Implementation of a context-sensitive augmented reality application

Seminar Thesis

David Philipp Diener
2396965

At the Department of Economics and Management
at the Institute of Information Systems and Marketing (IISM)
Information & Market Engineering

Reviewer:	Marius Schenkluhn
Second reviewer:	Dr. Christian Peukert

31st of August 2022

Contents

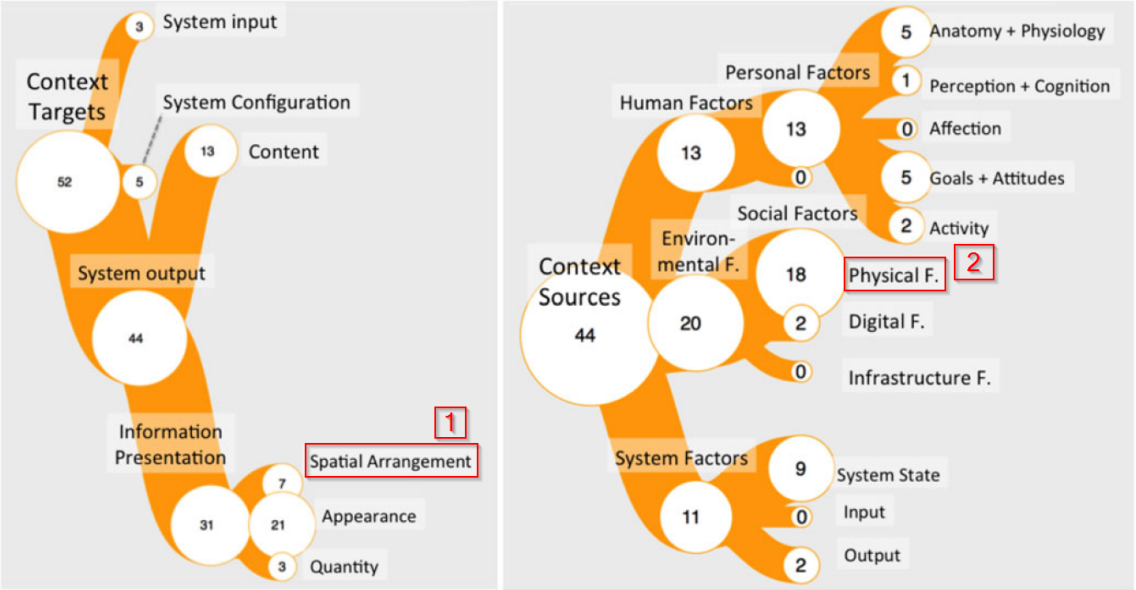
1. Introduction

In recent years Extended Reality (XR) Technology is on the rise, supporting more and more devices. We want to find ways to improve certain aspects of on XR Technology - Augmented Reality (AR). Supporting context awareness, can greatly enhance user experience in AR applications, for example by adjusting to the individual needs of each user (Yigitbas, Jovanovikj, Sauer, and Engels (2020))

While there is a number of conceptual works and system papers (where the state of the implementation appears unclear), complex user or context models are rarely developed Grubert, Langlotz, Zollmann, and Regenbrecht (2017). This is why we want to contribute to current research by proposing a way of implementing AR applications which support context awareness.

This paper will show the practical implementation of two aspects of context sensitivity according to the Taxonomy, shown in 1.1.

Figure 1.1: Taxonomy proposed by Grubert et al. (2017)



2. Aspects of Context-sensitivity

The implementation proposed in this paper shows the application of the context-sensitivity categories “Spatial Arrangement” and “Physical factors”.

Spatial Arrangement deals with the way, external information is presented inside the AR application. Information Presentation is described by Grubert et al. (2017) as “adapting the interface or the visualization used, while some other are less explicit how the context information is applied to the AR interface”. In this implementation, we show the application of spatial arrangement by creating a surface detection system.

Physical factors are part of the Environmental Factors. Environmental factors describes the surrounding of the user and the AR system in which interaction takes place, i.e., external physical and technical factors that are not under control of the mobile AR system or the user Grubert et al. (2017). In this implementation, we show the application of physical factors by creating a real-time 3D Navigation system on top of the spatial / geometric configuration of the physical world.

A navigation system, working with physical factors in AR was also proposed by Mulloni, Seichter, and Schmalstieg (n.d.). It is described as an “[...] interface that provides continuous navigational support for indoor scenarios where localization is only available at sparse, discrete locations (info points).” This solution utilized info points which could be scanned and would be used by the system to query the next navigation step. Although this solution uses physical factors, it does not use the geometric configuration of the surroundings to create the navigation in real time. The steps are hard-coded into the systems and only retrieved when scanning the info points.

3. Implementation

The following chapter describes the implementation details of the two systems "Surface-Detection" and "3D-Navigation". Beforehand, the overall structure and technology stack used in the application is introduced.

Unity Traditionally used as a game engine. Unity includes a XR framework, which facilitates the implementation of basic XR functionalities.

Mixed Reality Toolkit (MRTK) The MRTK is a project, developed by Microsoft. It contains utility functions which can be used when developing MR applications. One of the most important features we use here, are Spatial Awareness and Scene Understanding. Spatial Awareness can be used to scan and capture the spatial geometry around the device. We can then use the Scene Understanding API to get the semantic meaning of those objects.

3.1 Surface-Detection

Surface Detection relies on the MRTK Scene Understanding API. We request the active Windows Scene Understanding Observer (WSUO) in the Scene. This object hold all information about the present objects in the spatial environment at runtime. Iterating through those objects allows for assigning every object into a layer, depending on the assigned surface type by the Scene Understanding service (see 6.1). The Scene Understanding surface types are therefore assigned to a walkable and obstacle Unity Layer:

- Walkable: Floor
- Obstacle: Wall, Platform, Background, Inferred

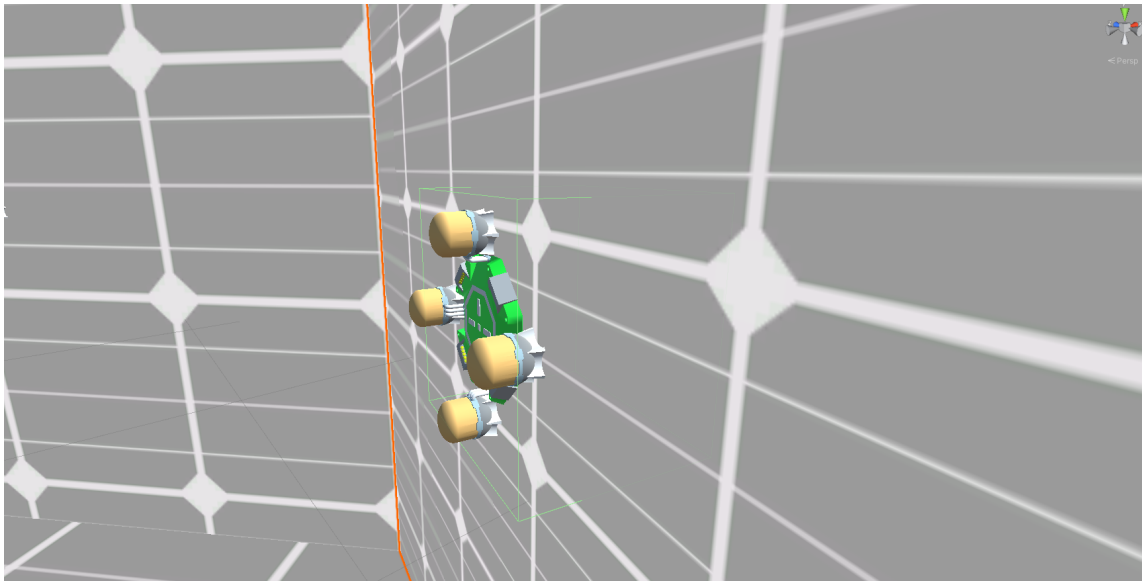
To enable placable objects to detect those surfaces now (with Unity's Physics System), the following conditions have to be met:

- Both participants of a collision need to have a collider component.
- At least one of the participants of a collision needs to have a rigidbody component
- The Layer Collision Matrix for Unity Physics has to be configured to allow the collision between the layers of the participants of the collision

All objects provided by the WSUO are by default already instantiated in the scene and come with a collider by default. The objects which will be placed in the context of surface detection therefore need to have a collider, as well as a rigidbody. Figure 3.1 shows a correctly configured game object, colliding with a wall object.

The surface detection itself takes place by defining the according trigger events for the collider components. By using the events OnTriggerEnter (see 6.2) and OnTriggerExit (see 6.3) Unity (2022) each objects tracks its own "placeability" state. If a collision in OnTriggerEnter is detected, the layer of the object hit is compared with a preconfigured list of allowed surfaces. If this check is positive the object can be placed. OnTriggerExit reverts the placeability accordingly.

Figure 3.1: A game object colliding with a wall on the obstacle layer



3.2 3D-Navigation

The 3D-Navigation system relies on the same layer structure as defined in 3.1. Additionally a third party library, Mercuna Mercuna (2022) is used.

To setup a scene for 3D Navigation with Mercuna, three components have to be configured:

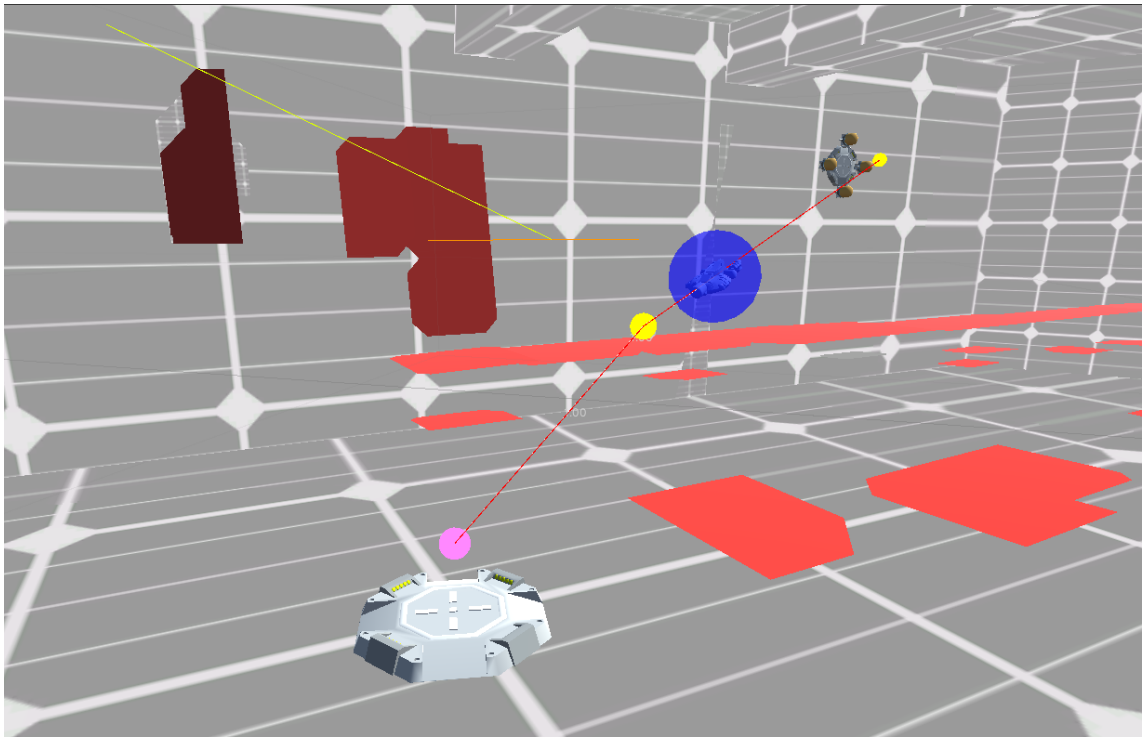
- Nav Octree: This game object holds the 3-Dimensional data structure representing the spatial geometry provided by the WSUO. The previously configured layers Walkable and Obstacle are set to be included in the geometry of the octree.
- Nav Volume: Defines the maximum bound of the octree.
- Nav Seed: Starting point on where Mercuna "floods" the octree structure to generate the navigation space.

To setup an agent for 3D Navigation with Mercuna, four components have to be configured:

- Mercuna Move Controller, Mercuna Obstacle, Mercuna 3D Navigation: The default components provided by Mercuna to ensure basic movement functionality.
- AstarAI3D: Custom script which uses Mercuna 3D Navigation as a middleware to simulate a patrol behavior (see 6.4).

Figure 3.2 shows a properly configured agent navigating on the Mercuna octree.

Figure 3.2: An agent (marked blue) navigating the octree structure in realtime.



4. Limitations

Cold Start Problem The two systems that are introduced in this paper heavily rely on the information about scene objects, provided by the WSUO. WSUO in return depends on scanning the spatial environment with the help of a scanner. Therefore surface detection and 3D-Navigation will only be possible after the XR-Headset has successfully scanned the environment. Alternatively a cached model of the environment could be used to solve the cold start problem.

Scanning Details This project was developed and tested on the HoloLens 2. The spatial scanning provided by this device turned out to be limited in the level of detail. For example, narrow geometric objects like table legs could not be reliably detected.

5. Future Work

The surface detection could be improved by checking other spatial features, like space needed on the target surface.

Especially in the context on mobile platforms like the HoloLens 2 the performance should be considered, when developing computational intensive applications. The main reasons for this are the maintenance of a stable framerate, while also keeping the battery usage of the device as small as possible. An analysis on the performance impact of different amounts of agents and parallel calculation of the paths could lead to insights on performance optimization.

6. Declaration

Ich versichere hiermit wahrheitsgemäß, die Arbeit selbständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderung entnommen wurde.

Karlsruhe, den 21. 08. 2022 _____

David Philipp Diener

Appendix

Listing 6.1: Assigning layers to Objects provieded by the SceneUnderstanding API

```
foreach (SpatialAwarenessSceneObject obj in sceneObserver.
    SceneObjects.Values)
    {
        // Walkable
        if (walkableSurfaces.HasFlag(obj.SurfaceType))
        {
            // set layers
            obj.GameObject.layer = LayerMask.NameToLayer("
                Walkable");
            foreach (Transform child in obj.GameObject.transform)
            {
                child.gameObject.layer = LayerMask.NameToLayer("
                    Walkable");
            }
            Debug.Log("Set " + obj.Id + " from " + obj.GameObject
                .layer + " to Walkable");
            // save last floor height
            floorHeight = obj.GameObject.transform.position.y;
        }

        // Obstacles
        if (obstacleSurfaces.HasFlag(obj.SurfaceType)) {
            // set layers
            obj.GameObject.layer = LayerMask.NameToLayer("
                Obstacles");
            foreach (Transform child in obj.GameObject.transform)
            {
                child.gameObject.layer = LayerMask.NameToLayer("
                    Obstacles");
            }
            Debug.Log("Set " + obj.Id + " from " + obj.GameObject
                .layer + " to Obstacles");
        }
    }
```

Listing 6.2: OnTriggerEnter

```
private void OnTriggerEnter(Collider other)
{
    if ((placeableLayerMask.value & (1 << other.transform.
        gameObject.layer)) > 0)
```

```
    {
        placable = true;
        meshRenderer.material = validMaterial;
    }
    else
    {
        placable = false;
        meshRenderer.material = errorMaterial;
    }
}
```

Listing 6.3: OnTriggerExit

```
private void OnTriggerExit(Collider other)
{
    if ((placeableLayerMask.value & (1 << other.transform.
       gameObject.layer)) > 0)
    {
        grounded = false;
        placable = false;
        meshRenderer.material = errorMaterial;
    } else
    {
        if(grounded)
        {
            placable = true;
            meshRenderer.material = validMaterial;
        }
    }
}
```

Listing 6.4: MonoBehaviour class to handle patrol behaviour of a Mercuna agent

```
public class AstarAI3D : MonoBehaviour{
    public float minDistance;

    private bool isAtHome = true;
    private Transform currentTarget;

    [HideInInspector] public Transform targetPosition;
    [HideInInspector] public Transform homePosition;

    void Start()
    {
        currentTarget = targetPosition;
```

```
        GetComponent<Mercuna3DNavigation>().NavigateToObject(
            currentTarget.gameObject, OnMoveComplete, minDistance)
            ;
    }

    private void OnMoveComplete(bool success)
    {
        if (isAtHome)
        {
            isAtHome = false;
            currentTarget = homePosition;
        }
        else
        {
            isAtHome = true;
            currentTarget = targetPosition;
        }

        GetComponent<Mercuna3DNavigation>().NavigateToObject(
            currentTarget.gameObject, OnMoveComplete, minDistance)
            ;
    }
}
```

References

Grubert, J., Langlotz, T., Zollmann, S., & Regenbrecht, H. (2017). Towards pervasive augmented reality: Context-awareness in augmented reality. *IEEE Transactions on Visualization and Computer Graphics*, *23*(6), 1706–1724. doi: 10.1109/TVCG.2016.2543720

Mercuna. (2022). *Mercuna - 3d navigation for games.* Retrieved from `https://mercuna.com/3d-navigation/`

Mulloni, A., Seichter, H., & Schmalstieg, D. (n.d.). Indoor navigation with mixed reality world-in-miniature views and sparse localization on mobile devices. In (pp. 212–215). doi: 10.1145/2254556.2254595

Unity. (2022). *Unity - scripting api: Collider.* Retrieved from `https://docs.unity3d.com/ScriptReference/Collider.html`

Yigitbas, E., Jovanovikj, I., Sauer, S., & Engels, G. (2020). On the development of context-aware augmented reality applications. In J. Abdelnour Nocera et al. (Eds.), *Beyond interactions* (pp. 107–120). Cham: Springer International Publishing.