TED UNIVERSITY
CMPE 492
Senior Design Project II
"TEDU GuidAR"
Spatial Computing for Indoor Navigation
Low Level Design Report Version 2

Supervisor: Tolga Kurtuluş Çapın
Jury Members: Fırat Akba, Kasım Murat Karakaya
Course Coordinator: Gökçe Nur Yılmaz

Authors and ID:
Berk Belhan 43906121950
Alperen Karadağ 14317165222
Altuğ Berke Akman 15349016582
Ceren Kızılırmak 14125057252

Project Page URL: https://berkbelhan.github.io/ar-navigation-senior-project/

Table of Contents

1. Introduction

Indoor environments such as university campuses, museums, airports often create navigation difficulties because of complex layouts, multi-floor structures, and insufficient directional guidance. To address this problem, this project proposes an Augmented Reality (AR) and mobile Indoor Navigation System, named GuidAR, which aims to provide accurate and intuitive indoor navigation support in real time.

The system is being developed in Unity and uses MultiSet for room scanning and mapping of indoor spaces. GuidAR uses the spatial understanding obtained from scanned environments together with device-based tracking and localization mechanisms to determine the user's position within the building. Based on this, the system generates navigation routes and overlays virtual guidance elements such as arrows, labels, and directional indicators onto the user's view. GuidAR is designed to improve spatial awareness, reduce user confusion, and provide a more interactive navigation experience in indoor environments.

This report describes the low-level architecture and the design of the GuidAR. Report comprises Object design trade-offs, Engineering standards, Packages and Class interfaces sections. Finally, the report is concluded with the class diagrams and the detailed explanations of software components.

1.1 Object design trade-offs

1.1.1. Efficiency vs Accuracy

In our project, one of the main design decisions is how the environment is mapped. Although mapping the entire university would be ideal, we observed that large-scale mapping leads to less stable and less accurate results. As the environment becomes more complex, the mapping system must process more data, which introduces noise and reduces tracking accuracy. To avoid this, we focus on smaller and controlled areas such as specific floors or corridors. Within these areas, we define key points of interest (POIs) like doors, classrooms, and laboratories. This approach prioritizes accuracy over full coverage. While the system does not represent the entire campus, it provides more stable tracking and more precise and accurate localization within the selected areas.

1.1.2. Usability vs Functionality

Usability is a critical factor in the design of the GuidAR system due to the limitations of the Meta Quest 3 interface. Since the user interacts with the system through a headset, displaying too many elements at once can create visual clutter and reduce clarity. To prevent this, the system does not present all available features or information simultaneously. Instead, it

provides only the most relevant content based on the user's current position, such as nearby points of interest (POIs) or directional cues. This approach improves usability by keeping the interface simple and easy to follow. However, it limits the amount of functionality and information that can be accessed at a given time.

1.1.3. Security vs Usability

The system uses the Meta Quest 3 camera and sensors continuously to track the user and understand the environment. This means that visual data from the real environment is constantly processed during the experience. To protect user privacy, all camera and sensor data are handled locally on the device and are not stored or transmitted to external systems. This prevents sensitive information about the environment from being shared. However, this decision limits some usability features. For example, maps, points of interest (POIs), and content cannot be updated dynamically from a remote server and must be predefined in the system. As a result, the system becomes more secure, but less flexible in terms of updates and content management.

1.1.4. Reliability vs. Compatibility

The system is designed to run reliably on Meta Quest 3 by using its built-in tracking, sensors, and AR capabilities. Focusing on a single device allows more stable localization, consistent AR visualization, and smoother interaction during the experience. However, this decision reduces compatibility with other platforms. Since the system depends on Meta Quest 3–specific features such as spatial tracking and gesture-based interaction, it cannot be directly used on mobile devices or other AR headsets without additional modifications.

1.2 Interface documentation guidelines

In this report, class interfaces are documented in a consistent format as shown below. Class, attribute, and method names follow the camelCase convention. Class names start with a capital letter, while attributes and methods start with lowercase letters. Each class is described with a brief explanation of its role in the system, followed by methods. This structure helps clearly present how system components work.

class className
Explanation of the Class
Methods

methodName 1
methodName 2

1.3 Engineering standards

UML guidelines are used in this report to represent class interfaces, system structure, and component interactions. Class diagrams are used to show relationships between core components such as tracking, content management, and AR visualization, helping to clearly explain the system design. The report also follows IEEE-style standards for referencing and citation to ensure a consistent and structured presentation. In addition, basic object-oriented design principles such as modularity and separation of concerns are considered. The system is divided into components like tracking, content handling, and user interaction, which simplifies development and allows the system to be extended more easily, for example by adding new areas or points of interest.

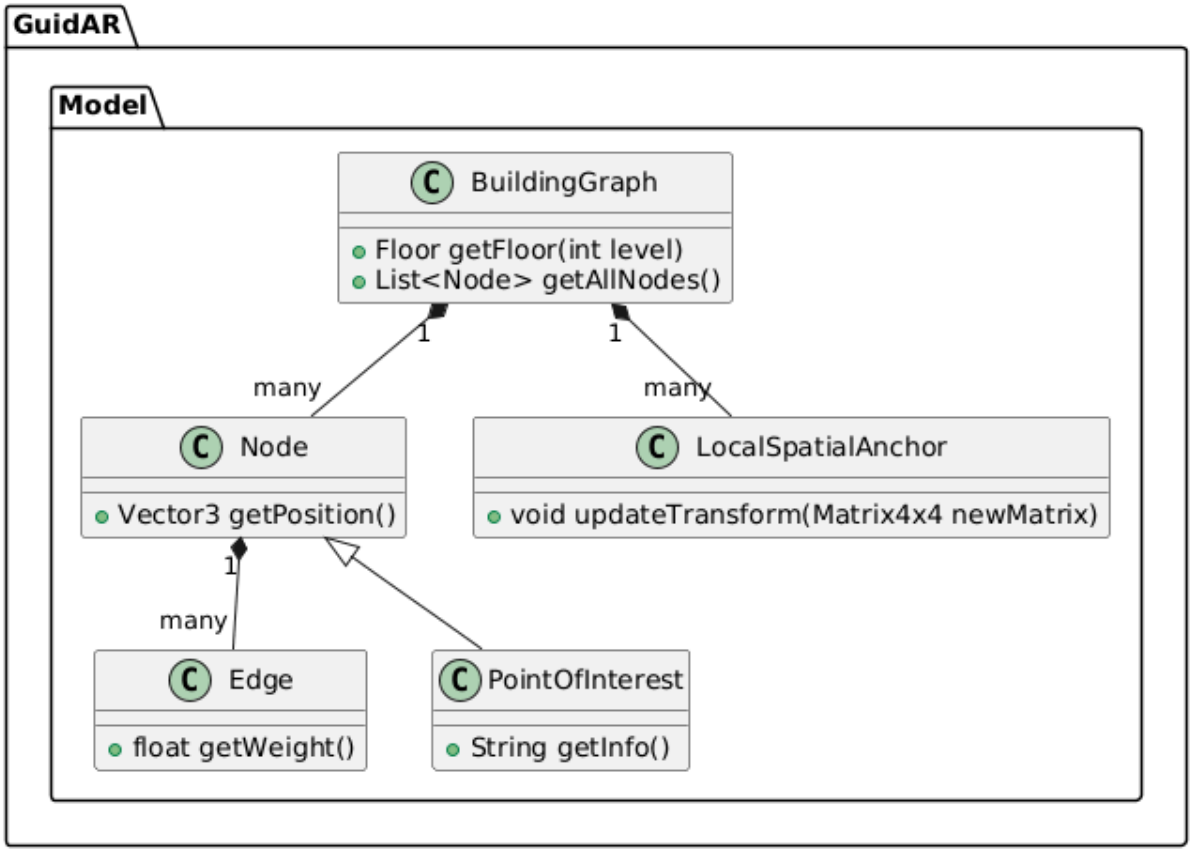
1.4 Definitions, acronyms, and abbreviations

- **API:** Application Programming Interface
- **AR:** Augmented Reality
- **POI:** Point of Interest
- **UI:** User Interface
- **UML:** Unified Modeling Language
- **SDK:** Software Development Kit
- **HUD:** Head-Up Display
- **MVC:** Model-View-Controller
- **IMU:** Inertial Measurement Unit
- **XR:** Extended Reality

2. Packages

The GuidAR system is designed around a local-first Model-View-Controller (MVC) architecture to meet the high-performance and low-latency requirements of real-time indoor navigation. By storing the entire building graph, Point of Interest (POI) metadata, and spatial anchor transforms directly on the device's internal storage, the system eliminates the risks of network instability and external server latency common in complex environments.

2.1 Model



The Model package represents the "Source of Truth" for the navigation environment. It is stored in the Meta Quest 3's persistent local storage as serialized data (e.g., JSON or ScriptableObjects).

BuildingGraph: This class acts as the primary data structure for the campus. It contains a collection of all floor levels and the topological connections between them.

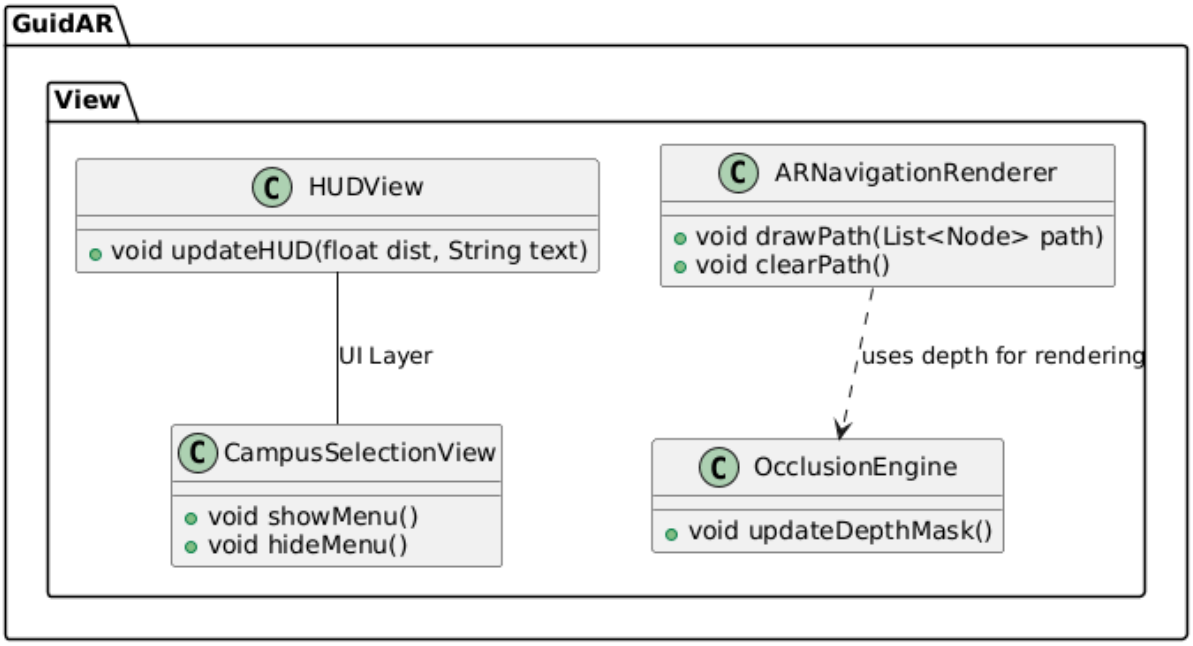
Node: Represents a discrete point in the physical environment (e.g., a room entrance or a corridor junction) with 3D coordinates relative to the building's origin.

Edge: Defines a navigable path between two Nodes. It stores "cost" metadata such as distance and type (e.g., "staircase," "elevator," or "hallway") to assist pathfinding.

PointOfInterest (POI): An extension of the Node class that stores searchable metadata for campus locations, such as room numbers, staff names, and office hours.

LocalSpatialAnchor: Manages the local persistent data for Meta Spatial Anchors. It stores the transformation matrices required to align virtual navigation cues with the physical room geometry without external cloud "resolving."

2.2 View



The View package is responsible for all visual output presented to the user through the Quest 3's passthrough lenses.

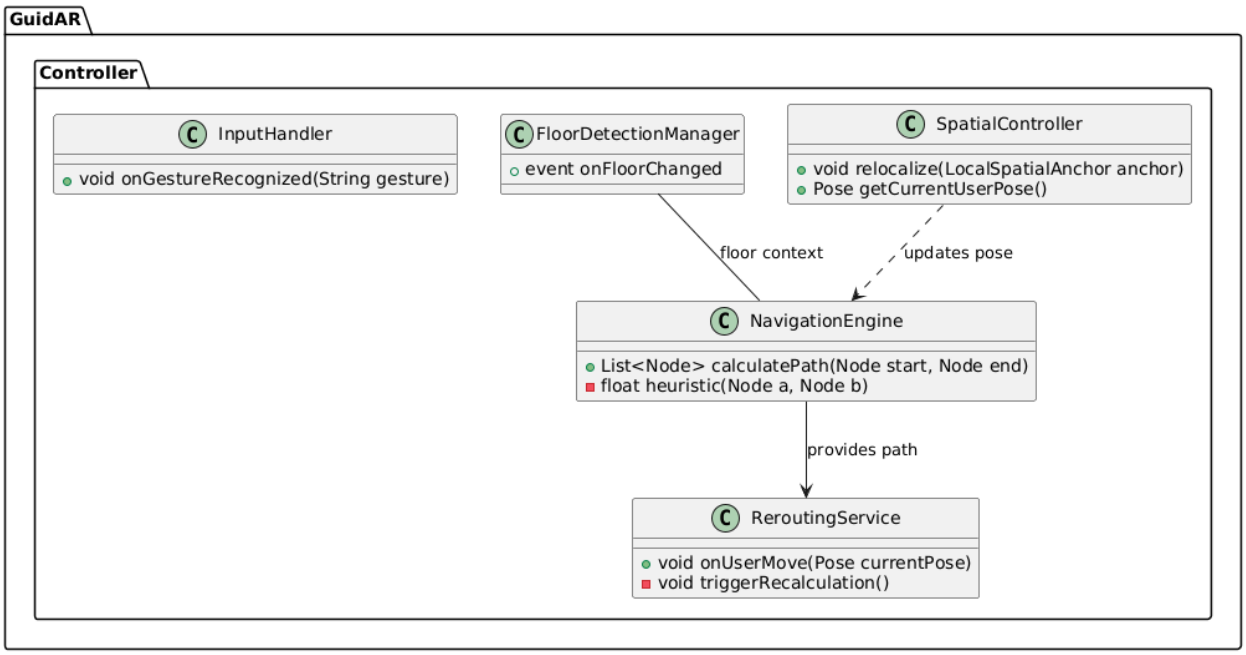
ARNavigationRenderer: Handles the real-time instantiation and placement of 3D navigation elements (arrows, path lines, and pins). It ensures these elements are "bolted" to the physical floor.

OcclusionEngine: Interfaces with the Meta Quest 3 Depth API to render a depth mask, allowing virtual arrows to be hidden by real-world objects like pillars, desks, or walls for a more realistic experience.

UserInterface: A heads-up display that provides constant feedback, such as the name of the next destination, distance remaining, and instructional text (e.g., "Turn Right at the stairs").

DestinationSelectionView: A world-space floating menu that allows users to browse the local POI database and select their desired destination.

2.3 Controller



The Controller package manages the application state and acts as the intermediary between the Meta Quest 3 hardware sensors and the static Data Model. It deals with high-frequency sensor fusion and the pathfinding logic.

SpatialController: The primary interface with the Meta Presence Platform. It handles the "Relocalization" process by matching real-time sensor features against the stored LocalSpatialAnchor data.

NavigationEngine: Implements the A* pathfinding algorithm locally. It queries the BuildingGraph to determine the most efficient route based on the user's current coordinates.

ReroutingService: A background observer that monitors the user's distance from the planned path. If the user deviates beyond a defined threshold, it automatically calls the NavigationEngine to update the route.

FloorDetectionManager: Processes data from the headset's IMU and barometer to identify vertical transitions. It triggers the loading of the appropriate floor segment from the local BuildingGraph.

InputHandler: Manages user interaction via the Meta XR Interaction SDK, translating hand-tracking gestures into commands for destination selection or map exploration.

3. Class Interfaces

class NavigationManager
It starts and stops navigation sessions, receives the selected destination, and requests route calculation from the path planning component. It also updates the current route according to the user's position and triggers rerouting when necessary.
Methods
StartNavigation()
StopNavigation()

class PathPlanner
Calculates the best route between the current position and the selected destination. It uses the navigation graph and pathfinding logic such as A*.
Methods
FindPath(startNode, endNode)
RecalculatePath()

class POIManager
Loads classrooms and offices.
Methods
GetPOIByName(string name)
GetAllPOIs()

class UIManager
It manages the user interface elements of the application. It displays destination options, navigation information
Methods
ShowDestinationList()
UpdateDistanceText()
ShowArrivalMessage()

3.1 Navigation Mash Scripts

startPoint property

This property stores the starting point of the NavMesh link as a Vector3.
The get part returns the current start point value.
The set part updates the value only if the new value is different from the current one. If the value changes, UpdateLink() is called so the NavMesh link is refreshed with the new start position.

```
public Vector3 startPoint
{
    get => m_StartPoint;
    set
    {
        if (value == m_StartPoint)
            return;

        m_StartPoint = value;
        UpdateLink();
    }
}
```

endPoint property

This property stores the ending point of the NavMesh link as a Vector3.
The get part returns the current end point.
The set part checks whether the new value is different from the old one. If it is changed, the new value is assigned and UpdateLink() is called to update the link.

```
public Vector3 endPoint
{
    get => m_EndPoint;
    set
    {
        if (value == m_EndPoint)
            return;

        m_EndPoint = value;
        UpdateLink();
    }
}
```

costModifier property

This property controls the traversal cost of the NavMesh link.
The get part returns the cost value, but its sign depends on whether cost overriding is enabled.
The set part checks the given value and decides whether the cost should override the default cost. It stores the absolute value of the cost and then calls UpdateLink() so the new movement cost is applied.

```
public float costModifier
{
    get => m_IsOverridingCost ? m_CostModifier : -m_CostModifier;
    set
    {
        var shouldOverride = value >= 0f;
        if (value.Equals(costModifier) && shouldOverride == m_IsOverridingCost)
            return;

        m_IsOverridingCost = shouldOverride;
        m_CostModifier = Mathf.Abs(value);
        UpdateLink();
    }
}
```

GetLocalPositions(...) method

This method calculates the start and end positions in local space.
It checks whether the start and end points are defined directly as local positions or attached to

transforms.
If transforms are used, their world positions are converted into the local coordinate system. This helps the system work correctly whether the link points are fixed manually or connected to moving objects.

```
internal void GetLocalPositions(
    out Vector3 localStartPosition,
    out Vector3 localEndPosition)
{
    var startIsLocal = m_StartTransform == null;
    var endIsLocal = m_EndTransform == null;
    var toLocal = startIsLocal && endIsLocal ? Matrix4x4.identity : LocalToWorldUnscaled().inverse;

    localStartPosition = startIsLocal ? m_StartPoint : toLocal.MultiplyPoint3x4(m_StartTransform.position);
    localEndPosition = endIsLocal ? m_EndPoint : toLocal.MultiplyPoint3x4(m_EndTransform.position);
}
```

POI Class

POI.cs

Library > PackageCache > com.multiset.sdk.quest@df62690d9e81 > Runtime > Scripts > Navigation > DataModel > POI.cs

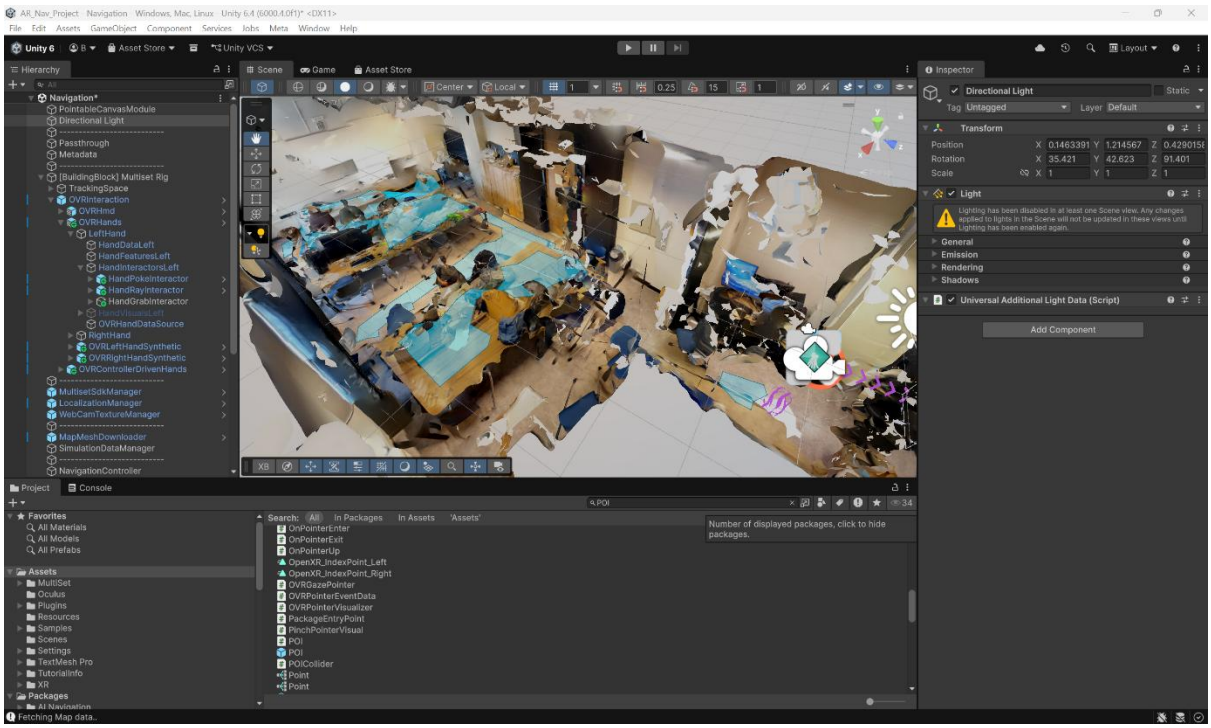
```
using UnityEngine;

public class POI : ListItemData
{
    // unique identifier for this POI
    int id;

    [Tooltip("Identification for this POI, e.g. room number")]
    public int identification;

    [Tooltip("Name of this POI")]
    public string poiName;

    [Tooltip("Description for this POI")]
    public string description;

    [Tooltip("Type for this POI")]
    public POIType type;

    [Tooltip("Collider for path calculation and user arrival detection. Place pivot point near NavMesh. Collider should be near NavMe
    public POICollider poiCollider;

    [Tooltip("Visual representation for this POI")]
    public POISign sign;

    void Awake()
    {
        base.listTitle = poiName;
        id = identification; // this can be adapted if you get id from external source
        sign.SetPOI(this);
        poiCollider.SetPOI(this);
    }

    // returns this id
    public int GetId()
    {
        return id;
    }

    // Handles arrival of user at POI
    public void Arrived()
    {
        if (NavigationController.instance.currentDestination != null && NavigationController.instance.currentDestination.GetId() == i
        {
            // arrived at the selected POI
            NavigationController.instance.ArrivedAtDestination();
        }
    }
```

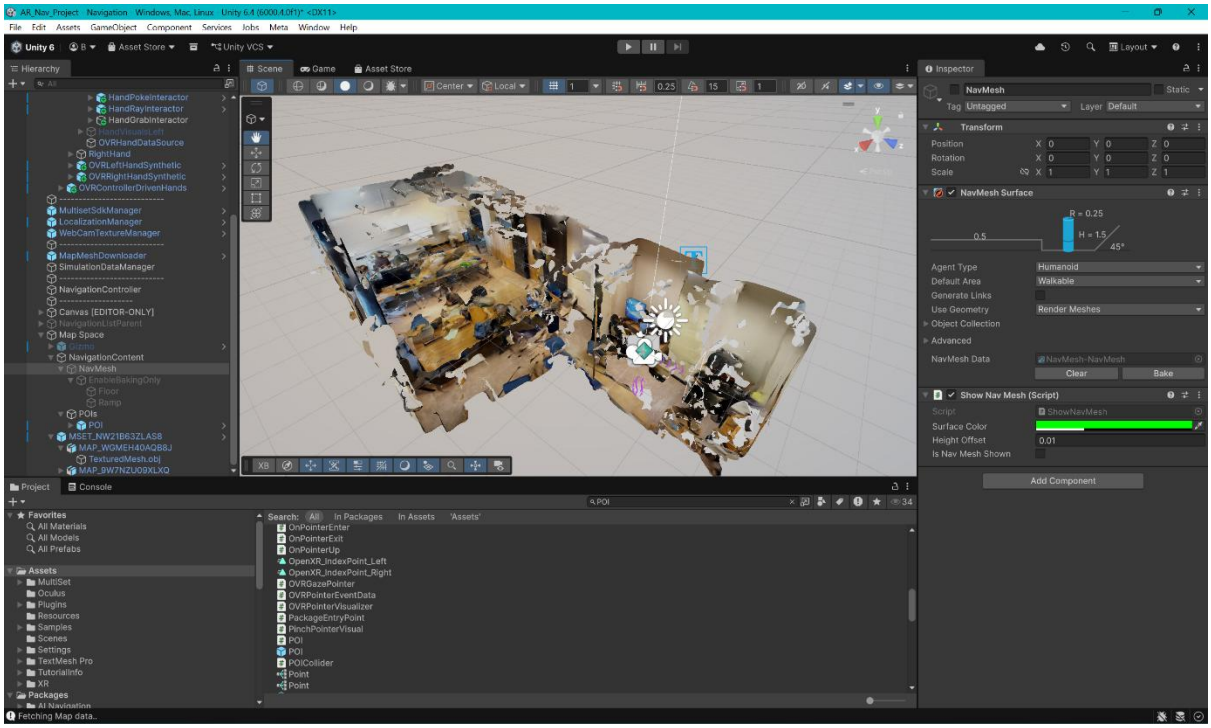
4. Mapping

The spatial representation in GuidAR is built upon a Navigation Mesh (NavMesh), which defines the traversable areas within the campus environment. Unlike a simple point-to-point graph, the NavMesh allows for more fluid user movement and better handling of physical obstacles in real-time.

The mapping system treats the floor as a collection of convex polygons. This structure is chosen for its computational efficiency in local pathfinding.

- **Surface Discretization:** Using the Quest 3's Scene Mesh, the system identifies horizontal planes to generate a high-fidelity NavMesh.
- **Agent Constraints:** The NavMesh is baked with specific agent parameters (e.g., human shoulder width and step height) to ensure the generated paths do not lead users too close to walls or through non-traversable furniture.





5. References

[1] IBM, "UML - Basics," June 2003. [Online]. Available: http://www.ibm.com/developerworks/rational/library/769.html. [Accessed 17-Mar-2026].

[2] IEEE, "IEEE Citation Reference," September 2009. [Online]. Available: https://m.ieee.org/documents/ieeecitationref.pdf. [Accessed 17-Mar-2026]

Volpis. (2025, October 23). A complete guide to developing augmented reality indoor navigation applications. [Online]. Available: https://volpis.com/blog/guide-to-developingaugmented- reality-indoor-navigation-applications/ [Accessed 16-Mar-2026]

[3] Google ARCore Documentation, Geospatial and Indoor Mapping APIs. [Accessed 17-Mar-2026]

[4] Apple ARKit Documentation, Visual-Inertial Odometry and Room Plan APIs. [Accessed 17-Mar-2026]

[5] Sukhareva, E., Tomchinskaya, T., & Serov, I. (2021). SLAM-based indoor navigation in university buildings. https://ceur-ws.org/Vol-3027/paper63.pdf [Accessed 17-Mar-2026]

[6] Meta Horizon Platform SDK Documentation. Meta for Developers. [Online]. Available: https://developers.meta.com/horizon/documentation/unity/ps-platform-intro/ [Accessed 17-Mar-2026]

[7] Parab, D. K., Deshpande, P. P., & Khanvilkar, S. (2024). Indoor navigation system using augmented reality. International Congress on Engineering and Computer Science. Semantic Scholar. [Accessed 17-Mar-2026]

[8] Vaqar, M. (2024). Augmented reality-based navigation system for indoor environments. International Journal of Research in Engineering, 6(2), 60–65. [Online]. Available: https://doi.org/10.33545/26648776.2024.v6.i2a.110 [Accessed 17-Mar-2026]

[9] Alluhaidan, A. S., et al. (2025). From GPS to AR: Leveraging augmented reality and grid-based systems for improved indoor navigation. IEEE Access, 13, 55210–55225. [Online]. Available: https://doi.org/10.1109/ACCESS.2025.10930457 [Accessed 15-Mar-2026]

[10] Zheng, L., & Wang, H. (2024). A review of research on SLAM technology based on the fusion of LiDAR and vision. Sensors, 25(5), 1447. https://doi.org/10.3390/s25051447 [Accessed 16-Mar-2026]

[11] Lokhande, J., et al. (2025). Campus navigation using augmented reality and virtual reality. International Journal for Multidisciplinary Research, 7(6). [Online]. Available: https://www.ijfmr.com/research-paper.php?id=19504 [Accessed 16-Mar-2026]