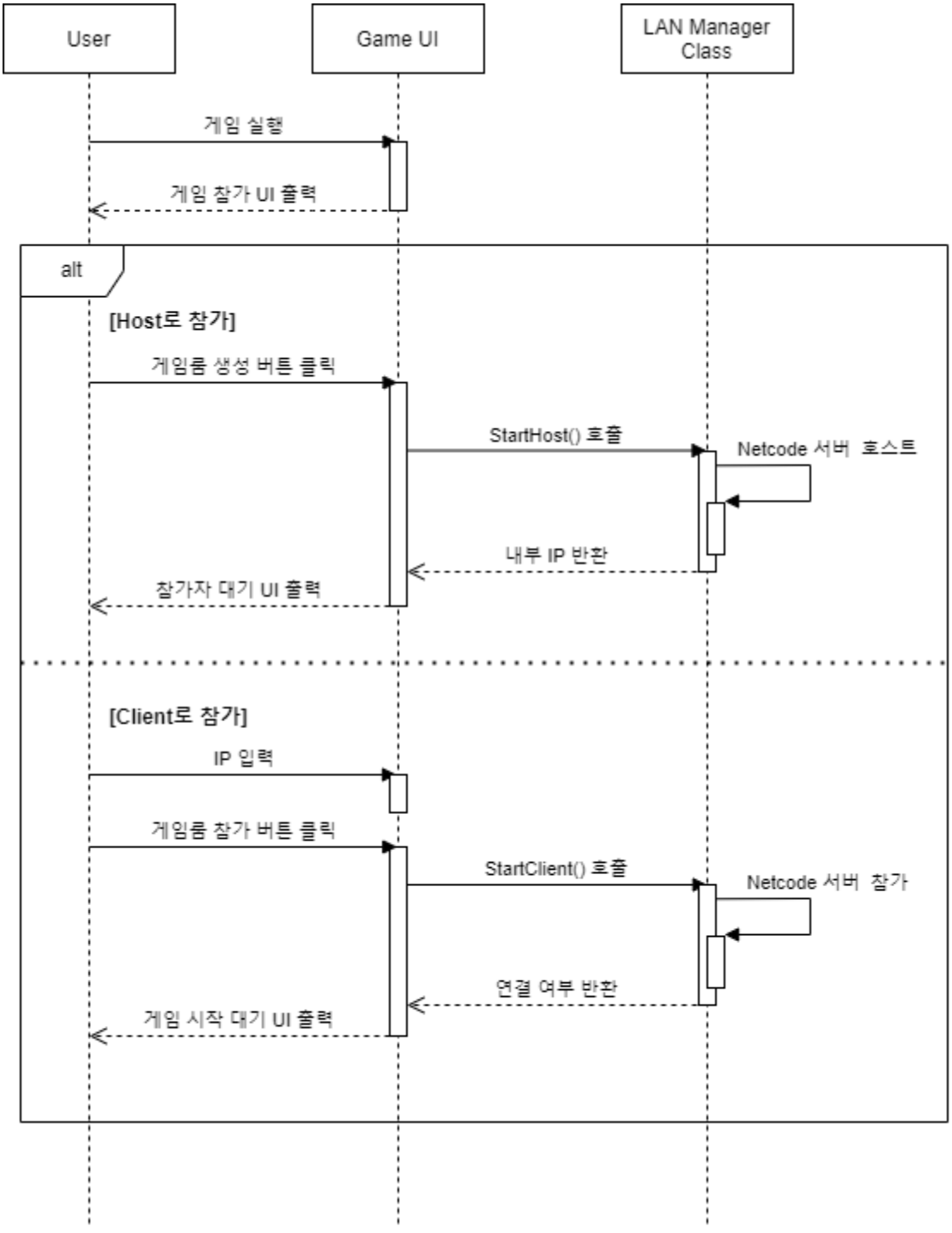
[그림 5] Class Diagram

[그림 5]는 게임의 주요 시스템을 담당하는 Class 사이의 관계를 나타낸 Diagram 이다. 먼저 LAN 과 CloudAnchorManager 는 멀티플레이와 데이터 동기화를 담당하는 Class 이다. LAN 은 동일한 Wi-Fi 상에서 데이터를 주고받을 수 있도록 디바이스를 연결해주는 기능을 담당하고, CloudAnchorManager 에서 이 네트워크를 통해 Anchor ID 를 전달하여 Host-Resolve 과정을 수행한다. 이때 생성된 CloudAnchor 를 기준으로 ARSyncObject 가 오브젝트의 위치를 동기화 한다.

DoorInteractionManager 는 NPC 와의 상호작용을 담당하는 클래스이다. 미리 지정한 NPC 앵커까지의 거리와 깊이값을 비교하여 문의 상태를 지속적으로 확인하여 상호작용 이벤트를 발생시킨다.

AREnvironmentManager 는 동적 게임오브젝트 생성을 위한 클래스이다. 주변 환경 인식을 위해 주변 360 도의 DepthMesh 파노라마를 생성하여 방의 크기를 추정하고 이를 바탕으로 생성할 오브젝트의 개수와 위치를 선정한다. 이때 DynamicProp 은 클래스는 아니지만 각 게임 내 아이템의 동적 생성 정보를 담기 위한 Data Container 로써의 역할을 수행한다.

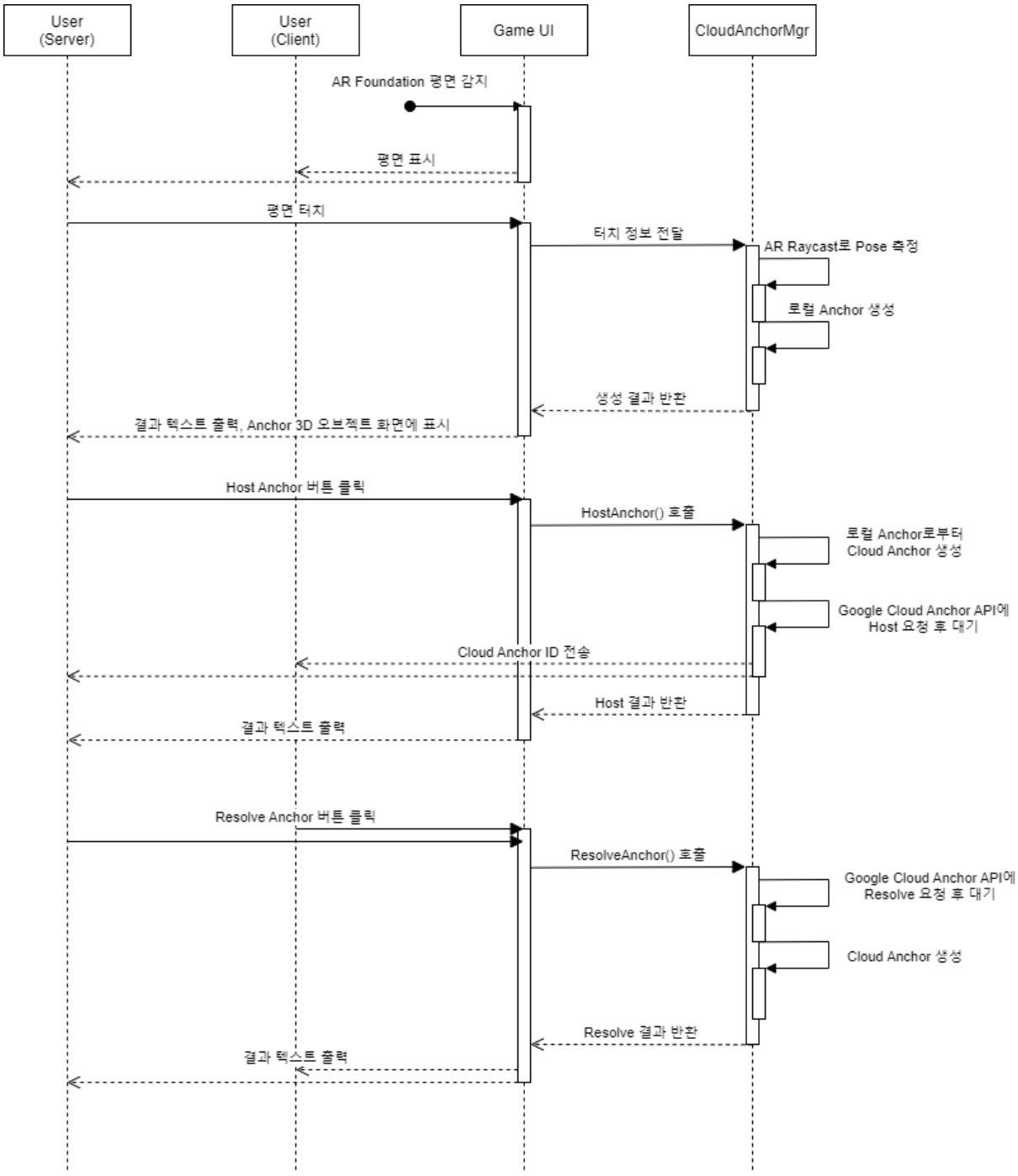
3.4.2.2. Sequence Diagram



[그림 6] 게임 참가 Sequence Diagram

Host 로 게임에 참가하는 경우 게임룸 생성 버튼을 터치하여 LAN 서버를 생성한다. LAN 서버 초기화에는 Unity Netcode 가 사용되고, 성공적으로 생성될 경우 내부 IP 가 반환된다. 반환된

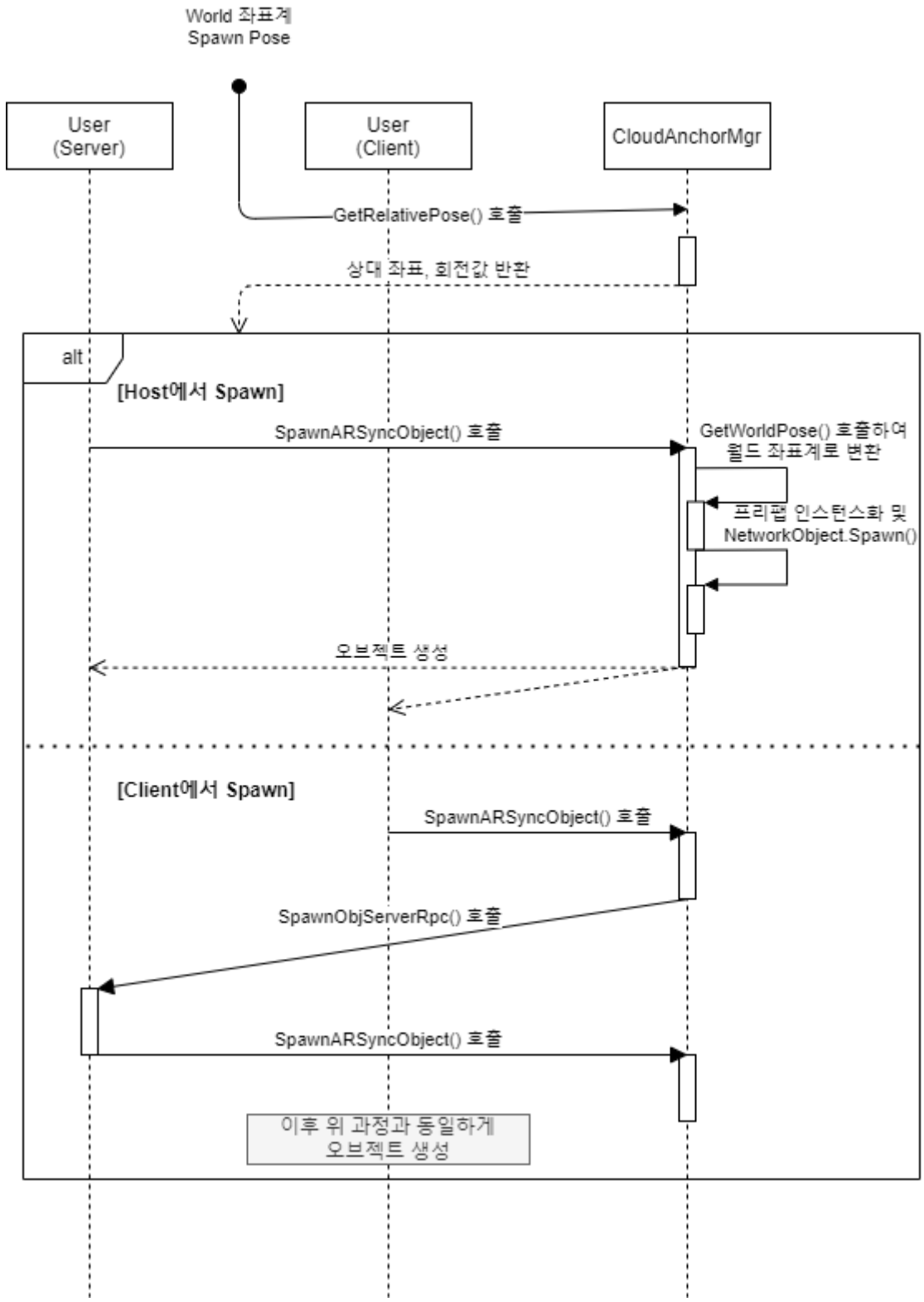
IP 는 UI 에 출력되고 이 IP 를 공유하여 다른 Client 가 게임에 참가할 수 있다. Client 는 전달받은 IP 를 UI 에 입력하여 Netcode 서버에 참가할 수 있다.



[그림 7] Cloud Anchor Host & Resolve Sequence Diagram

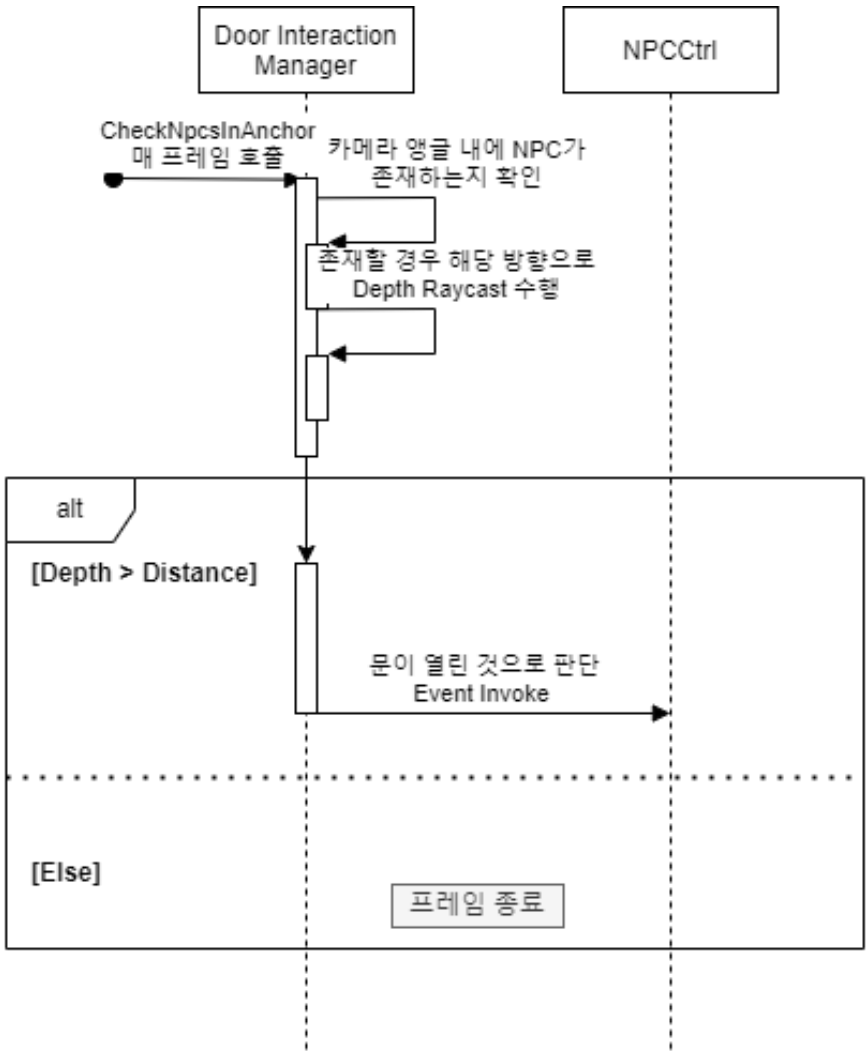
위치 동기화를 위한 Cloud Anchor 의 Host 와 Resolve 과정이 **[그림 7]**의 다이어그램에 나타나 있다. Host 가 앵커를 생성할 위치를 Raycast 로 지정하면 해당 위치에 로컬 앵커가 생성된다. Host Anchor 버튼을 누르면 Google Cloud Anchor API 를 통해 클라우드로 앵커가 호스팅되며,

성공적으로 수행될 경우 앵커 ID 가 반환된다. 이 Anchor ID 는 LAN 을 통해 모든 Client 에게 전송되고, Resolve Anchor 버튼을 눌러 모두가 동일한 위치에 앵커를 생성할 수 있게 된다.



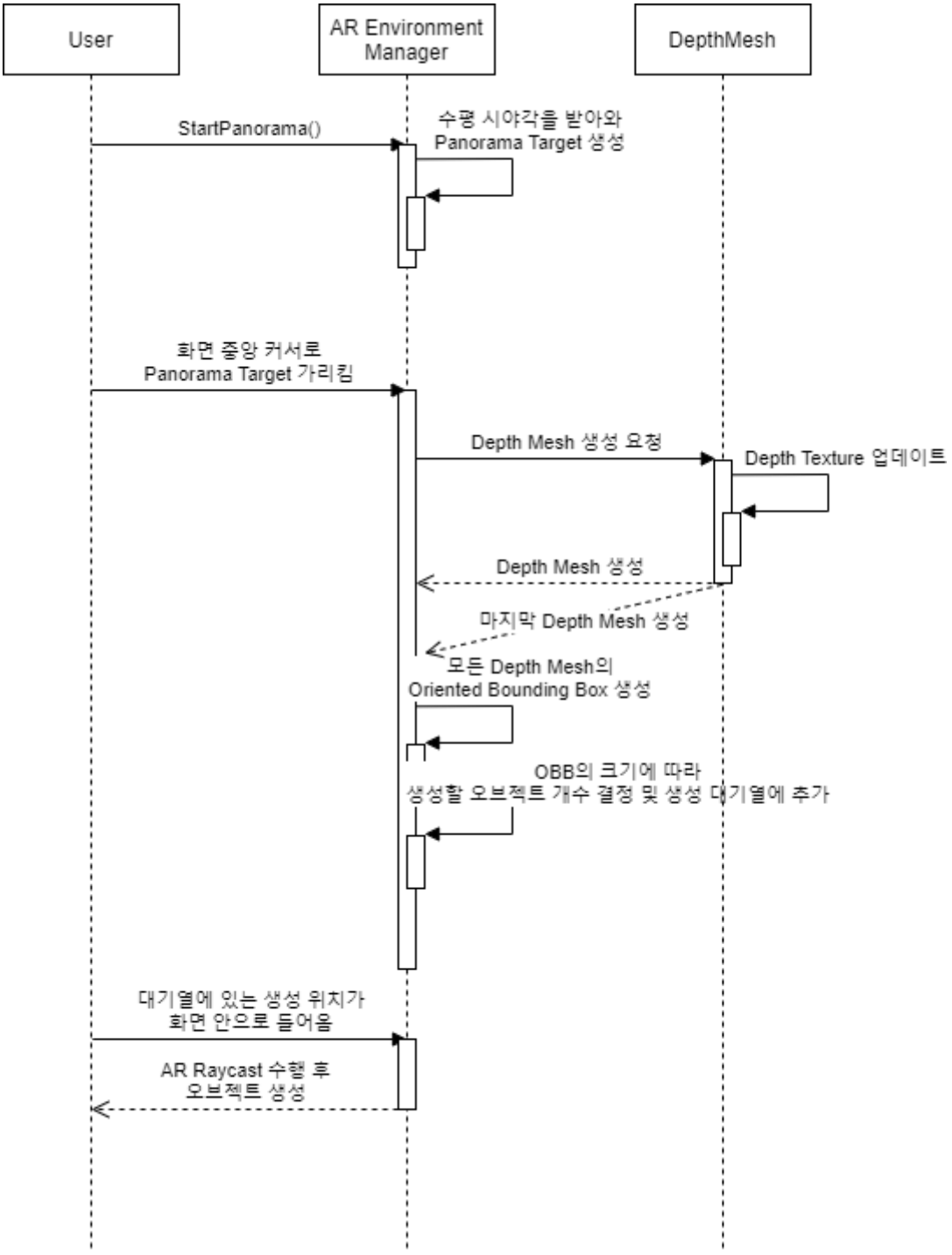
[그림 8] Spawn AR Sync Object Sequence Diagram

증강현실 환경에서는 각 클라이언트마다 월드 좌표계가 다르기 때문에 오브젝트를 다룰 때 클라우드 앵커를 기준으로한 상대 좌표계로 정보를 주고받는다. 서버(호스트)에서 오브젝트를 spawn 할 때는 직접 프리팹을 인스턴스화 하여 로컬에 먼저 생성한 뒤 NetworkObject.Spawn()을 호출하여 클라이언트에 생성명령을 내린다. 클라이언트에서 오브젝트를 spawn 할 때는 서버에 RPC(Remote Procedure Call)를 보내 서버에서 spawn 할 때와 동일한 과정을 수행한다.



[그림 9] 문과의 상호작용 Sequence Diagram

NPC 와의 상호작용을 위해 문의 상태를 지속적으로 확인해야 한다. NPC 앵커의 위치가 카메라 앵글 내에 있는지 매 프레임 확인하며 앵글 안에 들어올 경우 거리 비교를 위해 Depth Raycast 를 수행한다. 만약 문이 열려 있어 Depth 값이 NPC 까지의 거리보다 클 경우 문이 열린 것으로 판단하고 Event 를 Invoke 한다. 아닐 경우 프레임을 종료하고 다음 호출을 기다린다.



[그림 10] 주변 환경 추정 및 동적 오브젝트 생성 Sequence Diagram

동적 환경 생성을 위해서는 우선 주변 환경을 게임 내 3D 환경으로 추정해야 한다. 이를 위해 본 프로젝트에서는 플레이어 주변 360 도의 Depth Mesh 를 생성하는 Panorama 동작을 수행한다. Panorama 를 시작하면 수평 시야각을 받아와 360 도를 전부 커버할 수 있는 수의 Target 오브젝트를 생성한다. 생성된 Target 이 화면 중앙에 포착되면 Depth Mesh 생성 요청을 보내고 Depth Texture 로부터 Mesh 를 생성한다. 모든 Target 이 소모되고 마지막 Mesh 가 생성되면 모든

Mesh 를 둘러싸는 가장 작은 크기의 Oriented Bounding Box 를 생성한다. 이 OBB 의 크기, 부피 정보로부터 Dynamic Prop 의 조건에 맞게 동적으로 게임 오브젝트를 생성한다.

3.5. 구현

GitHub: https://github.com/jeonse3875/ARSurvival

구현된 프로젝트 파일은 위 GitHub 링크에 업로드 되어 있다. 주요 파일들의 경로는 다음과 같다. Depth Mesh 생성 코드는 Google Depth Lab 오픈 소스를 활용하였다. [6]

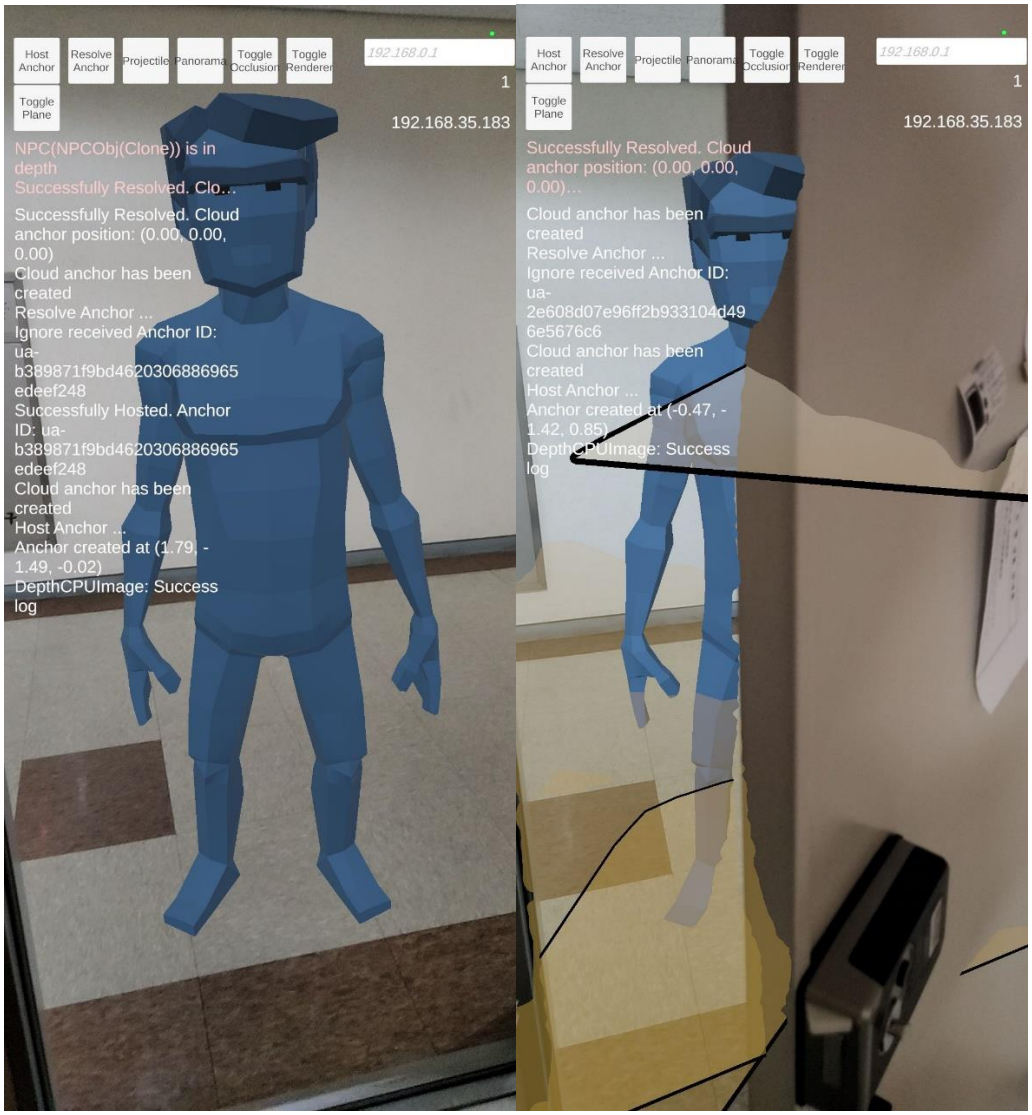
소스코드 경로	역할
Assets/Scripts/AREnvironmentMgr.cs	주변 환경 추정 및 동적 오브젝트 생성
Assets/Scripts/DoorInteractionMgr.cs	문과의 상호작용 이벤트 제어
Assets/Scripts/DynamicPropSO.cs	동적으로 생성될 오브젝트의 데이터 컨테이너
Assets/Scripts/LightEstimation.cs	AR 환경에서의 광원 추정
Assets/Scripts/TrackingMgr.cs	Occlusion 보완을 위한 이미지 트래킹 기능
Assets/Scripts/Multiplayer/ARSyncObject.cs	AR 멀티플레이 환경에서 오브젝트 동기화
Assets/Scripts/Multiplayer/AvatarCtrl.cs	플레이어 아바타 제어
Assets/Scripts/Multiplayer/CloudAnchorMgr.cs	Cloud Anchor 제어 및 동기화 관련 유틸리티
Assets/Scripts/Multiplayer/LANMgr.cs	LAN 멀티플레이 관리
Assets/Scripts/Depth/DepthMeshCollider.cs	DepthTexture 로부터 Mesh 및 Collider 생성

Scene 경로	역할
Assets/Scenes/MainScene.unity	전체 게임 씬

시스템 설계에서 계획된 내용들은 모두 위 소스코드 상에서 구현되었으며, 자세한 내용은 GitHub 에서 확인할 수 있다.

4. 프로젝트 결과

4.1. 문과의 상호작용 기능



[그림 11] NPC 가 문에 가려지지 않은 상태(좌)와 가려진 상태(우)

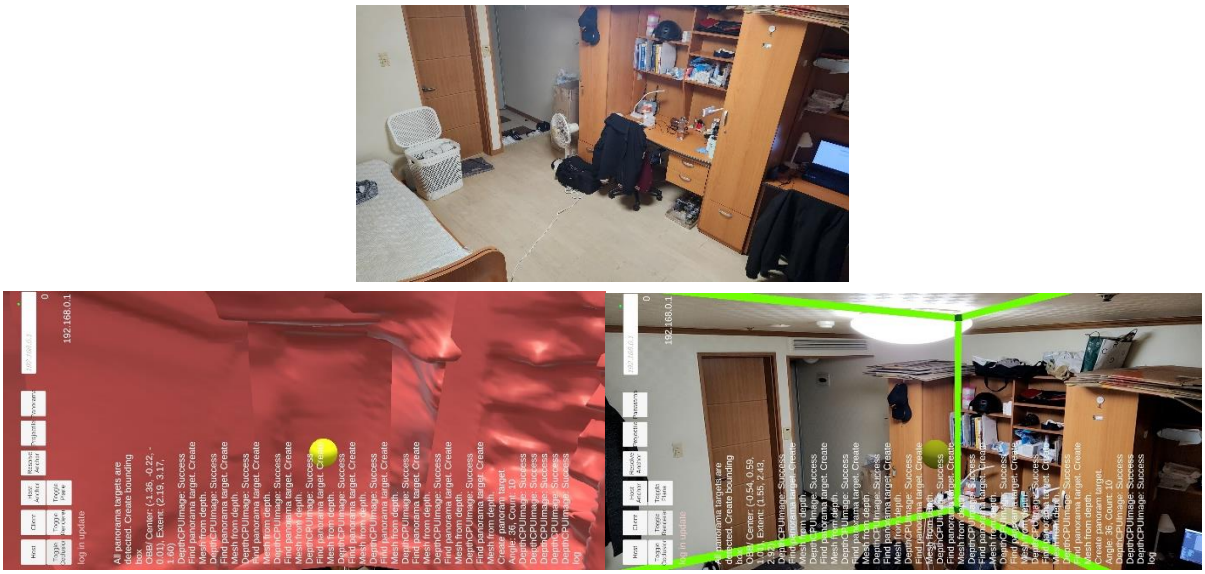
본 프로젝트에서 개발한 게임은 문을 여는 것으로 NPC 와의 상호작용한다. 문을 여는 행동을 감지하기 위해 게임 시작 전 미리 문 앞에 NPC 가 생성될 앵커를 지정한다. NPC 가 찾아오는 게임 내 이벤트가 발생하면 이 앵커 위치에 NPC 가 스폰된다.

NPC 가 스폰되면 매 프레임마다 NPC 의 중심이 카메라 앵글 안에 들어오는지 확인한다. NPC 가 앵글 안에 들어왔을 경우 해당 방향으로 Depth Raycast 를 수행하여 NPC 방향으로의 Depth 값을 얻는다. 이 Depth 값과 디바이스와 NPC 사이의 거리를 비교하여 문이 열려있는지 여부를 판단한다. Depth 값이 Distance 보다 크면 NPC 가 가려지지 않은 상태를 의미하므로 문이 열린 것으로 판단하고, 반대의 상황에서는 닫힌 것으로 판단한다.

[그림 11]의 좌측 이미지를 보면 좌상단의 'NPC(NPCObj(Clone)) is in depth'라는 로그를 통해 해당 기능이 정상적으로 작동하는 것을 확인할 수 있다. 반대로 우측의 이미지처럼 NPC 의 중심이 가려진 상황에서는 NPC 와의 상호작용 이벤트가 Invoke 되지 않는다.

ToF 센서를 이용하지 않고 단순 Motion Stereo 만을 이용해 Depth 값을 받아오는 기기의 경우에는 문이 열고 닫힐 때 Depth Texture 의 업데이트가 느리기 때문에 NPC 가 제대로 가려지지 않는 문제가 있었다. 이 문제를 보완하기 위해 이미지 트래킹으로 NPC 를 가릴 수 있는 오브젝트를 문 위치에 생성하는 방법을 사용했다. 문에 미리 지정된 이미지를 출력하여 부착하면 ToF 센서가 없더라도 정상적으로 게임을 플레이할 수 있다.

4.2. 주변 환경 인식 및 동적 오브젝트 생성 기능



[그림 12] 방의 모습(상), DepthMesh 파노라마(좌), Bounding Box(우)

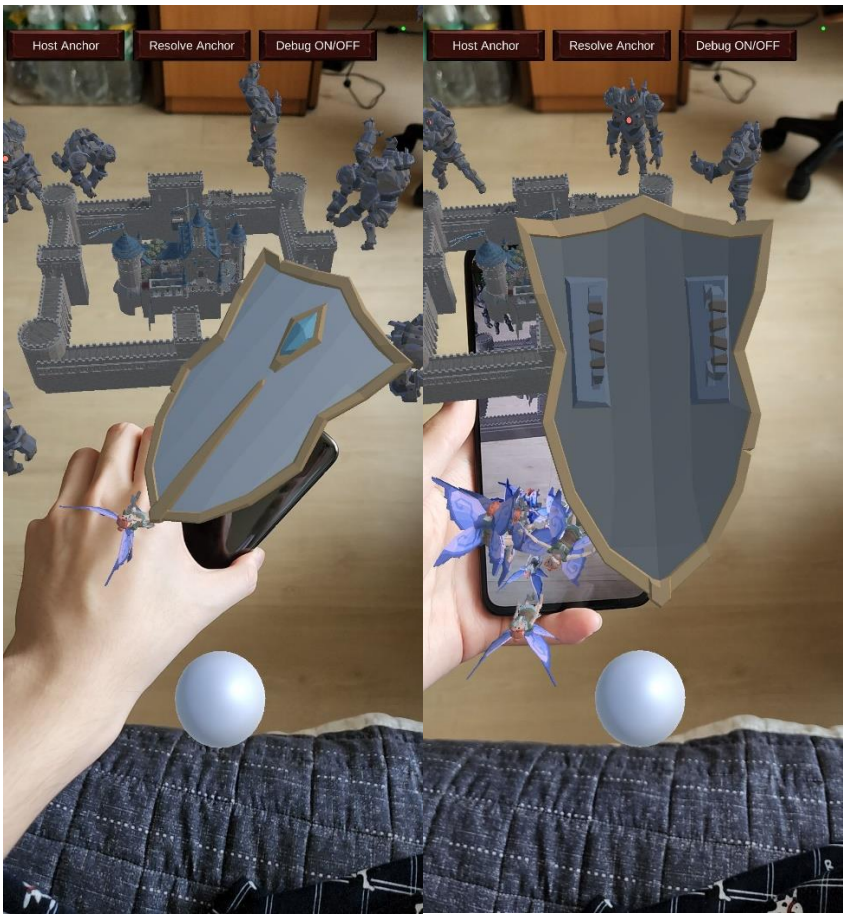
장소에 따른 다양한 경험을 제공하기 위해 본 프로젝트에서는 주변 환경 추정 및 동적 오브젝트 생성 기능을 개발하였다. 환경 추정은 기본적으로 Depth Mesh 를 통해 게임 내에 실제 환경을 재구성하는 것으로 이루어지며, 플레이어 주변의 360 도 전체를 커버링하기 위해 Depth Mesh 파노라마를 생성한다.

[그림 12]의 상단 사진은 게임을 플레이한 실제 방의 모습을 보여준다. 이 장소에서 Depth Mesh 파노라마를 생성할 경우 좌측 사진처럼 주변 환경의 굴곡에 맞게 Mesh 로 둘러싸인다. 파노라마는 기기의 수평 시야각을 받아와 360 도를 커버할 수 있게끔 일정한 간격으로 Depth Mesh 를 생성하는 것으로 이루어진다.

모든 Depth Mesh 가 생성되면 방의 크기와 모양을 추정하기 위해 모든 Mesh 를 포함하는 Bounding Box 를 생성한다. 최대한 정확한 크기의 추정을 위해 Oriented Bounding Box 알고리즘을 사용하며 실제 생성된 OBB 를 시각화해보면 **[그림 12]**의 우측 사진과 같다. 실제 방의 윤곽과 비슷한 형태의 Bounding Box 가 생성된 것을 확인할 수 있으며, 대부분의 방이 직육면체의 형태를 띠는 것을 고려하면 보편적인 Use Case 에서의 활용이 가능할 것으로 예상된다.

성능 평가를 위해 실제 방의 부피와 추정한 크기를 비교해보면. **[그림 12]**의 우측 사진에서 추정한 OBB 의 extent 는 (1.55, 2.43, 2.92)이다. 1 Unity 단위가 1m 와 같으므로 부피로 환산해보면 약 87.98m^3이다. 실제 게임을 플레이한 방의 크기는 약 6 평 정도로, 면적으로는 19.83 m^2, 부피로는 약 88.30m^3이다. 보고서에는 한 장의 사진만 포함되어 있지만, 여러 번의 테스트로 일정한 부피를 얻어내는 것을 확인하였고, 직육면체 형태의 방의 경우 실제와 유사하게 추정할 수 있음을 알 수 있었다.

4.3. AR 오브젝트 동기화 기능



[그림 13] Cloud Anchor 기반 멀티플레이어 시스템으로 다른 플레이어를 트래킹

증강현실에서 오브젝트의 위치를 실제 위치에 동기화시키기 위해 Cloud Anchor 를 활용하여 멀티플레이어 시스템을 개발하였다. Host 역할을 하는 기기가 앵커를 호스팅하면 LAN 을 통해 전달받은 ID 로 Resolve 를 수행하여 동기화한다. 이후 생성되는 오브젝트들은 Cloud Anchor 의 자식 오브젝트로 추가되며, Cloud Anchor 가 계속 동기화되는 한 실제 위치가 고정된다.

이 기능을 활용하면 게임 내 오브젝트의 위치 동기화뿐만 아니라 다른 플레이어의 위치도 트래킹할 수 있다. **[그림 13]**에서 보이는 것처럼 다른 스마트폰의 이동과 회전을 정교하게 추적할 수 있다. 플레이어가 들고 있는 물체의 정보를 공유할 수 있고, 행동 맥락을 파악할 수 있다는 점에서 활용도가 높을 것으로 예상한다.

5. 결론 및 기대효과

새로운 상호작용 방식을 통해 현실과 가상의 경계를 허물 수 있다. 사용자가 일상적으로 하는 행동들이 게임 속에 영향을 미칠 수 있다는 점에서 괴리감을 줄일 수 있고, 이는 몰입감을 높이는 효과를 가져온다. 기존 GPS 기반 증강현실과는 다른 종류의 몰입감을 줌으로써 AR 장르의 인식 개선도 기대해볼 수 있다.

동적 오브젝트 생성을 통해 장소마다 색다른 경험을 얻을 수 있다. 여러 장소에서 움직이며 플레이할 수 있는 증강현실 게임의 장점을 살려 장소에 따라 다르게 증강된 모습을 즐길 수 있고, 이는 한 게임을 여러 번 즐기게 하는 명분이 된다. 다회차 플레이를 통해 게임에 더 몰입하게 만들 뿐만 아니라 증강현실 게임의 유저층을 늘릴 수 있는 기회로 작용할 수 있다.

여럿이 함께 즐기는 게임의 장르로 증강현실이 자리매김할 수 있다. 같은 장소에서 각자의 디바이스를 정교하게 트래킹할 수 있기 때문에 활용도가 높으며, 여러 명의 플레이어가 동시에 즐길 때 재미가 극대화될 여지가 있다. 일반적인 모바일 게임보다 활동적이고 가상현실 게임보다 진입장벽이 낮기 때문에 여럿이 즐길 수 있는 게임 장르 중 하나로 발전할 가능성이 높다.

6. 참고문헌

[1] 스마트폰 교체 주기 통계: https://www.strategyanalytics.com/

[2] ARCore: https://developers.google.com/ar/develop

[3] ARKit: https://developer.apple.com/kr/augmented-reality/arkit/

[4] ToF 이미지: https://www.stemmer-imaging.com/en/knowledge-base/cameras-3d-time-of-flight-cameras/

[5] Niantic, AR Voyage(Google Play)

[6] ARCore Depth Lab: https://github.com/googlesamples/arcore-depth-lab