종합설계 프로젝트 수행 보고서

프로젝트명	**증강 현실기술을 이용한 실시간 타일 인테리어 프리뷰 서비스**
팀번호	S1-8
문서제목	수행계획서(O) 2차발표 중간보고서(O) 3차발표 중간보고서(O) 최종결과보고서(O)

2020.11.19

팀원 : 김동원 (팀장)

송재우

허제민

지도교수 : 박정민 교수

지도교수 : 최종필 교수

목 차

1-1. 작품 선정 배경 및 필요성

필요성	내 용
수요배경	• 재개발 지역, 신도시가 늘어나고, 1인 가구가 늘어남에 따라 국내 **인테리어, 리모델링 시장**의 규모가 **계속해서 성장**하는 추세다. • 이에 **셀프 인테리어 시공** 또한 젊은 세대에게 새로운 트렌트로 자리 잡고 있다.
필요성 1	• 가구를 배치하는 것과 달리, 타일을 한번 시공하는 데에는 **많은 시간과 노력**이 필요하다. 하지만 시공의 결과가 마음에 들지 않을 경우 더 큰 손해를 감수하고 다시 시공을 하거나 불만족스러운 결과에 만족을 하게 된다. • 따라서 소비자는 시공 전, 결과에 대한 **불확실성을 감수**해야한다.
필요성 2	• **타일 프리뷰 서비스**를 이용하면, 이러한 불확실성을 감수하지 않아도 되고 이로 인해 소비자로 하여금 **잠재적인 소비에 대한 기대효과**를 이끌어낼 수 있다.

1-2. 기존 연구 분석

개발내용	내 용
기존 연구	■ **증강현실 기반의 인테리어 설계 시스템에 관한 연구** • (소개) 현실공간과 가상공간을 접목 시킨 증강현실을 통해 가상공간 안에서 인테리어를 설계하는 시스템의 프로토타입을 제시한다. • (장점) 사이버 공간 안에 인간의 오감을 통한 상호작용을 실현하여 현실세계에서의 활동 또는 **공간적, 물리적 제약에 의해 직접 경험하지 못하는 상황을 간접적으로 체험**할 수 있게 한다. ■ **공간증강현실을 활용한 실내 벽지 인테리어 시스템** • (소개) 증강현실을 활용해 실내 벽지 인테리어 프리뷰 서비스를 제공한다. • (장점) 소비자가 **원하는 벽지의 색상과 무늬를 선택**할 수 있고 색상변환기술을 사용하여 벽지의 주변 환경과 조화를 이루는 벽지를 추천해준다. • (단점) 원하는 **구역을 설정하지 못함.** ■ **증강현실을 이용한 인테리어 도우미** • (소개) 카메라를 통해 패턴을 검색하여 가상의 공간을 구성하고 DB에 저장된 가상 이미지를 불러와 **가상의 공간에 영상을 송출**한다. • (장점) 이미지를 출력하는데 있어서 위치이동, 크기 변환, 방향 변환을 수행하도록 하여 **원하는 모양과 위치에 물체를 배치**시킬 수 있다. • (단점) **특정 패턴**이 있어야만 가상 이미지를 배치시킬 수 있음.

1-3. 개발 목표

최종목표

■ **증강 현실 기술(AR)** 기반으로 **실시간 타일 인테리어 프리뷰 서비스**를 사용자에게 제공하여 **사용자의 편의를 증대**시킨다.

단계	내 용
1단계	■ 프리뷰 서비스 구현을 위한 **조사** • 타일의 **종류**와 **재질의 특성** 조사. • 사용자가 실질적으로 사용할 **증강 현실(Augmented Reality, AR)**에 대해 조사. • 증강 현실 구현에 필요한 **소프트웨어, 라이브러리** 조사.
2단계	■ 프리뷰 서비스 구현을 위한 아키텍처 설계 • **현실 세계에서 작동**하고 **제어**가 가능한 **AR 환경** 설계 • 서비스에 사용할 **타일 데이터베이스** 설계
3단계	■ 프리뷰 서비스 기술 구현 • **UNITY, ARCore**를 이용한 **AR 환경** 구축 • **UNITY Material**을 이용한 타일 **재질 표현** 기능 구현 • 프리뷰할 타일의 정보를 가지고 있는 **데이터베이스** 구현 • 서비스를 쉽게 이용할 수 있는 **사용자 인터페이스** 구현
4단계	■ 구현한 서비스에 대한 **테스팅 수행** • 여러 환경에 대한 **테스트 시나리오** 작성 • 테스트 시나리오에 대한 **단위 테스팅** 진행

1-4. 팀 역할 분담

팀 원	역 할
김동원	AR Core 에서의 물체 시각화 방법 AR 공간에 배치할 포인트 객체 구현
송재우	AR Core를 활용한 바닥 인식 AR 공간에 인테리어 가구 배치 방법
허제민	Unity의 DB 개발 방법 Unity material를 통한 타일 재질 구현 방법

1-5. 개발 일정

일정	1월	2월	3월	4월	5월	6월	7~9월
시스템 설계	●						
시스템 구현		●	●	●			
프로토 타입 완성				●	●		
통합 테스트 및 보완					●	●	●
최종 보고서 검토 및 발표							●

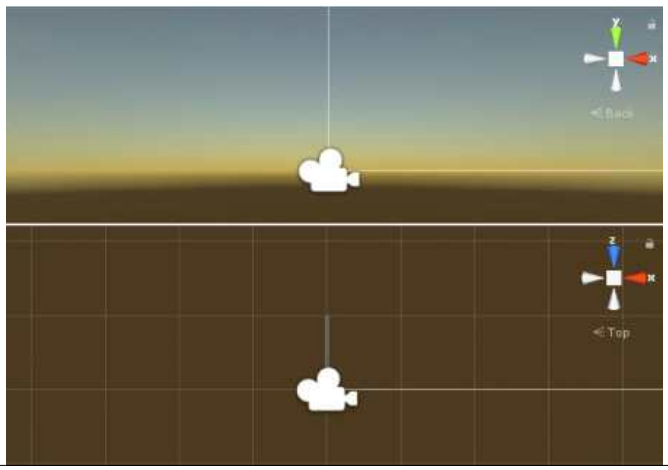
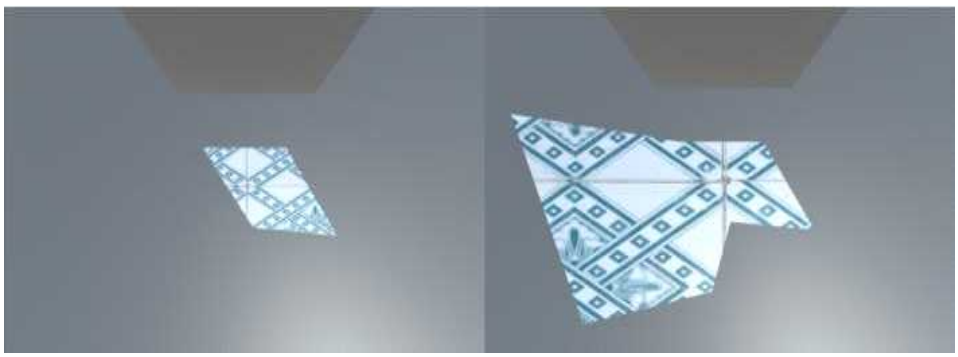
1-6. 개발 환경

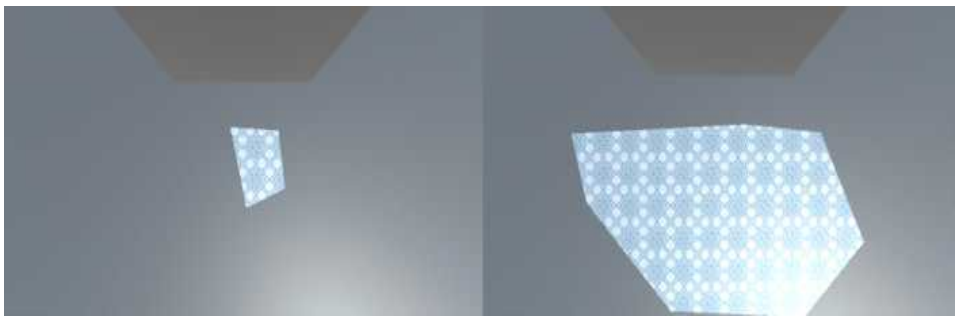
도 구	역 할
Unity	**AR Core** 라이브러리를 사용할 소프트웨어. 전체적인 애플리케이션 및 UI 개발을 위해 사용.
VS studio	MS사의 IDE. Unity 에서의 **스크립트** 작성을 위해 사용.
Blender	**3D 모델을 제작**하는 용도로 사용하는 프리 소프트웨어. 가구 3D 모델을 Unity에서 사용할 수 있도록 최적화하기 위해 사용.
AR Core	Google 에서 제공하는 AR 애플리케이션 **개발용 라이브러리**. AR 공간에서의 상호작용을 위해 사용.

2-1. 개발 내용

모듈	역 할
바닥 인식	AR 공간에 오브젝트를 배치하기 위한 바닥을 인식.
사용자 입력	AR 공간에서의 상호작용을 위해 사용자의 입력을 받아들임.
타일 생성	사용자가 입력한 내용에 따라 AR 공간에 타일을 생성.
가구 배치	인식된 바닥면에 가구를 배치함.

2-2. 문제점 및 해결 방안

	설 명
문제점 1	Unity에서 메쉬를 동적으로 생성하기 위해 사용되는 기존의 Triangulator 스크립트는 3D 공간에서 X, Y축의 데이터만 받아와서 작업을 수행함. 그러나, 바닥면에 배치하는 타일은 X,Z 축의 공간에서 만들어져야 하므로 타일을 생성했을 때 바닥이 아닌 바닥면에 수직으로 생성됨. 
문제점 2	Unity에서 메쉬를 생성했을 때, 메쉬의 전체적인 크기에 따라 타일 이미지의 사이즈도 그 크기에 따라 계속해서 변화함. 

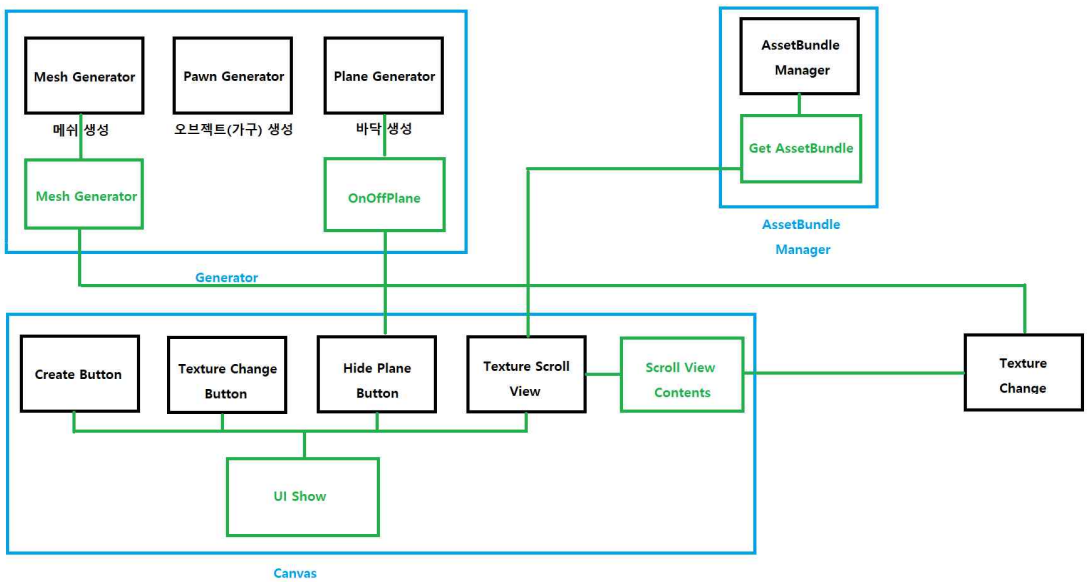
해결 방안 1	기존 X,Y 축 데이터를 가지고 있는 Vector2D 데이터 타입을 다루는 Triangulator의 매개변수 타입을 X,Y,Z 축 데이터를 담고있는 Vector3D 타입으로 바꾸고, 기존 Y축 연산 코드들을 Z축으로 바꾸어 주었음.
해결 방안2	메쉬의 텍스쳐 사이즈를 메쉬의 전체적인 크기로 나눠줌으로써, 타일의 크키와 텍스쳐의 이미지 비율을 같게함. 

2-3. 시험 시나리오

시나리오명	내 용
바닥 인식	• 카메라 기반 디바이스를 이용하여 현실세계를 비춘다. • 오브젝트를 배치할 바닥을 정확히 인식하는 확인.
사용자 입력	• 구역을 설정하는 **포인트 객체**를 이용하여 **원하는 구역을 설정**한다. • 사용자가 터치했을 때, 정확한 위치에 객체가 생성되는지 확인.
메쉬 생성	• 사용자가 입력한 포인트 객체에 따라 정확한 모양의 타일이 만들어지는지 확인.
가구 배치	• 사용자가 입력한 공간에 가구가 정확히 배치되는지 확인.

2-4. 상세 설계

2-4-1. 모듈 간 인터페이스



2-4-2. 시나리오 상세 설계

바닥 인식

```
private List<DetectedPlane> m_NewPlanes = new List<DetectedPlane>();
```

AR Core 라이브러리에서 제공하는 **Detected Plane 클래스** 이용.

Detected Plane 클래스는 AR Core를 통해 **감지되고, 추적되는 평면 정보를 담고 있음.**

```
public void Update()
{
    // Check that motion tracking is tracking.
    if (Session.Status != SessionStatus.Tracking)
    {
        return;
    }

    Session.GetTrackables<DetectedPlane>(m_NewPlanes, TrackableQueryFilter.New);
```

현재 **tracking 상태인지 확인** 후, 평면들을 받아 오기 위해 조건문을 설정.

Tracking 상태(사용자가 평면을 포착하기 위해 카메라를 상하좌우로 이동시킴)일 경우, Session 클래스의 **GetTrackables 함수**의 템플릿 파라미터를 **DetectedPlane**으로 설정, 첫 번째 인자로 앞서 생성한 **평면 List**를 주어, 받아온 평면 정보를 리스트에 저장한다.

두 번째 인자로 쿼리 필터 속성을 New로 설정해 새로운 평면들을 받아오게 함.

GetTrackables 함수를 Update() 메서드 안에서 호출하여 **매 프레임마다 새로운 평면들을 포착**함.

사용자 입력

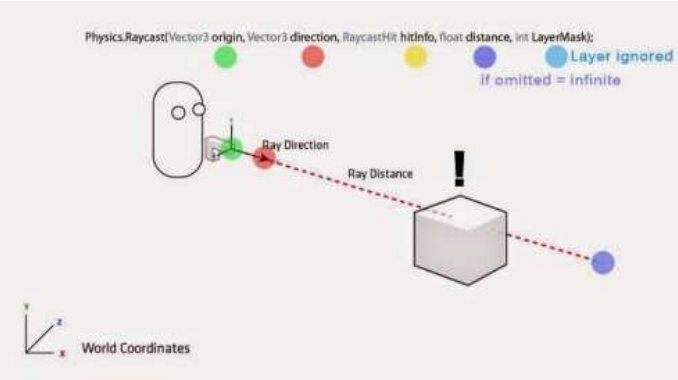
```
37      private List<GameObject> Points = new List<GameObject>();
38      private List<Vector3> positions = new List<Vector3>();
```

사용자가 화면을 터치했을 때, 생성할 포인트 객체를 담을 GameObject 타입의 리스트 생성. 이후에 타일을 만들기 위해 앞서 생성한 포인트 객체들의 위치를 저장할 Vector3 타입의 리스트 생성.

```
47      // Update is called once per frame
48      void Update()
49      {
50          Touch touch;
51          if (Input.touchCount < 1 || (touch = Input.GetTouch(0)).phase != TouchPhase.Began)
52          {
53              return;
54          }
55          // Should not handle input if the player is pointing on UI.
56          if (EventSystem.current.IsPointerOverGameObject(touch.fingerId))
57          {
58              return;
59          }
```

Touch 타입의 변수 생성. 터치 횟수가 1 미만 또는 TouchPhase가 Began이 아닌 경우에 작업 스킵. TouchPhase.Began(화면에 터치가 시작된 상태를 의미함.)

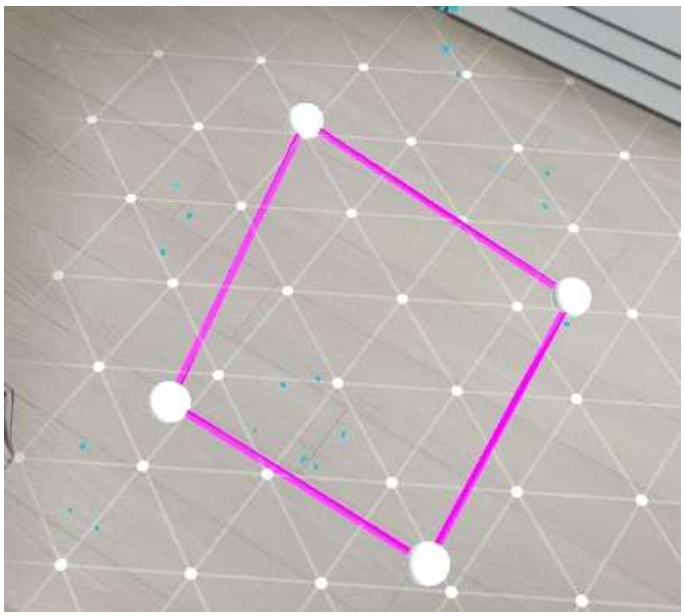
또한, 입력 및 Raycast 등을 담당하는 EventSystem 클래스를 통해 UI를 터치할 경우에도 화면에 포 인트 객체를 생성하지 않을 것이기 때문에, 작업을 건너뜀.



터치 입력에 대한 AR 공간에서의 상호작용을 위해 Raycast를 사용하였음.
기본적으로 Raycast는 화면상에 보이지 않는 직선 광선을 쏘아서 특정 Collider(물리 충돌을 위한 오브젝트)와 충돌하는지 체크하는 방법이다.
이를 바탕으로 AR Core 에서는 사용자의 터치 위치를 기준으로 Raycast 과정에서 광선이 오브젝트와 충돌하는지 체크하게 되는데, 이때 우리가 이전에 카메라를 통해 인식한 바닥면도 일종의 오브젝 트 이므로 광선이 바닥면과의 충돌을 감지 할 수 있게 된다.

```
// Raycast against the location the player touched to search for planes.
TrackableHit hit;
TrackableHitFlags raycastFilter = TrackableHitFlags.PlaneWithinPolygon |
    TrackableHitFlags.FeaturePointWithSurfaceNormal;

if (!isCreated)
{
    if (Frame.Raycast(touch.position.x, touch.position.y, raycastFilter, out hit))
    {
        // Use hit pose and camera pose to check if hittest is from the
        // back of the plane, if it is, no need to create the anchor.
        if ((hit.Trackable is DetectedPlane) &&
            Vector3.Dot(FirstPersonCamera.transform.position - hit.Pose.position,
                hit.Pose.rotation * Vector3.up) < 0)
        {
            Debug.Log("Hit at back of the current DetectedPlane");
        }
        else
        {
            // Instantiate prefab at the hit pose.
            var ob = Instantiate(GameObjectPointPrefab, hit.Pose.position, Quaternion.identity);
            Points.Add(ob);
            positions.Add(ob.transform.position);
```

TrackableHit : AR Core를 통해 추적되는 오브젝트에 대한 Raycast 충돌 정보를 저장함.
사용자가 화면을 터치했을 때, Frame.Raycast를 통해 AR Core를 통해 인식된 바닥 표면으로 Raycast 과정을 거치면, 이에 대한 결과가 앞서 생성한 hit 변수에 저장되고,
이를 hit.Pose.position 등으로 Unity 월드 좌표로 변환하여 이 좌표에 Instantiate 메서드를 통해 포인트 객체를 생성하고 이를 리스트에 담는다.
생성된 포인트 객체들을 바탕으로 Line Renderer를 이용해 각 객체들을 이어주도록 했다.



메쉬 생성

```
37      private List<GameObject> Points = new List<GameObject>();
38      private List<Vector3> positions = new List<Vector3>();
```

이전에 입력 모듈에서 만든 리스트를 활용하여 지정한 포인트 객체의 위치를 기반으로 타일 메쉬를 생성.

```
108     void GenerateMesh()
109     {
110         if (Points.Count >= 4)
111         {
112             Vector3[] vert = new Vector3[positions.Count];
113
114             for (int i = 0; i < Points.Count; i++)
115             {
116                 vert[i].x = positions[i].x;
117                 vert[i].y = positions[i].y;
118                 vert[i].z = positions[i].z;
119             }
120
121             TriangulatorZ tr = new TriangulatorZ(vert);
122             int[] indices = tr.Triangulate();
123
124             Vector3[] vertices = new Vector3[vert.Length];
125             for (int i = 0; i < vertices.Length; i++)
126             {
127                 vertices[i] = new Vector3(vert[i].x, vert[i].y, vert[i].z);
128             }
```

```
129
130             mesh.Clear();
131             mesh.vertices = vertices;
132             mesh.triangles = indices;
133             mesh.uv = tr.CalculateUV();
134             mesh.RecalculateNormals();
135             mesh.RecalculateBounds();
136             mat.mainTextureScale = tr.CalculateScale(1f);
137         }
138     }
```

기존 유니티 커뮤니티 위키에서 제공하는 Triangulator 스크립트는 x축, y축 기준으로 수행하기 때문에, 바닥면에 메쉬를 생성하기 위해서는 x축, z축이 필요.

따라서, 기존 Vector2 타입을 매개변수로 하는 스크립트를 Vector3 타입 매개변수로 수정한 뒤, y축 대신 z축을 대입하여 연산하게 함.

이를 통해 삼각형 모양의 메쉬를 만들기 위한 각 인덱스 값들을 산출하고, 메쉬의 꼭짓점 위치를 Vector3 타입의 배열을 선언하여 넣어줌.

가구 배치

```
/// <summary>
/// Manipulation system allows the user to manipulate virtual objects (select, translate,
/// rotate, scale and elevate) through gestures (tap, drag, twist, swipe).
/// Manipulation system also handles the current selected object and its visualization.
///
/// To enable it add one ManipulationSystem to your scene and one Manipulator as parent of each
/// of your virtual objects.
/// </summary>
28 references
public class ManipulationSystem : MonoBehaviour
{
    private static ManipulationSystem s_Instance = null;

    private DragGestureRecognizer m_DragGestureRecognizer = new DragGestureRecognizer();

    private PinchGestureRecognizer m_PinchGestureRecognizer = new PinchGestureRecognizer();

    private TwoFingerDragGestureRecognizer m_TwoFingerDragGestureRecognizer =
        new TwoFingerDragGestureRecognizer();

    private TapGestureRecognizer m_TapGestureRecognizer = new TapGestureRecognizer();

    private TwistGestureRecognizer m_TwistGestureRecognizer = new TwistGestureRecognizer();
```

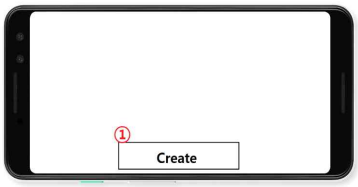
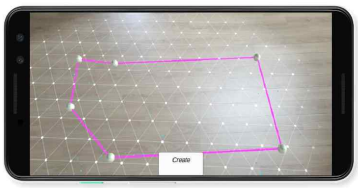
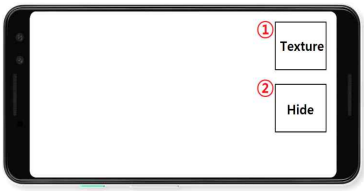
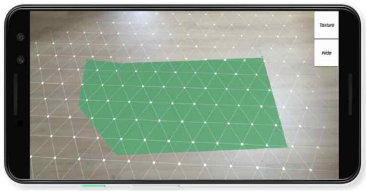
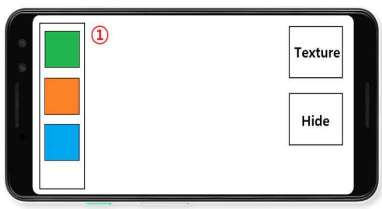
AR Core 라이브러리에서 제공하는 ManipulationSystem을 이용하였다.

ManipulationSystem은 사용자의 다양한 제스쳐들을 받아오기 위해 다양한 변수들을 가지고 있다. 상단에 설명된 것과 같이, 먼저 ManipulationSystem을 씬에 배치하고, AR 공간상에 오브젝트를 생성한 뒤, 그 오브젝트의 부모 인스턴스로 Manipulator를 지정해 주면 된다.

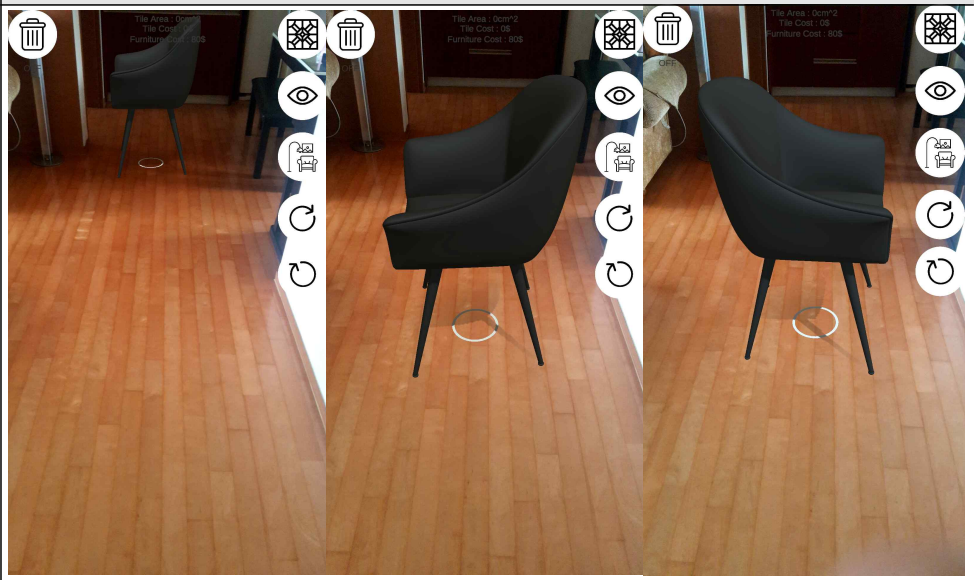
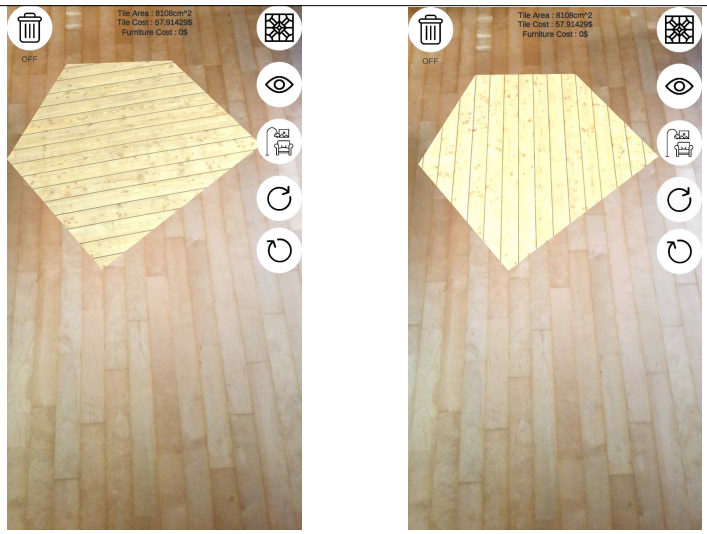
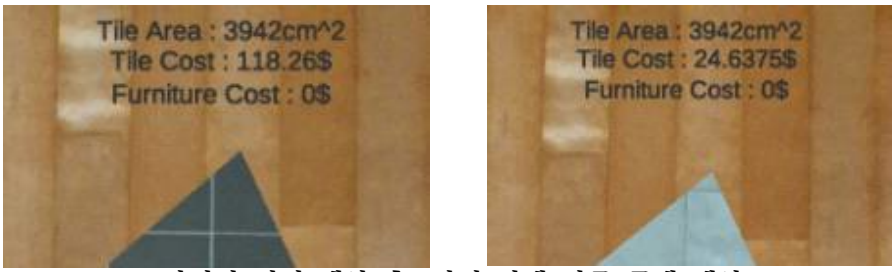
```
if (Frame.Raycast(
    gesture.StartPosition.x, gesture.StartPosition.y, raycastFilter, out hit))
{
    // Use hit pose and camera pose to check if hittest is from the
    // back of the plane, if it is, no need to create the anchor.
    if ((hit.Trackable is DetectedPlane) &&
        Vector3.Dot(FirstPersonCamera.transform.position - hit.Pose.position,
            hit.Pose.rotation * Vector3.up) < 0)
    {
        Debug.Log("Hit at back of the current DetectedPlane");
    }
    else
    {
        // Instantiate game object at the hit pose.
        var gameObject = Instantiate(PawnPrefab, hit.Pose.position, hit.Pose.rotation);

        // Instantiate manipulator.
        var manipulator =
            Instantiate(ManipulatorPrefab, hit.Pose.position, hit.Pose.rotation);

        // Make game object a child of the manipulator.
        gameObject.transform.parent = manipulator.transform;
```

위 스크립트와 같이 먼저 AR 공간에 배치할 오브젝트를 생성, Manipulator 인스턴스도 생성한 뒤에, Manipulator를 오브젝트의 부모로 지정해주었다. 이렇게 함으로써 오브젝트는 Manipulator의 영향을 받아 사용자의 제스쳐에 따라 트랜스폼이 가능하다.

2-5. Prototype 구현

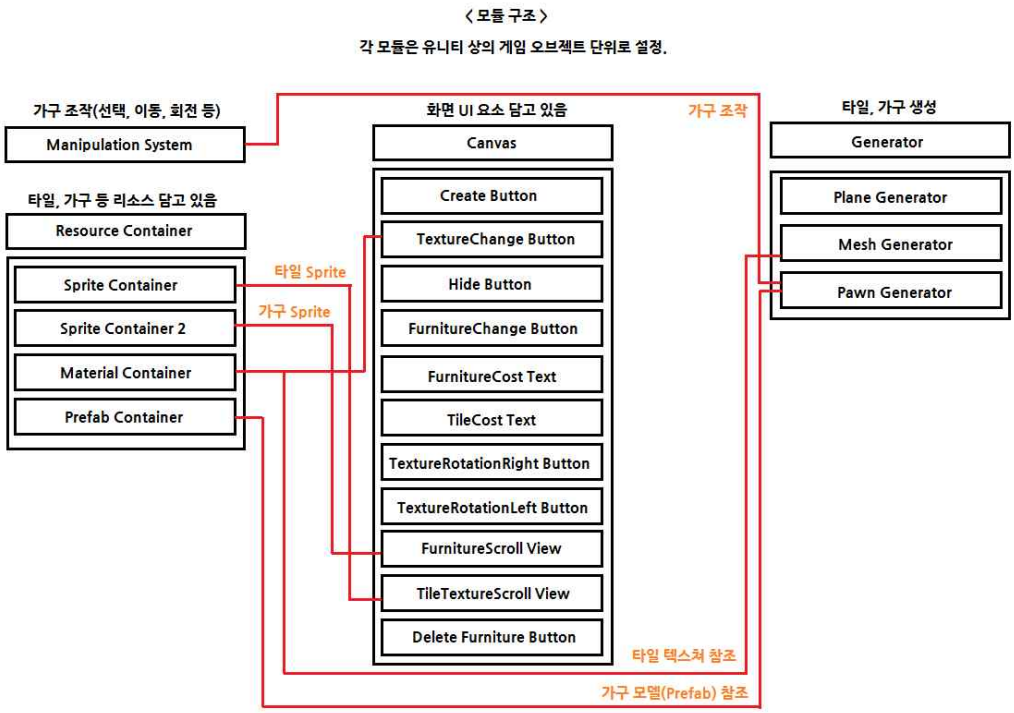
순 서	설 명
초기 화면	  ① Create Button - 메쉬 생성 버튼. - 클릭 시, 이 버튼은 제거되고 다음 화면으로 넘어감.
메쉬 생성 후 화면 (Create 클릭 후)	  ① Texture 버튼 - 메쉬 텍스쳐를 변경하는 버튼. - 클릭 시, 다음 화면 처럼 좌측에 스크롤 바를 생성. ② Hide 버튼 - AR 공간에서 포착된 Plane 시각화를 끄는 버튼.
텍스쳐 변경 화면 (Texture 클릭 후)	  ① Scroll view - 텍스처를 바꾸는 버튼을 포함한 스크롤 바. - 각 버튼 클릭시 생성된 메시의 텍스쳐를 변경.

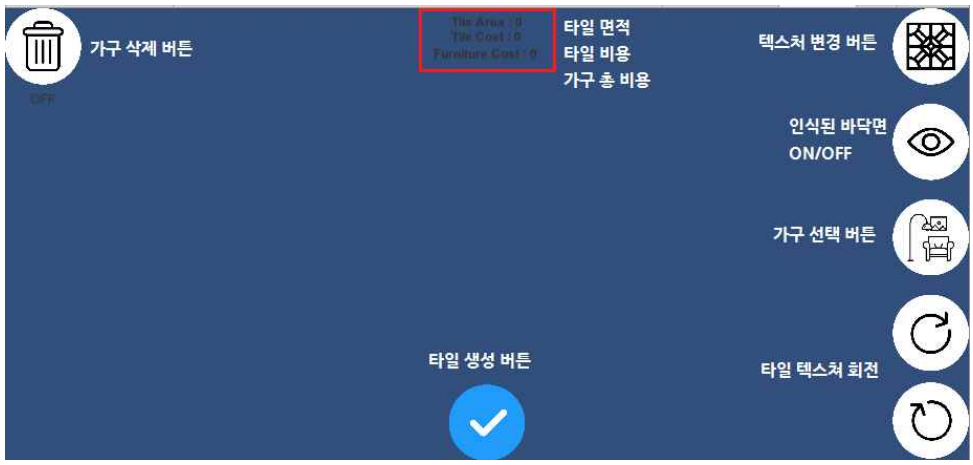
2-6. 시험/테스트 결과 (지적 사항 위주)

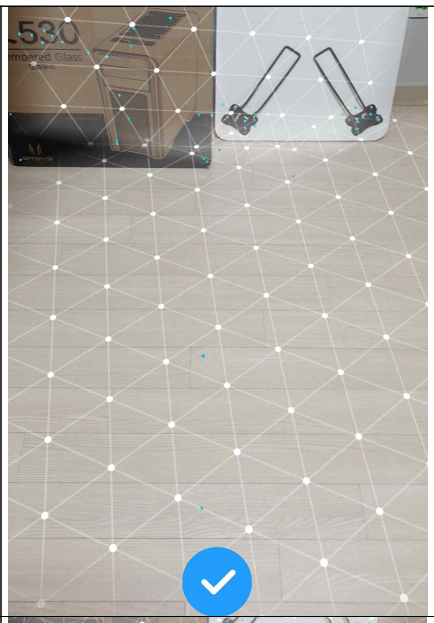
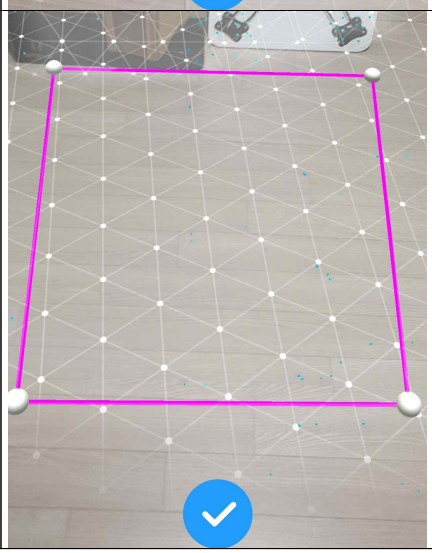
순 서	설 명
가구 생성 후 배치, 회전	 가구 생성 후, 가구 이동, 회전
텍스트 회전	 바닥 축에 맞춰 회전
면적, 금액 계산	 타일의 면적 계산 후, 타일 값에 따른 금액 계산

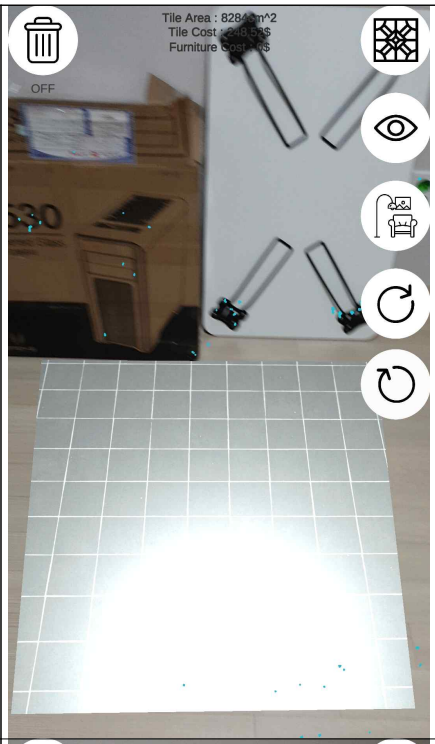
2-7. DEMO

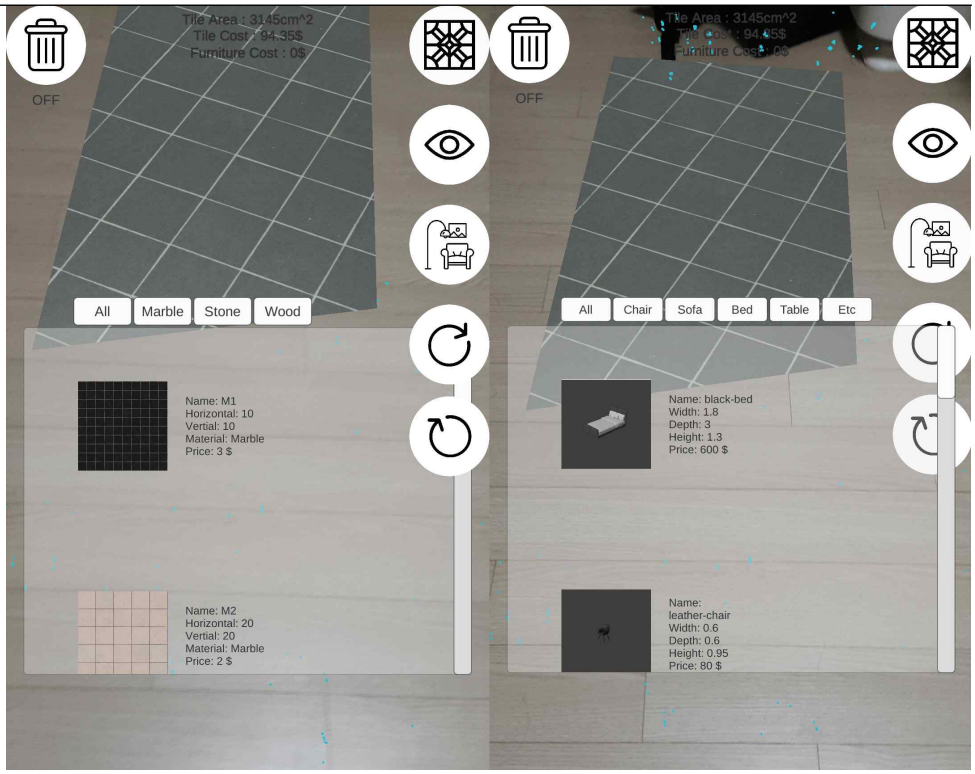
데모 모듈



순 서	설 명
화면 구성	

바닥 인식		Plane Generator 모듈과 안드로이드 디바이스 카메라를 이용, 타일 생성 및 가구 배치를 위한 바닥면을 인식, AR 공간에 생성.
타일 생성을 위한 포인트 객체 생성		Mesh Generator 모듈을 이용, AR 공간에 타일 생성을 위한 포인트 객체를 화면 터치로 생성함.

타일 객체 생성		마찬가지로, Mesh Generator 모듈을 이용, 이전에 만든 포인트 객체를 기준으로 타일 객체를 생성함. 타일 객체 생성을 위해 Create Button을 누르면, 해당 버튼은 사라지면서, 다음 단계에 사용될 버튼들이 Active됨. 생성된 타일 크기와 선택한 타일 기준으로 면적과 비용을 계산하여 화면 상단에 표시함.
가구 선택 및 배치		FurnitureChange 버튼으로 가구를 선택하고, 화면에 배치함. 화면에 배치한 가구들의 가격의 총합을 계산해 이전과 마찬가지로 화면 상단에 표시함.

타일 텍스쳐, 가구 선택 메뉴	 위와 같이, 텍스쳐 변경, 가구 변경 버튼을 누르면 카테고리 별로 정렬된 타일 텍스쳐, 가구 모델들이 표시됨.

3-1. 연구 결과

	내 용
연구 결과 1	■ **기존 연구 : 공간증강현실을 활용한 실내 벽지 인테리어 시스템** • (소개) 증강현실을 활용해 실내 벽지 인테리어 프리뷰 서비스를 제공한다. • (단점) 원하는 **구역을 설정하지 못함.** ■ **개선안** • 인테리어 프리뷰 기술의 관점에서 **원하는 구역을 통제하지 못한다는 건 인테리어에 프리뷰에 도움이 되지 않는다**. 이것을 해결하기 포인터를 설정해 원하는 구역을 통제할 수 있게 한다. ■ **해결방안** • 인식된 바닥 평면으로부터 사용자가 설정한 구역의 정보를 저장한다. • 저장된 정보를 바탕으로 Unity 좌표로 변환하여 이미지가 생성될 구역을 설정한다. • **포인터와 포인터 간의 Line을 사용자가 볼 수 있게 구현해 원하는 구역을 설정할 수 있게 한다.**
연구 결과 2	■ **(기존 연구) 증강현실을 이용한 인테리어 도우미** • (소개) 카메라를 통해 패턴을 검색하여 가상의 공간을 구성하고 DB에 저장된 가상 이미지를 불러와 **가상의 공간에 영상을 송출**한다. • (단점) **특정 패턴**이 있어야만 가상 이미지를 배치시킬 수 있음. ■ **개선안** • 인테리어 프리뷰 서비스는 어디에서든 원하는 곳에 3D 인테리어 객체를 배치할 수 있어야 한다. 기존의 연구는 특정 패턴에 반응해 3D 객체를 생성하기 때문에 특정 패턴이 아닌 원하는 곳에도 3D 객체를 배치할 수 있게 한다. ■ **해결 방안** • 평면 인식을 한 후에 3D 객체를 배치시킨다. 평면을 인식하는 과정에서 **공간에 대한 정보를 얻고 원근감을 이용해 가구를 이동, 회전 시킬 수 있다.**

3-2. 작품 제작 소요재료 목록

	재 료
H·W	• 카메라 탑재 스마트기기 (안드로이드 SDK API lever 24 이상)
S·W	• Unity • Visual Studio

참고 문헌

순번	참고문헌 정보
1	• 김연수, 오해창, 정승주, 이추복, 안서아, 하미란, 최영미, 주무원(2009) 증강현실 기반의 인테리어 설계 시스템에 관한 연구. 한국멀티미디어학회 학술발표논문집 301-303
2	• 박민기, 임규제, 유재덕, 서명국, 정순종, 이관행(2014) 공간증강현실을 활용한 실내 벽지 인테리어 시스템. 한국 컴퓨터그래픽스학회. 117-118
3	• 정성연, 유광상, 김시관, 이해연(2017). 증강현실을 이용한 인테리어 도우미. 한국정보기술학회 340-341
4.	• 절대 강자 유니티 AR/VR, 이재현 저, 위키북스
5	• 만들면서 배우는 유니티 VR 게임 개발. 김광일, 김도윤 저. 한빛미디어
6	• ARCore document • https://developers.google.com/ar/reference
7	• Unity document • https://docs.unity3d.com/Manual/index.html • https://docs.unity3d.com/kr/530/ScriptReference/Mesh-bounds.html • https://docs.unity3d.com/kr/530/ScriptReference/Mesh-triangles.html • https://docs.unity3d.com/kr/530/ScriptReference/Mesh-uv.html
8	• Unity Commuity Wiki (Triangulator) https://wiki.unity3d.com/index.php/Triangulator
9	• Unity Community Answer • https://answers.unity.com/questions/517555/texture-tiling-based-on-object-sizescale.html
10	• Stackoverflow • https://stackoverflow.com/questions/47254092/disable-toggle-visualization-of-tracked-planes-in-arcore-unity