Pusan National University
Computer Science and Engineering
Technical Report 2023-10

2023 전기 졸업과제 최종보고서

NeRF 를 이용한 중고시장 플랫폼 개발

은승우

김성현

권재섭

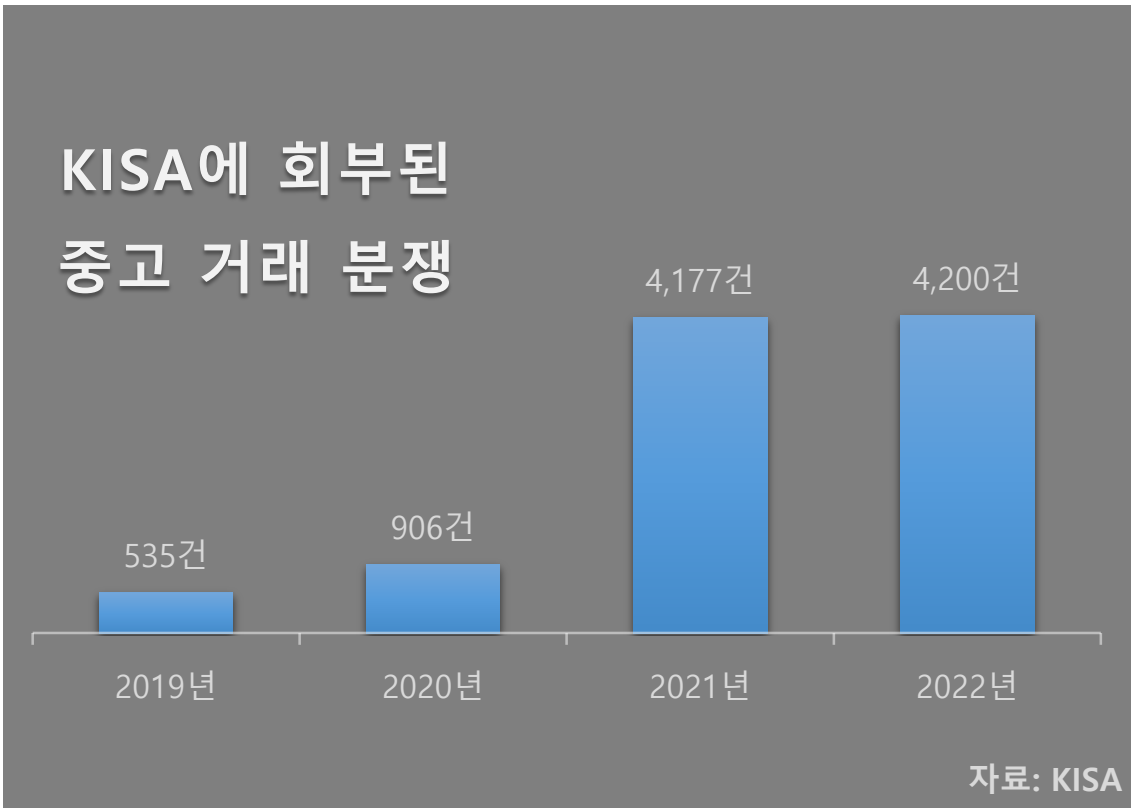
지도교수 이명호

목 차

1. 연구 개요

1.1. 연구 배경

2008년 4조원에 불과했던 중고 거래 시장 규모가 2021년 기준 24조원을 기록하는 등 중고 거래 시장이 급속도로 성장하고 있는 가운데 거래 당사자간 분쟁도 급격하게 증가하고 있다. KISA(한국인터넷진흥원)에 회부된 중고 거래 분쟁 건수는 2019년 535건에서 2020년 906건, 2021년 4177건, 2022년 4200건까지 늘어났다(그림1 참고).



<그림 1 : KISA에 회부된 중고 거래 분쟁>

1.2. 기존 문제점 및 요구 조건

거래 전 상품에 대한 정보 제공이 비대면으로만 이루어지다 보니 판매자와 구매자 간에 정보에 대한 비대칭성이 존재하게 되는데, 이 정보의 비대칭성이 분쟁의 주요 원인으로 추측된다. 기존의 2d 이미지를 3d화해서 제공한다면 2d 이미지만을 제공할 때보다 많은 정보를 구매자가 획득할 수 있게 돼서 정보의 비대칭성을 어느정도 극복할 수 있을 것이고, 그렇게 되면 자연스럽게 분쟁을 줄일 수 있을 것이다.

1.3. 연구 목표

차세대 기술로 주목받고 있는 NeRF(Neural Radiance Fields for View Synthesis)를 이용하여 사용자가 촬영한 상품의 2D 이미지들을 NeRF기술을 사용해 3d로 변환해서 제공하는 중고 거래 플랫폼을 만드는 것을 목표한다. 이로 인해 중고거래 중 불확실한 제품을 사진으로만 판단하는 것 보다, AR을 통해 실제로 제품을 확인할 수 있도록 하 여 제품을 설계하도록 한다. 목표를 위하여 밑과 같이 크게 2가지의 목표를 잡았다.

- 안드로이드와 Unity의 연동 AR 중고 플랫폼 개발
- NeRF를 이용하여 개인이 이용 가능한 3D Reconstruction 기술 개발

2. 연구 구성 및 배경환경

2.1. 시스템 구성

2.1.1. 전체 구성도

<그림 2 : 연구 전체 구성도>

전체구성도는 위(그림 2)와 같이 안드로이드, 유니티 AR, 서버로 3가지로 나누기로 결정하였다. 각각 파트 별 목표는 다음과 같다.

- Android 파트
  - 플랫폼 역할을 가능하도록 하는 기본적이 UI 구성.
  - 서버와 통신하여, 새로운 글을 추가하고, 전체 목록을 화면에 표시.
  - 유니티와 연동하여, 화면에 표시할 제품의 ID 제공.

- Server 파트
  - 별도 조작없이 데이터 전처리, NeRF 모델 학습, mesh 추출까지 자동으로 실행되게 하는 것
  - Android로부터 글 등록 요청이 오면, 글 정보를 DB에 저장하고 첨부된 동영상을 입력으로 nerfstudio를 실행시키는 것
  - Android로부터 글 목록 및 글 정보에 대한 요청이 오면, DB에 저장된 해당 정보를 조회해서 전송하는 것
  - Android로부터 mesh파일 요청이 오면, 서버에 저장된 mesh파일을 전송하는 것
- Unity 파트
  - Obj, mtl, png 파일에 대한 요청을 하면, 그것을 서버에서 받아서 하나의 obj로 렌더링 하는 것
  - 스마트폰 화면의 터치로 원하는 곳에 렌더링 한 obj를 AR상에서 현실감 있게 생성하는 것
  - 생성한 obj를 회전, 스케일링을 지원하는 것
  - Obj에 대하여 Occlusion을 적용시키는 것

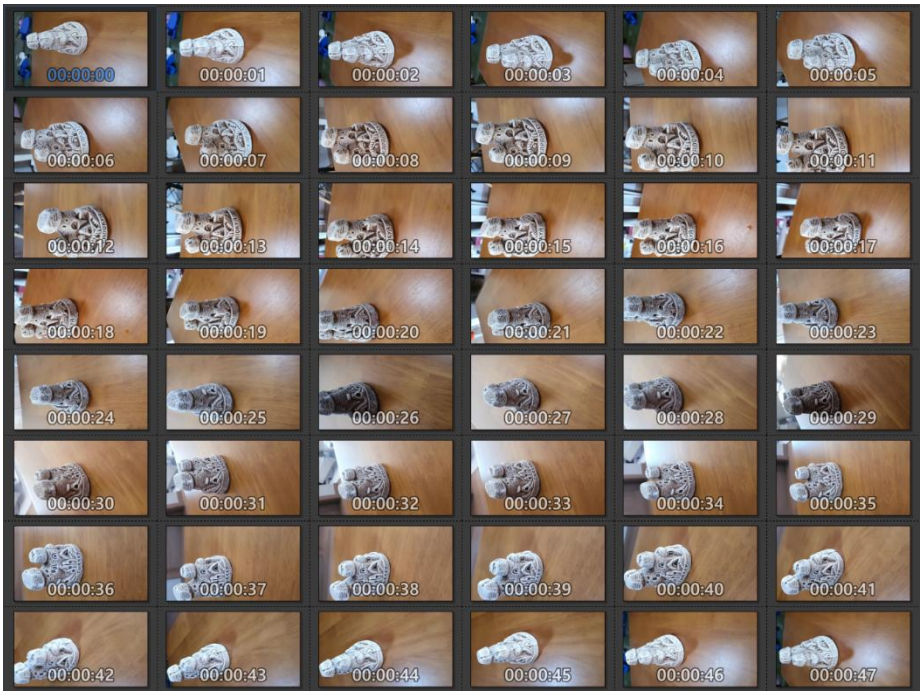
2.1.2. 단계별 구성도

2.1.2.1. 판매자 기준 구성도

판매자를 기준으로 작성한 구성도는 아래 그림 3 과 같다.

<그림 3 : 판매자의 기준으로 생성된 구성도>

판매자는 판매를 원하는 그림 4와 같이 물품에게 1분정도 삥 둘러서 촬영한다. 그 이후, 물체의 크기, 가격, 그리고 간단한 설명을 작성하여 제출한 다음, 입력한 데이터가 서버에 전달된다. 이때, 전달된 동영상은 서버에서 자동으로 NeRF Studio에게 전달해 NeRF를 이용해 Mesh가 생성되며, 그것이 서버에 obj, mtl, png형태로 저장이 된다.



<그림 4 : 물체를 1분간 촬영하는 과정>

2.1.2.2. 구매자 기준 구성도

구매자를 기준으로 작성한 구성도는 아래 그림 5 와 같다.

<그림 5 : 구매자의 기준으로 생성된 구성도 >

서버는 어플로부터 게시글 목록, 게시글 정보 등을 요청 받으면. 해당하는 정보를 database에서 조회해 어플에게 전송한다. 만약 구매자가 제품에 대하여 구체적으로 살펴보고 싶으면, 상세 페이지로 넘어가며, 만약 제품이 어떤 지 AR상으로 보고 싶으면 체험하기 버튼을 누르면 어플이 자동으로 Unity AR 어플로 이동한다. 이때, Unity AR에서 서버에게 통신하여 원하는 제품을 오브젝트 형태로 만들어내며, AR상에서 물체를 이동, 회전, 그리고 스케일링이 가능하다. 또한, 물체에 Occlusion을 적용시켜 물체가 일부만 보일때의 모습도 확인이 가능하다.

2.2. 개발 환경

2.2.1. 개발언어 및 도구

서버 개발 언어로는 Java를, 서버 개발 framework로는 Spring boot를 선택했다.

또한, DBMS로는 MariaDB를 사용하였고, NeRF를 손쉽게 사용할 있는 API인 Nerfstudio를 사용하였다.

안드로이드의 개발도구는 밑과 같다.

- 안드로이드 스튜디오 2022.2.1
- 서버 통신 라이브러리 : Volley 1.2.1
- NDK 25.2.9519653
- JDK 1.8
- 데이터 처리 라이브러리 : gson 2.10.1
- 이미지 로더 라이브러리 : glide 4.15.1
- 개발언어 : Kotlin

AR 부분은 Unity AR Core를 C#을 이용하여 개발하였다.

2.2.2. 실행환경

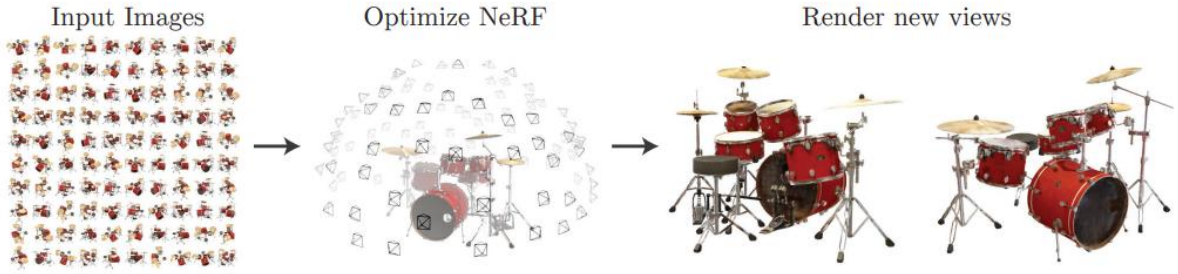
Spring boot, Nerfstudio, MariaDB 모두 docker container환경에서 동작하는데, spring boot와 Nerfstudio는 동일한 container에서 동작되고, MariaDB는 별도의 container에서 동작한다. 두 container간 통신은 network container를 통해 이루어진다.

어플은 안드로이드 기준으로 실행되게 설계되었으며, 안드로이드 어플의 Embedding을 통해 Unity AR 어플로 접속하여 동작한다.

2.3. 배경 설명

2.3.1. NeRF 설명

NeRF는 view direction(카메라 시점)과 해당 시점에서 촬영한 이미지 쌍들을 사용해 학습하고, 이후 새로운 view direction이 주어졌을 때 해당 view direction에서 촬영한 장면을 계산해내는 기술이다(그림6 참조).



<그림 6 : NeRF의 기본적 구성>

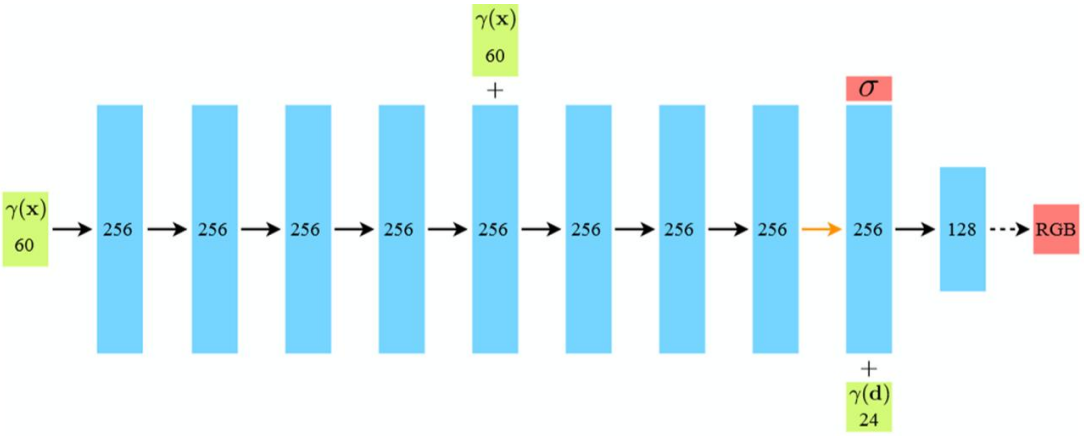
NeRF의 학습 과정을 살펴보면 각 view direction, 이미지 쌍에 대해서 이미지의 각 pixel 좌표마다 pixel 좌표에서 view direction으로 ray를 쏘고, ray위의 N개의 voxel 좌표를 sampling한다. 그리고 sampling된 각 voxel 좌표에 대해서 voxel 좌표(x,y,z)와 view direction(θ,ϕ)을 MLP의 입력으로 사용해서 voxel 좌표의 radiance(RGB)와 volume density(σ)를 계산한다.

이때 성능향상을 위해 position encoding을 사용해 MLP의 입력 차원수를 늘려준다. position encoding 수식은 아래 수식 1와 같다.

$$\gamma(p) = \left(\sin\left(2^0\pi p\right), \cos\left(2^0\pi p\right), \cdots, \sin\left(2^{L-1}\pi p\right), \cos\left(2^{L-1}\pi p\right)\right)$$

<수식 1 : MLP의 입력 차원수를 위한 position encoding 수식>

이때, 그림 7과 같이 기본 세팅에서는 voxel 좌표에 대해서는 L을 10, view direction에 대해서는 L을 4로 두어서 voxel 좌표는 60차원으로, view direction은 24차원으로 늘려서 입력으로 주었다.



<그림 7 : MLP의 구성도>

이렇게 계산된 각 voxel 좌표의 radiance와 volume density를 사용해 pixel 좌표의 RGBA값을 계산하는데, 수식은 아래 수식 2와 같다.

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i(1 - \exp(-\sigma_i\delta_i))\mathbf{c}_i\,, \text{ where } T_i = \exp\left(-\sum_{j=1}^{i-1}\sigma_j\delta_j\right)$$

<수식 2 : Pixel 좌표의 RGBA값>

위 수식에서 σi, ci는 각각 i번째 voxel의 volume density, radiance, δi는 i번째 voxel과 i+1번째 voxel사이의 거리, Ti는 i번째 voxel이 보일 확률이다. Ti 수식을 살펴보면, 1번째~i-1번째 voxel의 volume density 합을 사용하는데, 합이 클수록 Ti는 작아진다.

이렇게 계산한 pixel 좌표의 RGBA값과 실제 값을 수식 3을 사용해 L2 Loss값을 계산할 수 있다.

$$\mathcal{L} = \sum_{\mathbf{r}\in\mathcal{R}} \left[\left\| \hat{C}_c(\mathbf{r}) - C(\mathbf{r}) \right\|_2^2 + \left\| \hat{C}_f(\mathbf{r}) - C(\mathbf{r}) \right\|_2^2 \right]$$

$\hat{C}_c$ from Coarse Sampling
$\hat{C}_f$ from Fine Sampling
C(r) Ground Truth

<수식 3 : L2 Loss 값>

2.3.1.1. NeRF와 유사한 기술의 비교

2d 이미지를 3d화 하는 기술로 photogrammetry라는 기술도 있다.

photogrammetry는 동일한 물체나 장면에 대해 여러 각도에서 촬영한 이미지를 입력으로 사용하여 이 이미지들에서 common feature들을 식별한 다음 삼각 측량 기법을 사용하여 식별한 feature들의 3d 위치를 결정하는 방식으로 3d 모델을 생성한다.

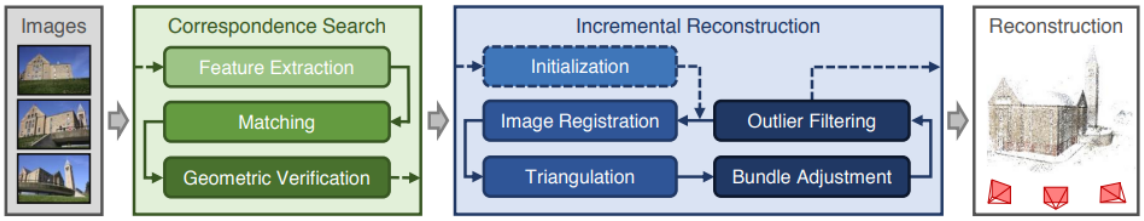
NeRF는 common feature들을 식별하는 방식이 아니다 보니, photogrammetry와 달리 반사를 가진 투명한 물체나 금속 물체를 정확하게 표현할 수 있다. 또한 결과의 품질이 입력 이미지의 품질, 즉 촬영자의 숙련도에 비교적 덜 의존적이다.

전문가가 아닌 일반 사용자가 촬영하고, 대상 재질에 제한이 있으면 안 되는 중고 거래 플랫폼 특성상 NeRF가 photogrammetry보다 더 적합하다.

2.3.2. NeRF Studio 설명

nerfstudio는 Data Preprocess, Data Loader, Model Training, Visualizing, Rendering 등을 API형태로 제공하는 Framework로 Mip-NeRF, instant-NGP, Nerfacto 등 다양한 NeRF model을 지원해서, 여러 model들을 nerfstudio만으로 손쉽게 사용해 볼 수 있다.

NeRF는 입력 값으로 사진과 해당 사진을 찍은 카메라의 위치, 방향 정보를 요구한다. 일반적인 동영상이나 사진의 경우 카메라의 위치, 방향정보가 없기 때문에 COLMAP을 사용한 데이터 전처리과정이 필요한데, COLMAP도 nerfstudio가 지원하는 기능 중 하나이다. COLMAP은 2d 이미지들로 3d 정보, 즉 카메라의 위치, 방향 정보를 얻어내는 기술로 대략적인 과정은 아래 그림 8과 같다.



<그림 8 : COLMAP 과정>

2.3.3. Unity AR Core

Unity ARCore는 Google의 ARCore를 Unity 게임 및 애플리케이션 개발 환경에 통합하는 플러그인이다. 안드로이드 기기에서 증강 현실(AR) 경험을 가능하게 하는 Google의 플랫폼이며 주요기능은 다음과 같다.

- 환경인식
  - 평면 탐지: ARCore는 실제 세계의 수평 표면(예: 테이블, 바닥 등)을 인식할 수 있다.
  - 포인트 클라우드: 3D 지점의 클라우드를 제공하여 물리적 공간을 이해하는 데 도움을 준다

- 모션 추적 : 사용자의 폰이 물리적 공간에서 어떻게 움직이고 있는지 추적한다. 가상 객체를 정확하게 배치하고 애니메이션 효과를 주는 데 주로 사용된다.

- 라이트 추정 : 실제 세계의 조명을 추정하여 가상 객체의 조명을 실제와 유사하게 만들어, 가상환경을 더 실감나게 만든다

- 증강 이미지 : 사전에 정의된 2D 이미지를 인식하고 그 위에 3D 컨텐츠를 오버레이할 수 있다.

본 연구에서는 AR Tracked Images Manager, AR Raycast Manager, AR Plane Manger, AR Occlusion Manager와 같은 기능들을 이용하여 연구를 진행하였다.

2.3.4. Android – Unity Embedding

안드로이드에 유니티를 추가하기 위해, 유니티 프로젝틀를 안드로이드 버전에 맞게 export 한다. 이후 결과로 나온 unityLibrary 폴더를 안드로이드 프로젝트에 추가한다. 안드로이드에서 유니티를 실행 할 수 있도록, gradle에 unityLibrary를 포함시키고 UnityPlayerActivity를 AndroidManifest에 추가한다. 유니티 AR과 연동하기 위해, arcore_client, UnityARCore, ARPresto, unityandroidpermissions를 연결한다.

안드로이드에서 유니티 프로젝트를 실행하기 위해서는 제품의 상세 페이지에 접근해야한다.

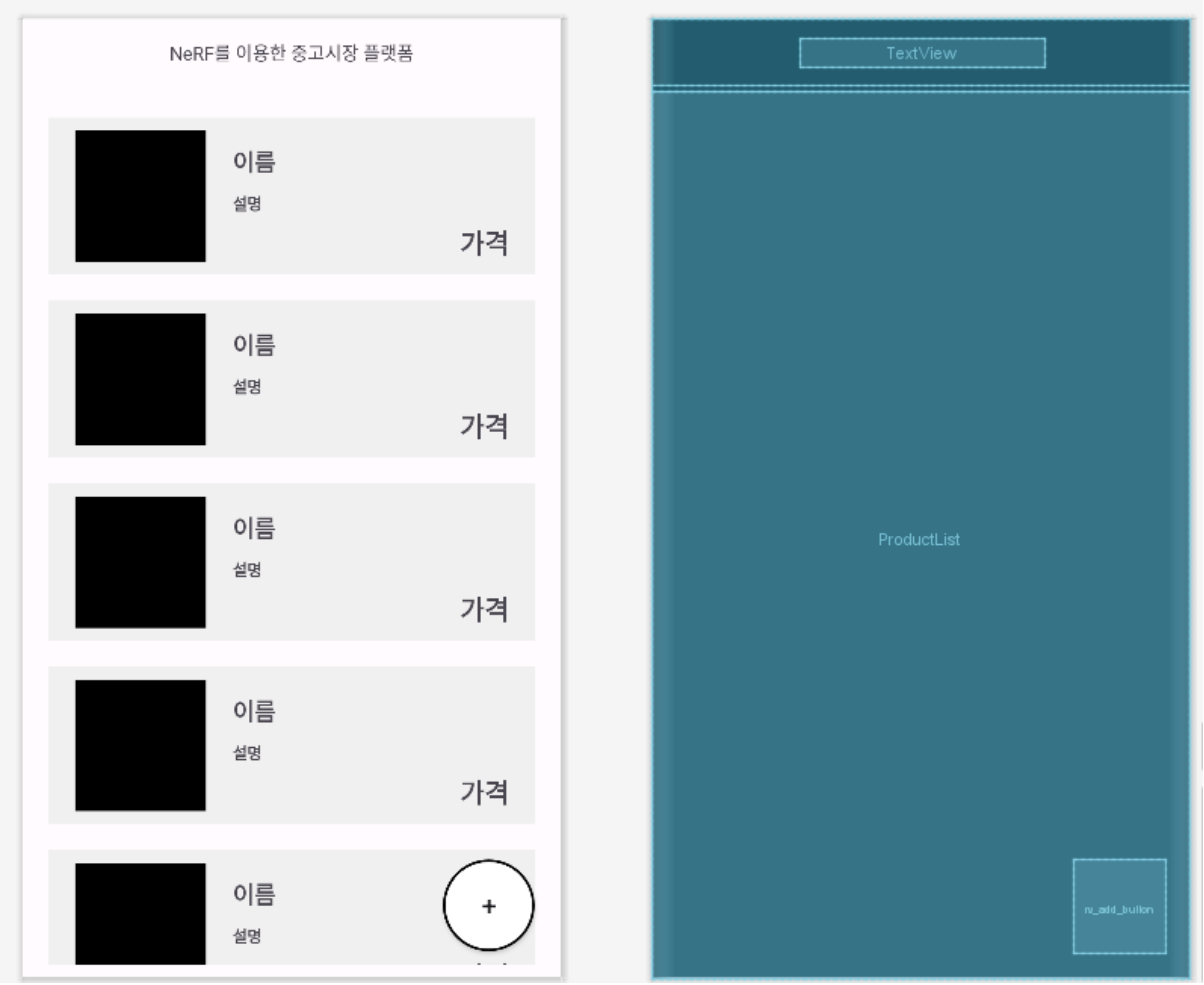
사용자가 제품 글을 서버에 올리는 경우, 서버에서 글 마다 ID를 할당한다. 어플리케이션 실행 시, 메인 액티비티에서 ID를 포함한 글 정보를 가져온다.제품의 상세 페이지에서 유니티를 실행하는 경우, 사용자가 선택한 제품의 ID를 유니티에 전달한다.

3. 연구 내용

3.1. Android 파트

유니티는 자체적으로 안드로이드 어플리케이션을 빌드 가능하다. 카메라나 GPS 같은 모바일 하드웨어에 관련된 API를 제공하고, 디자인 관련 에셋을 제공한다. 그러나 유니티로 만든 어플리케이션은 무거운 런타임 라이브러리를 포함해, 앱의 크기가 크고, 배터리 사용량이 많다. 최적화된 성능을 위해 안드로이드로 개발을 시작했다. MVP 구성을 위해 4가지 액티비티로 설정했다.

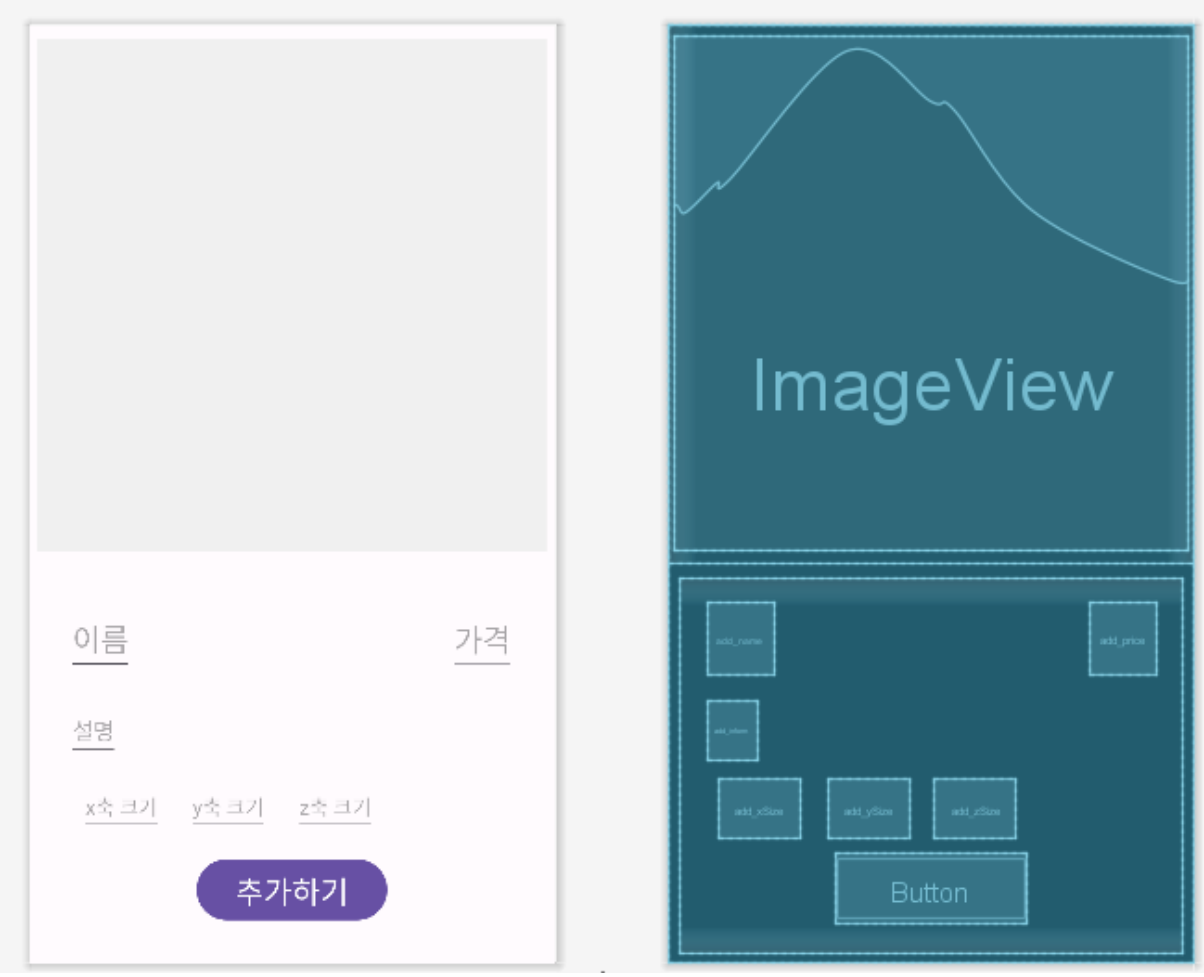
3.1.1. 메인페이지



<그림 9 : 메인 페이지 레이아웃>

Volley를 사용하여 서버에서 전체 글에 대한 정보를 가져온다. 서버에서 가져온 데이터를 리사이클러 뷰 어댑터를 통해 리사이클러 뷰로 만든 후, 리사이클러 뷰의 아이템을 클릭하는 경우 제품의 상세 페이지로 이동하는 이벤트를 추가한다. 이때의 Intent는 제품의 ID, 이름, 설명, 가격을 object 형식으로 만들어 전달한다.

3.1.2. 제품 등록 페이지



<그림 10 : 제품 등록 페이지 레이아웃>

이미지 뷰를 클릭 시 갤러리에 접근하여, 동영상을 선택할 수 있도록 만든다. 각각의 텍스트 컴포넌트에 입력된 값을 통해 Volley Request를 구성하여, 서버에 전달한다. 전송에 성공한 경우, 다시 메인 페이지로 돌아간다.

3.1.3. 제품 상세 정보 페이지

<그림 11 : 제품 상세 정보 페이지 레이아웃>

메인 페이지에서 Intent로 object를 받아 제품의 사진, 이름, 가격, 설명을 화면에 표시한다. 체험하기 버튼 클릭 시, 제품의 ID를 message 변수에 담아 유니티 연동 페이지로 전달한다.

3.1.4. 유니티 연동 페이지

유니티와 연동하기 위해서, 유니티 프로젝트를 안드로이드 버전으로 export 한 후에, unityLibrary를 안드로이드 프로젝트에 연결한 후, 버전에 맞게 gradle과 manifest를 수정한다. 유니티 연동 페이지가 실행된다면, 바로 UnityPlayerActivity를 실행한다. 상세 페이지에서 Intent를 통해 제품의 ID 값을 받은 후, UnityPlayer의 UnitySendMessage 메소드를 사용하여, 유니티의 DataManager 컴포넌트의 ReceivedMessage 메소드에 제품의 ID 값을 발신한다.

3.2. Server 및 NeRF Studio 파트

3.2.1. NeRF 모델 비교

NeRF model	Instant-NGP	Mip-NeRF	Nerfacto
학습량(기본값)	30,000 iterations	1,000,000 iterations	30,000 iterations
학습소요시간	약13분	24시간이상	약13분

<그림 12 : NeRF model별 학습량, 학습소요시간>

(CPU: Intel i5-12600K, VGA: NVIDIA GeForce RTX 3080Ti)

NeRF Studio는 몇몇개의 NeRF 모델을 제공하며, 각각 간의 장단점이 존재한다. 대표적으로 자주 사용하는 3가지의 모델을 대상으로 테스트를 해봤으며, 결과는 그림 12와 같다.

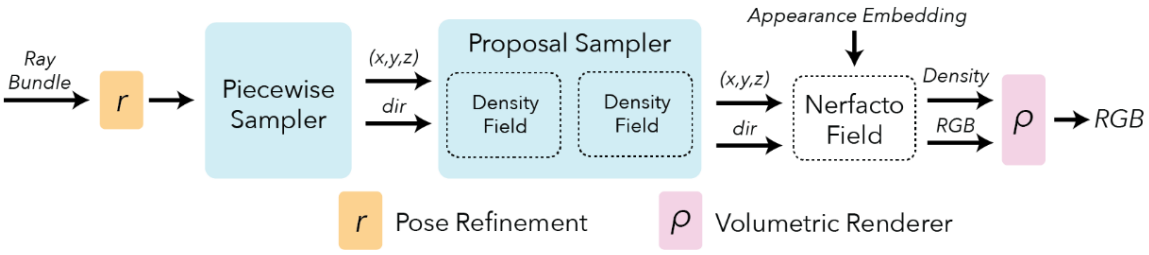
- instant-NGP : 속도향상을 목표로 한 model로 학습소요시간은 약 13분으로 매우 빠른 편이다.
- Mip-NeRF : 속도향상이 아닌 Anti-Aliasing을 목표로 한 model이다보니 결과의 품질은 좋으나, 학습요구량이 많고, 각 iteration의 소요시간 또한 다른 두 model에 비해 길어서 학습소요시간이 매우 길다.
- Nerfacto : nerfstudio에서 추천하는 model로 여러 model을 통합시킨 model이다. 학습소요시간은 약 13분으로 instant-NGP와 거의 동일하고, mesh 추출의 소요시간은 instant-NGP보다 3배이상 빠르다. 또한, Nerfacto의 경우 다른 두 model과 달리 학습단계에서 normal을 예측하도록 옵션을 줄 수 있는데, 이렇게 할 경우 학습시간이 1.2~1.3배로 늘어나기는 하지만 추출되는 mesh의 품질이 매우 좋아진다. 시현 결과는 아래 그림 13와 같다.

(좌)instant-NGP mesh 추출 결과 (우)Nerfacto(학습시 normal 예측) mesh 추출 결과

<그림 13 : 각 모델별 Mesh의 추출 결과>

3.2.2. Nerfacto

Nerfacto의 파이프라인은 아래 그림 14와 같다.



<그림 14 : Nerfacto Pipeline>

카메라 포즈(camera pose) 예측에서 오류는 드물지 않게 발생하는데, 이 오류는 NeRF의 성능 감소를 야기한다. Pose Refinement 단계에서 loss gradient를 역전파(backpropagation)할 수 있고, 이를 통해 카메라 포즈를 최적화하고 개선할 수 있다

Piecewise Sampler를 사용해 장면의 초기 샘플 세트를 생성하는데, 샘플의 절반을 카메라로부터 최대 1의 거리까지 균일하게 할당하고 나머지 절반은 각 샘플마다 step size가 증가하도록 분배한다. 이렇게 함으로써 가까운 물체에 대한 밀집된 샘플 세트를 유지하면서 동시에 멀리 있는 물체를 샘플링할 수 있다.

Proposal Sampler는 샘플 위치를 최종 렌더링에 가장 큰 기여를 하는 장면 영역으로 통합하는데, 이는 재건(reconstruction) 품질을 크게 향상시킨다.

3.2.3. 서버

서버는 HTTP POST 메시지를 통해 게시글 등록 요청을 받으면 게시글 정보를 DB에 저장하고, 첨부된 동영상을 입력으로 nerfstudio를 실행시켜서 mesh파일을 추출한 후 서버에 저장한다.

이때 nerfstudio를 통해 데이터 전처리, 모델 학습, mesh 추출을 순서대로 수행시키기 위한 shell command들을 shell script로 작성하여, 서버는 shell script만 실행시키면 되도록 하였다.

```
#!/bin/bash

id=$1

echo "start sh"

#process
echo "start process"
ns-process-data video --data /workspace/video/$id.mp4 --output-dir /workspace/result/data/$id

#train
echo "start train"
ns-train nerfacto --data /workspace/result/data/$id --output-dir /workspace --experiment-name result --method-name train --timestamp
$id --pipeline.model.predict-normals True --viewer.quit-on-train-completion True --logging.steps-per-log 30001

#export
echo "start export"
ns-export poisson --load-config /workspace/result/train/$id/config.yml --output-dir /workspace/result/mesh/$id --target-num-faces 500
00 --num-pixels-per-side 2048 --normal-method model_output --num-points 1000000 --remove-outliers True --use-bounding-box True --boun
ding-box-min -1 -1 -1 --bounding-box-max 1 1 1

#HTTP message(inform server nerf is done)
echo "send message"
curl -X GET "http://localhost:3389/finishNerf?id=$id"

echo "All done"
```

<그림 15 : nerfstudio 실행을 위한 shell script>

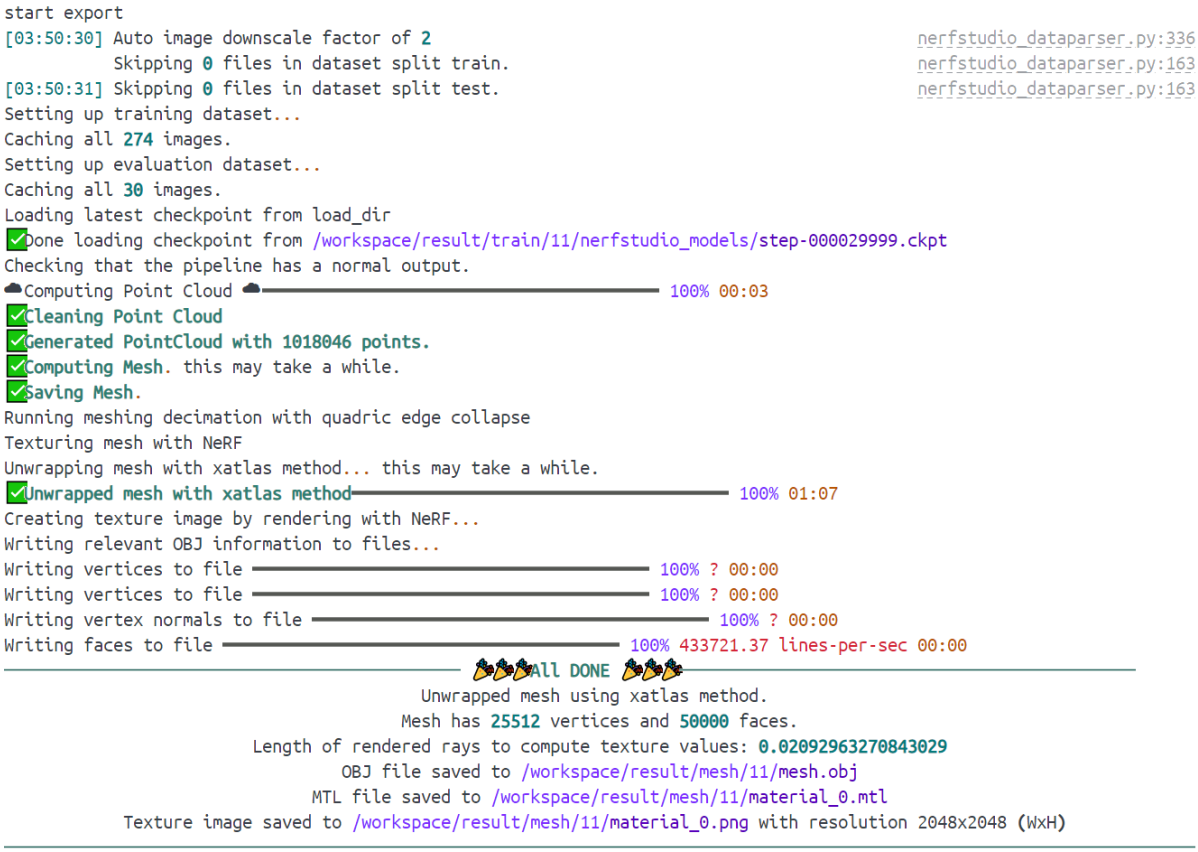
```
start sh
start process
Number of frames in video: 2426
Number of frames to extract: 304
[03:31:10] Done converting video to images.                          process_data_utils.py:173
[03:32:41] Done downscaling images.                                  process_data_utils.py:361
[03:33:01] Done extracting COLMAP features.                              colmap_utils.py:131
[03:34:30] Done matching COLMAP features.                                colmap_utils.py:145
[03:35:44] Done COLMAP bundle adjustment.                                colmap_utils.py:167
[03:35:49] Done refining intrinsics.                                     colmap_utils.py:176
           All DONE                                          video_to_nerfstudio_dataset.py:118
           Starting with 2426 video frames                   video_to_nerfstudio_dataset.py:121
           We extracted 304 images                           video_to_nerfstudio_dataset.py:121
           We downsampled the images by 2x, 4x and 8x        video_to_nerfstudio_dataset.py:121
           Colmap matched 304 images                         video_to_nerfstudio_dataset.py:121
           COLMAP found poses for all images, CONGRATS!      video_to_nerfstudio_dataset.py:121
```

<그림 16 : shell script 실행 결과 - 데이터 전처리>

```
start train
[03:35:54] Using --data alias for --data.pipeline.datamanager.data                    train.py:235

           Saving config to: /workspace/result/train/11/config.yml                experiment_config.py:128
           Saving checkpoints to: /workspace/result/train/11/nerfstudio_models              trainer.py:136
           Auto image downscale factor of 2                                  nerfstudio_dataparser.py:336
           Skipping 0 files in dataset split train.                          nerfstudio_dataparser.py:163
[03:35:55] Skipping 0 files in dataset split val.                            nerfstudio_dataparser.py:163
Setting up training dataset...
Caching all 274 images.
Setting up evaluation dataset...
Caching all 30 images.
                                    Viewer
╭──────────────────────────────────────────────────────────────────────────────╮
│ HTTP │ https://viewer.nerf.studio/versions/23-05-15-1/?websocket_url=ws://localhost:7007 │
╰──────────────────────────────────────────────────────────────────────────────╯
[NOTE] Not running eval iterations since only viewer is enabled.
Use --vis {wandb, tensorboard, viewer+wandb, viewer+tensorboard} to run with eval.
No Nerfstudio checkpoint to load, so training from scratch.
Disabled tensorboard/wandb event writers
[03:36:10] Printing max of 10 lines. Set flag --logging.local-writer.max-log-size=0 to disable line      writer.py:408
           wrapping.
Step (% Done)       Train Iter (time)    ETA (time)
------------------------------------------------------------
0 (0.00%)           2 s, 300.794 ms      19 h, 10 m, 23 s
------------------------------------------------------------------------------------------------------
Viewer at: https://viewer.nerf.studio/versions/23-05-15-1/?websocket_url=ws://localhost:7007
                                  Training Finished
╭──────────────────────────────────────────────────────────────────────────────╮
│ Config File           │ /workspace/result/train/11/config.yml                │
│ Checkpoint Directory  │ /workspace/result/train/11/nerfstudio_models         │
╰──────────────────────────────────────────────────────────────────────────────╯
Printing profiling stats, from longest to shortest duration in seconds
Trainer.train_iteration: 0.0278
VanillaPipeline.get_train_loss_dict: 0.0175
```

<그림 17 : shell script 실행 결과 – 모델 학습>



<그림 18 : shell script 실행 결과 – mesh 추출>

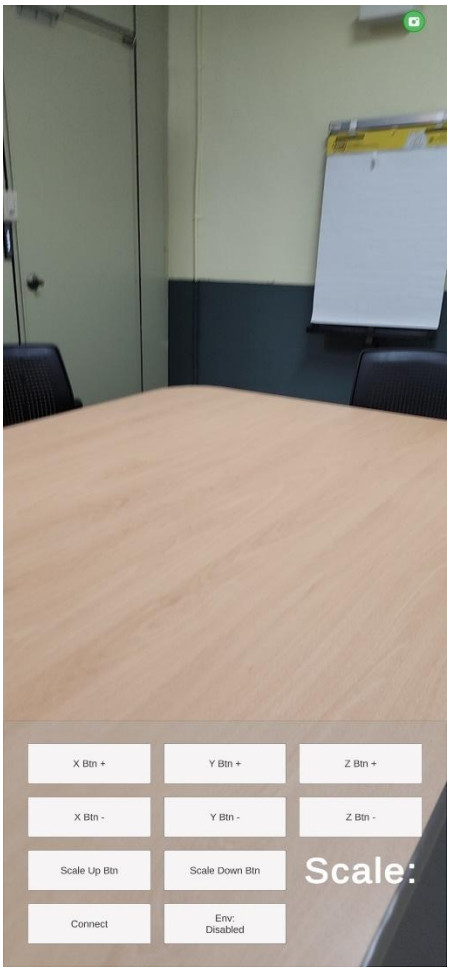
이후 서버는 HTTP GET 메시지를 통해 게시글 목록, 게시글 내용 등을 요청 받으면, DB 에서 해당 정보를 조회해서 전송한다.

HTTP GET 메시지를 통해 mesh 파일(obj, mtl, png 파일)을 요청 받으면, 서버에 저장된 해당 파일을 전송한다.

3.3. AR 파트

3.3.1. 메인 화면 및 모델 생성

본 연구에서의 Unity AR Core는 사전에 서버에서 만들어진 중고물품 모델을 실제 모바일 AR 환경에서 배치하여 중고 물품을 확인하는데 목표를 둔다. 어플리케이션에서 체험하기를 누르면, 사진에 나와있는 물체를 3D AR로 확인하기 위해 Unity어플과의 연동이 이루어진다. 이때, 휴대폰 후방 카메라를 이용한 AR화면이 나오며(그림 19 참조), Connect 버튼을 누른 후 원하는 위치에 터치하면, 보고 있었던 물체의 3D 모델이 실제 자기 위치에 AR로 생성된다(그림 20 참조). 이때, 생성된 3D모델은 서버에 있는 obj, mtl, png 파일을 이용하여 생성된다.



<그림 19 : 어플을 실행시켰을 때 기본 화면>

<그림 20 : Connect를 누른 후 화면을 터치했을 때 모델이 생성된 화면>

3.3.2. Rotation 및 Scaling

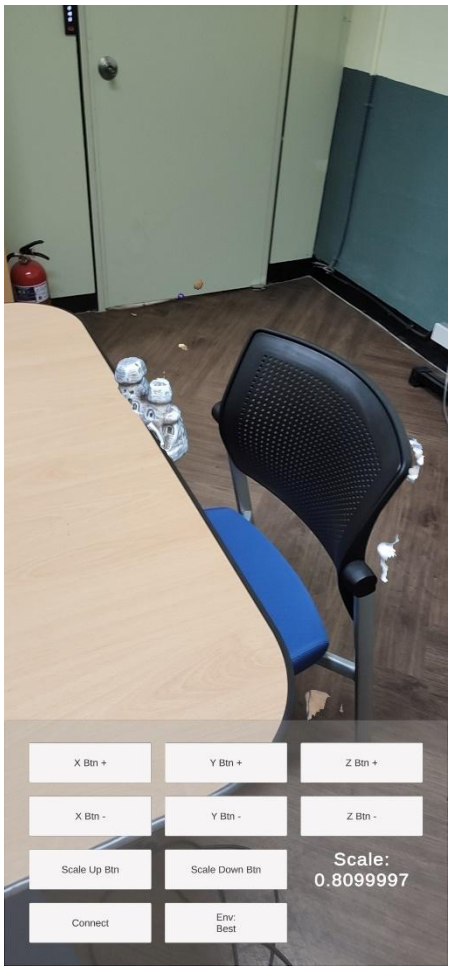
생성된 오브젝트는 UI의 Scale로 크기를 알 수 있으며, Scale Up Btn과 Scale Down Btn으로 사전에 판매자가 입력하였던 크기에 맞춰 조절 할 수 있다(그림 21 참조). 또한, X축, Y축, Z축으로 오브젝트에 회전가능한 기능을 추가하여, 자기가 원하는 위치에 확실하게 고정할 수 있게 도와주는 기능도 추가하였다(그림 22 참조).

<그림 21 : Scale Up Btn을 눌러 모델의 크기를 키운 화면>

<그림 22 : Y축으로 모델을 회전시켜 모델이 90도 회전한 화면>

3.3.3. Occlusion

마지막으로, 실제감을 증폭시키기 위하여 Occlusion을 적용시키는 방식도 채택하였다. 이 기능으로, 사용자는 물체의 깊이를 인식하는데 도움을 줄 것이며, 물체의 모양, 크기 및 상대적인 위치를 파악하여 가상 객체를 현실적으로 배치하는데 도움을 줄 것 이다(그림 23 참조). Occlusion에는 Unity AR Core에 기반한 3가지의 기능이 있으며, 각각 Fastest, Medium, Best가 있다. 각각 Occlusion이 되는 성능과 속도의 TradeOff에 기반한 기능이며, Occlusion버튼을 누를 때마다 Mode가 변경되어 적절한 모드가 사용 가능하다.

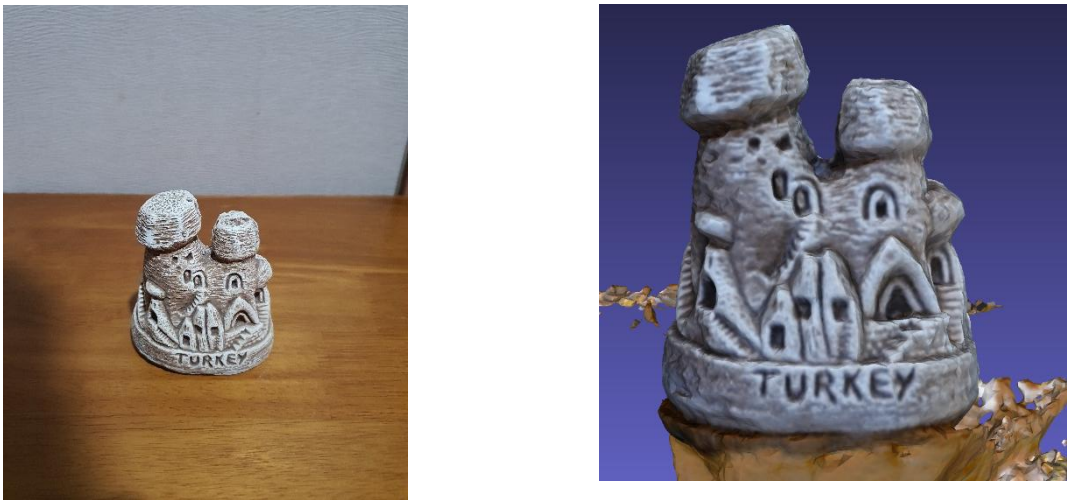


<그림 23 : Best Occlusion으로 생성된 모델이 책상의 뒤편으로 일부 가려진 화면>

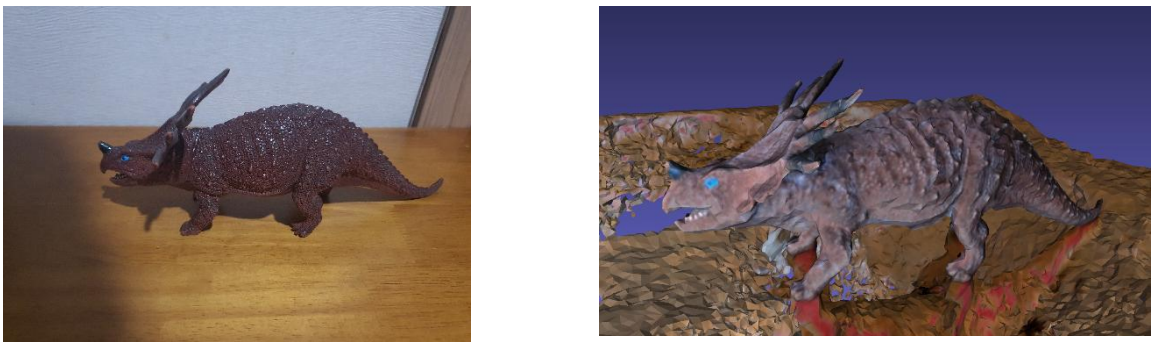
4. 연구 결과 분석 및 평가

4.1. 연구결과 분석

본 연구 결과를 분석하기 위해, Statue, Toy, Pot 총 3가지의 일상생활에서 사용하는 물체를 가지로 테스트를 진행하였다. NeRF로 학습시킨 Mesh의 결과는 밑의 그림 24, 그림 25, 그림 26과 같으며, 왼쪽에는 NeRF를 학습시킬 때 사용된 동영상의 일부분, 그리고 그 동영상으로 학습된 모델의 결과물이다.



<그림 24 : 원본 모형 Statue(좌), NeRF로 학습된 Statue(우)>

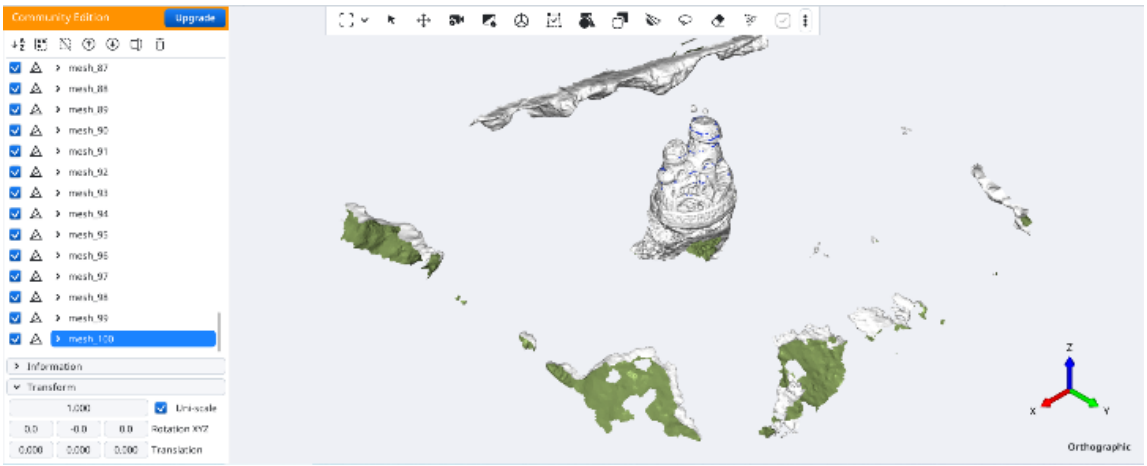


<그림 25 : 원본 모형 Toy(좌), NeRF로 학습된 Toy(우)>

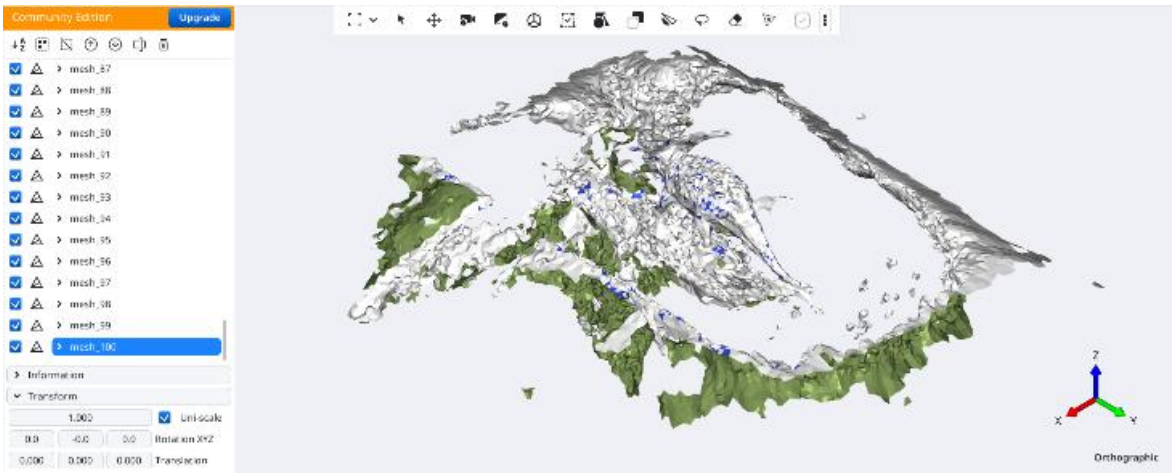
<그림 26 : 원본 모형 Pot(좌), NeRF로 학습된 Pot(우)>

연구결과, 육안으로 보기에는 꽤나 높은 유사도의 NeRF Mesh가 생성되는 것으로 확인되었다. 학습시간도 평균적으로 10~15분정도, 비교적 빠르게 학습되는 것도 확인되었다.

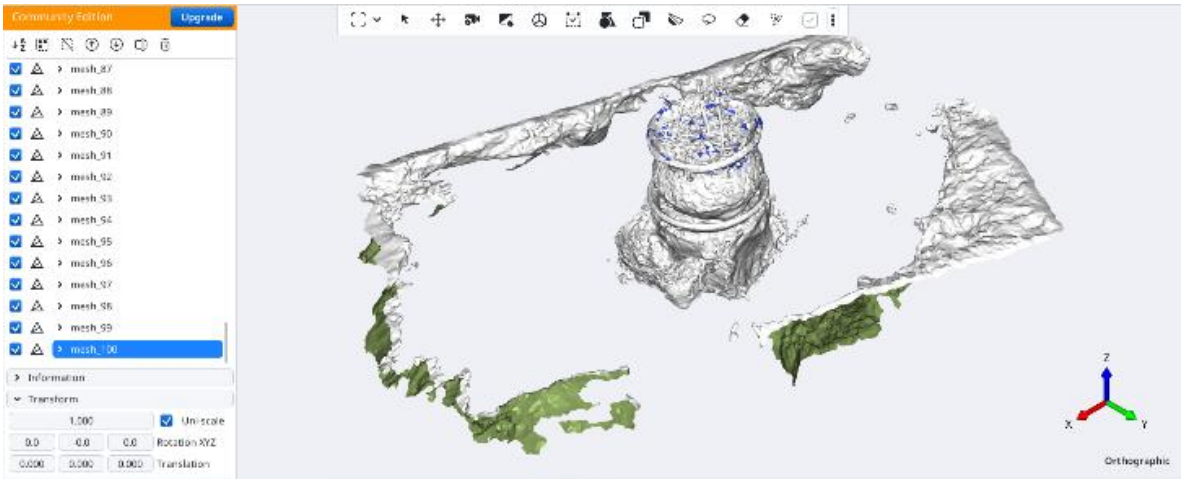
그러나, Mesh에서 확인되는 바닥부분 밑 주변 배경은 제거하지 못하는 단점도 확인되었으며, Mesh가 지저분해 Mesh Segmentation도 적용시킬 수 없는 단점도 확인되었다.(그림 27, 28, 29 참고) 밑의 그림들은 Mesh Segmentation을 시도하다가 실패한 과정을 보여주며, Mesh 위에 있는 파란색이 Segmentation을 시도한것이다.



<그림 27 : Statue를 Mesh Segmentation한 것>



<그림 28 : Toy를 Mesh Segmentation한 것>



<그림 29 : Pot를 Mesh Segmentation한 것>

4.2. 산학협력 멘토링

산학협력 멘토링을 통한 피드백을 통해 다양한 측면에서 졸업과제를 발전시킬 수 있는 중요한 포인트들을 확인하였다. 멘토링 세션은 중간보고서를 읽은 뒤 피드백 받는 형식으로 받았다.

먼저, 중고 시장을 타겟으로 한 주된 이유는 중고 상품의 진상 상태와 품질을 온라인으로만 판단하기 어렵다는 문제점을 다시한번 제 상기시켰으며, NerF 기술을 통해 사용자는 상품의 3D 모델을 AR로 체험함으로써 상품의 상태를 더 정확하게 평가하는 것을 목표로 설정하였다.

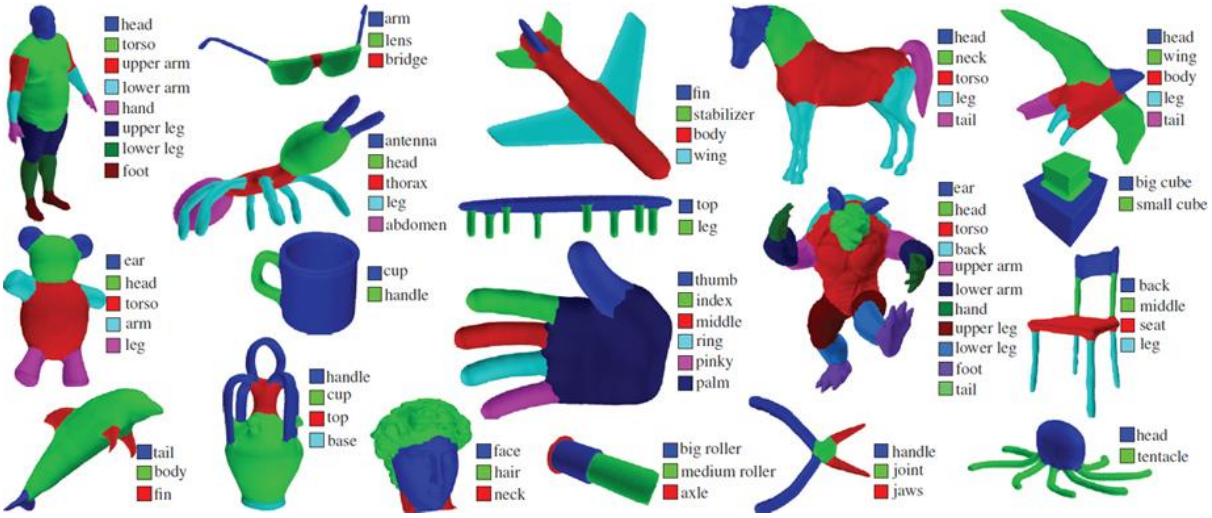
또한, NerF 기술의 상세한 구현 방법과 이를 AR과 효과적으로 연동하기 위하여, NeRF Framework와 Unity AR Core와의 연동을 더욱 더 신경썼다. 마지막으로, 저희 연구 주제와 비슷한 사업을 하는 리콘랩스라는 기업의 성공 사례를 분석하여, 우리 프로젝트에 적용할 수 있는 인사이트에 대하여 확인하여 연구를 반전시킬 수 있었다.

5. 결론 및 향후 연구 방향

5.1. NeRF의 Mesh Segmentation

NeRF를 통해 추출한 mesh를 보면 우리가 원하는 물체 외에 배경의 일부 등도 같이 추출된 것을 확인할 수 있다. 그렇기에 우리는 우리가 원하는 물체만을 남겨서 mesh의 품질을 향상시키려고 하였다.

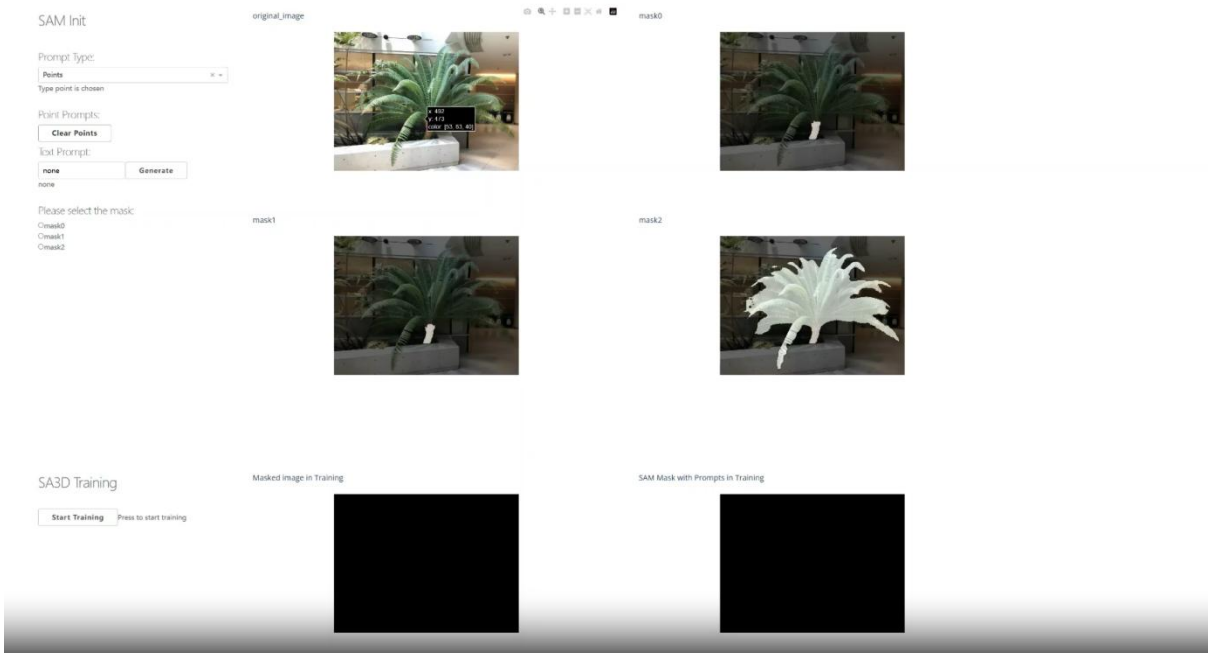
먼저 NeRF를 통해 mesh를 추출한 후, mesh segmentation(그림 30 참조)을 통해 가장 큰 component만 남기려고 했다. 하지만 mesh segmentation에 대해 찾아보면 mesh



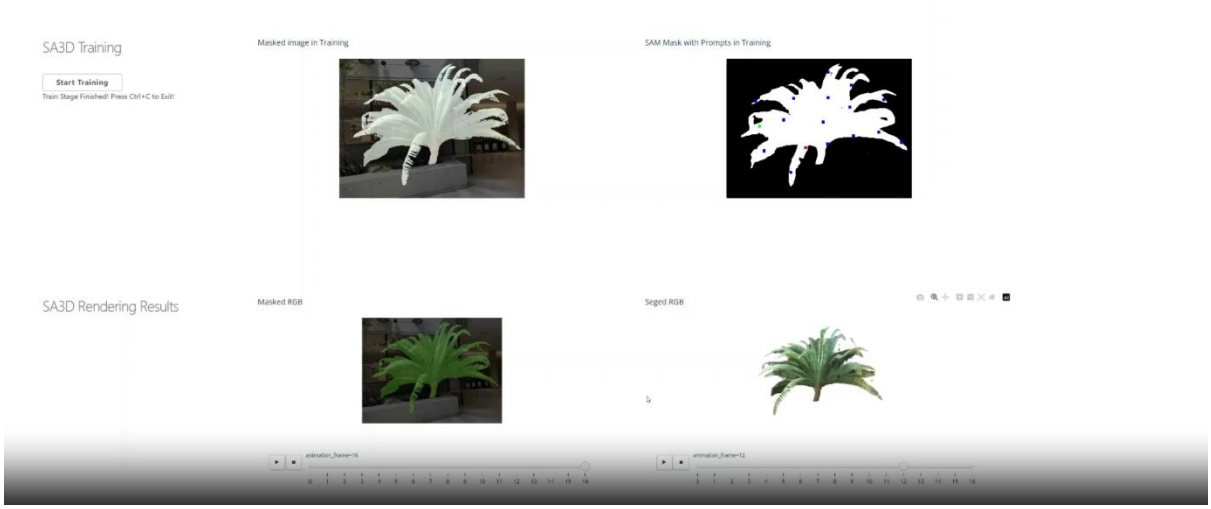
의 component들을 구분하는 정도일뿐, 원하는 component만 추출해내는 건 보이지 않았다. 만약 있어서 사용한다고 하더라도 obj파일에 대해서만 수정이 일어나고, mtl파일에 대해서는 수정이 일어나지 않을 것이기 때문에 obj파일과 png파일간의 mapping에 문제가 생겨 사용하기 힘들 것이라고 판단하였다.

<그림 30 : mesh segmentation 예시>

NeRF를 돌리기 전에 사용자가 직접 GUI를 통해 원하는 물체를 선택하면 해당 물체만 mesh로 추출해주는 SA3D(그림 31, 그림 32 참고)라는 것이 있긴 하였지만, GUI를 통해 동작하다 보니 사용자의 조작없이 서버단에서 자동으로 동작하는 것이 힘들었다. 'mesh의 품질 증가'의 가치가 'NeRF 자동화'의 가치보다 크지 않다고 생각되어 SA3D를 사용하긴 힘들 것이라고 판단하였다.



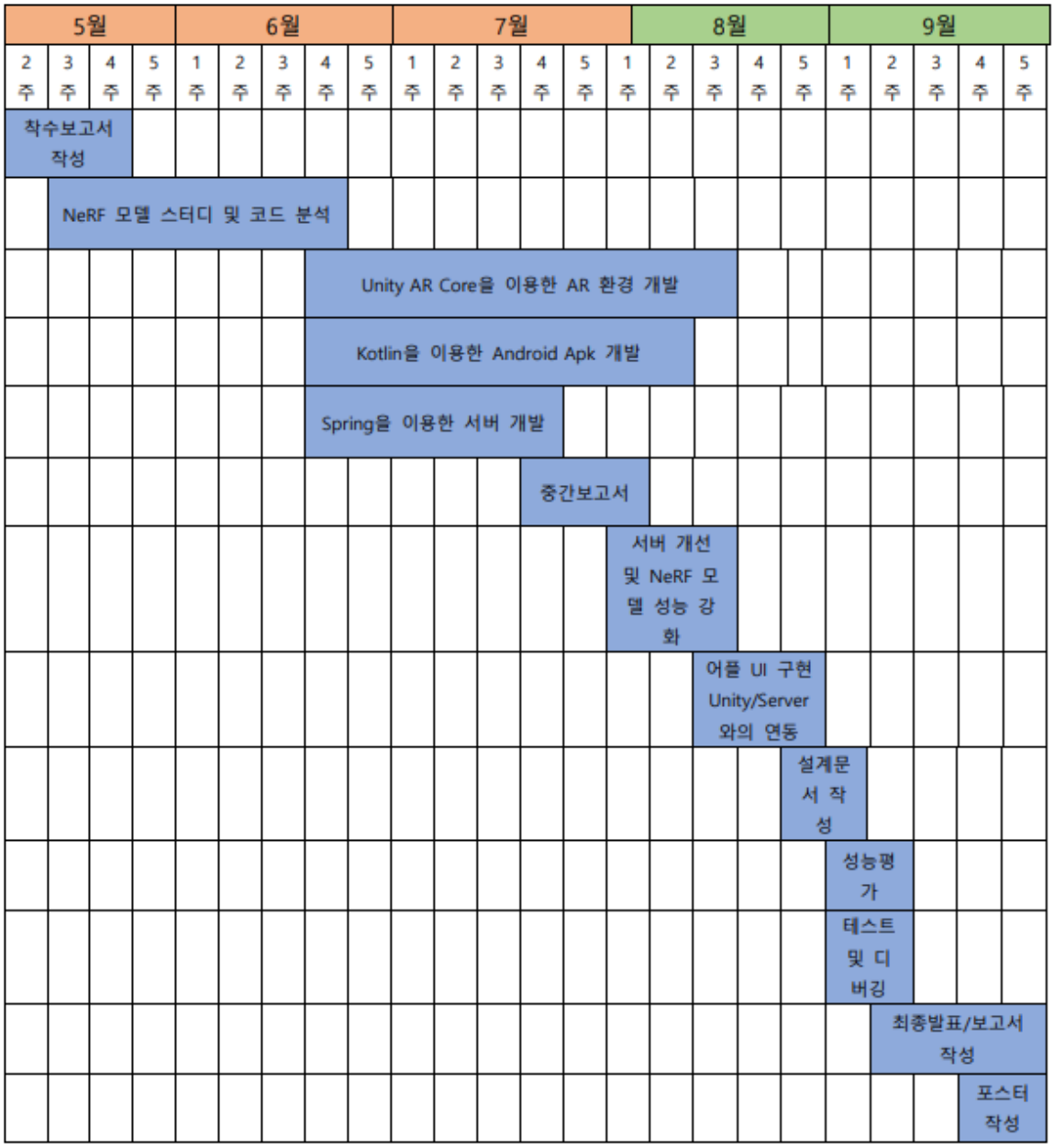
<그림 31 : SA3D GUI 예시 1>



<그림 32 : SA3D GUI 예시 2>

6. 개발 일정 및 역할 분담

6.1. 개발 일정



6.2. 역할 분담

이름	역할분담
은승우	- Unity AR Core 개발 - Mesh 전처리 - Occlusion/택스쳐 개발
김성현	- 안드로이드 프런트 개발 - 안드로이드/유니티 연동화 - 입력 데이터관리
권재섭	- shell script를 통한 nerfstudio 자동화 - spring boot를 사용해 서버 개발 - 서버에 MariaDB 연동

7. 참고 문헌

[1] J. Schonberger and J. Frahm, "Structure-from-Motion Revisited," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 4104-4113, 2016

[2] E. Kalogerakis, A. Hertzmann, and K. Singh, "Learning 3D Mesh Segmentation and Labeling," ACM transactions on graphics, Vol. 29, No. 4, pp. 102, 2010

[3] Jiazhong Cen, Zanwei Zhou, Jiemin Fang, Chen Yang, Wei Shen, Lingxi Xie, Dongsheng Jiang, Xiaopeng Zhang, Qi Tian. Segment Anything in 3D with NeRFs [Online]. Available: https://jumpat.github.io/SA3D

[4] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, Ren Ng, "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis." ACM Transactions on Graphics (TOG), Vol. 39, No. 2, 2020, Article No. 31.

[5] Matthew Tancik, Ethan Weber, Evonne Ng, Ruilong Li, Brent Yi, Justin Kerr, Terrance Wang, Alexander Kristoffersen, Jake Austin, Kamyar Salahi, Abhik Ahuja, David McAllister, Angjoo Kanazawa, “Nerfstudio: A Modular Framework for Neural Radiance Field Development”, SIGGRAPH '23: ACM SIGGRAPH 2023 Conference Proceedings, July 2023,Article No.: 72, Pages 1–12

[6] Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, Pratul P. Srinivasan, “Mip-NeRF: A Multiscale Representation for Anti-Aliasing Neural Radiance Fields”, ICCV 2021 Oral

[7] Thomas Müller, Alex Evans, Christoph Schied, Alexander Keller, “Instant Neural Graphics Primitives with a Multiresolution Hash Encoding” ACM Transactions on Graphics (SIGGRAPH), July 2022