3GPP TSG-SA WG4 Meeting #133-eS4-251353r01

Online, 18th July 2025 - 25th Jul 2025

**Source: Nokia**

**Title: [FS\_ARSpatial] pCR on spatial computing functions**

**Spec: 3GPP TR 26.819**

**Agenda item: 9.7**

**Document for: Agreement**

**1. Introduction**

This contribution proposes to add information related to 3D model retrieval based on spatial regions of interest.

**2. Reason for Change**

Clause 4.1.5 describes the 3D model reconstruction as a spatial computing function and outlines the potential need for significant computing power and bandwidth for retrieval of 3D models. However, it only describes the input data needed for 3D model reconstruction, omitting control data required for requesting a spatial region of interest or a Level of Detail (LoD) of the world mesh.

This functionality is enabled in OpenXR using the [XR\_MSFT\_scene\_understanding](https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html#XR_MSFT_scene_understanding) extension. An application can pass one or more bounding volumes when calling [xrComputeNewSceneMSFT](https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html#xrComputeNewSceneMSFT). These bounding volumes are used to determine which scene components to include in the resulting scene. Scene components that intersect one or more of the bounding volumes should be included, and all other scene components should be excluded. The bounding volumes can be indicated in the form of an oriented box, sphere or frustum.

In addition to a spatial region of interest, applications can also indicate a desired Level of Detail (LoD) for the mesh, specifiying the desired number of triangles per volume. The OpenXR extension [XR\_MSFT\_scene\_understanding](https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html#XR_MSFT_scene_understanding) specifies four possible levels: coarse, medium, fine and unlimited.

**3. Proposal**

It is proposed to agree the following changes to 3GPP TR 26.819 v1.0.0.

\* \* \* First Change \* \* \* \*

# 4 Introduction to spatial computing

## 4.1 Spatial computing functions

### 4.1.1 General

XR spatial computing encompasses a set of functions which process sensor data to generate information about the world 3D space surrounding the AR user. These include functions such as relocalization (establishing the position of users and objects within that space), anchoring, 3D Model reconstruction, segmentation and labeling (semantic perception), light extraction, and real object removal, which are achieved through a combination of advanced sensors, cameras, and algorithms that enable devices to understand and interact with the three-dimensional space around them.

This requires accurately localizing the AR device worn by the end-user in relation to a spatial coordinate system of the real-world space. Vision-based localization systems reconstruct a sparse spatial mapping of the real-world space in parallel (e.g., SLAM). Beyond the localization within a world coordinate system, which is usually based on a sparse spatial map, dense spatial mapping of objects is also essential in order to place 3D objects on real surfaces and provides the ability to occlude objects behind surfaces, do physics-based interactions based on surface properties, provide navigation functions, or provide a visualization of the surface.

For the purpose of understanding and perceiving the scene semantically, machine-learning and/or artificial intelligence may be used to provide context for the observed scene.

The output of spatial computing is spatial mapping information that is organized in a data structure called the *XR Spatial Description* for storing and exchanging the information. Some spatial computing functions may also take an XR Spatial Description and may result in updates to the XR Spatial Description. Spatial computing functions typically include data exchange and require a network architecture.

An AR device may provide sensor data to a spatial computing function to create or update the spatial mapping information. The device may also access a spatial computing function to retrieve different spatial mapping information depending on the needs of the XR application.

The spatial computing functions can run locally on the AR device or can be executed remotely on the cloud or on the edge in a spatial computing server accessed through dedicated interfaces as detailed in TR 26.298 [2].

The main functions provided by a spatial computing service are given in Figure 4.1.1-1 and explained in the following subclauses.

A diagram of a model

AI-generated content may be incorrect.

**Figure 4.1.1-1: Spatial computing functions**

\* \* \* Second Change \* \* \* \*

### 4.1.5 3D model reconstruction and retrieval

Spatial computing enables the creation of accurate 3D models of surrounding space. It accurately captures real-world scenes and objects using 3D scanning techniques or photogrammetry. These 3D models can be displayed in immersive 3D environments in real-time to provide users with a sense of interactivity and presence. When built in real-time, they can, for example, be used to manage occlusion and physics behaviors between real and virtual objects. Such 3D models can also be used offline for authoring AR experiences, and for completing or correcting a digital twin of a real environment.

The 3D model of a real-world environment may also be constructed collectively by aggregating meshes captured by an AR device.

HoloLens (Microsoft) and AR SDKs from Apple and Meta can all build a 3D model of the surrounding environment in real-time.

Achieving highly realistic and accurate 3D model reconstruction can require significant computing capabilities, especially when dealing with multiple users capturing the same space, and when using advanced structure-from-motion algorithms or radiance-field-based representations, such as Gaussian splatting. This can be an issue for some devices. These algorithms can then be executed offline on remote servers using sensor data provided by one or multiple AR devices, with the reconstructed models sent back upon completion.

To build the 3D model, the following input data can be used:

* Sensor data:
* Images captured by AR Device
* pose of AR Device
* Depth map (image or texture)
* Mesh captured by AR Device

The output of the 3D Model construction is a 3D Model. A 3D model is a broader term that encompasses the complete representation of a 3D object, including its geometry (which may be represented by a mesh), texture, and materials.

When using a remote computing server, streaming all these data from several AR client can require a high bandwidth. As an example, considering an RGB image and depth map with a resolution of 1280×720 compressed in JPEG format, and an uncompressed mesh composed of 200000 vertices (encoded as three 32-bit floating point coordinates per vertex), this would require approximately a bandwidth of 1 Gbps. This corresponds to the requirements of the Magic Leap AR Cloud platform [62] (mentioning between 480 Mbps and 1 Gbps).

To optimize bandwidth usage and reduce its processing load, an XR device can request a subset of the 3D model of the surrounding space from the server. The desired subset can be represented as a bounding volume that indicates the spatial region of interest for which the XR device wishes to receive spatial mapping data. For example, this functionality is enabled in OpenXR using the [XR\_MSFT\_scene\_understanding](https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html" \l "XR_MSFT_scene_understanding) extension. An application can pass one or more bounding volumes when calling [xrComputeNewSceneMSFT](https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html" \l "xrComputeNewSceneMSFT). These bounding volumes are used to determine which scene components to include in the resulting scene. Scene components that intersect one or more of the bounding volumes are included, and all other scene components are excluded. The bounding volumes can be indicated in the form of an oriented box, sphere or frustum.In addition to a spatial region of interest, applications can also indicate a desired Level of Detail (LoD) for the mesh, specifiying the desired number of triangles per volume. The OpenXR extension [XR\_MSFT\_scene\_understanding](https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html" \l "XR_MSFT_scene_understanding) specifies four possible levels: coarse, medium, fine and unlimited.

#### 4.1.5.1 Example: HoloLens (Microsoft)

HoloLens [9] has built-in cameras that continuously scan the environment, allowing it to construct virtual world geometry for real-world objects. Figure 4.1.5.1-1 demonstrates examples of 3D models constructed by HoloLens.

A person wearing virtual reality goggles

Description automatically generatedA green grid of lines

Description automatically generated

Figure 4.1.5.1-1: HoloLens 3D Model construction and spatial mapping (from Unity)

In HoloLens, spatial surfaces are represented by dense triangle meshes. To minimize the required processing and storage resources, applications can enable a mesh caching scheme appropriate for their needs. This scheme determines which meshes to keep, which to discard, and when to update the mesh for each spatial surface.

An application can define the regions of space in which the application wishes to receive spatial mapping data by defining bounding volumes. It is recommended in the documentation that new spatial surfaces and updated spatial surfaces near and in front of the user are given higher priority by the application when spatial data is requested.

HoloLens also enables visualization of the spatial mesh to give the user an understanding of the detected spatial surfaces. An application can use this feature, for example, to help the user when the user is trying to place a virtual object onto a surface.

More details on spatial mapping in HoloLens is provided in [9].

#### 4.1.5.2 Example: Snapdragon Spaces (Qualcomm)

Spatial meshing enables real-time creation of a 3D map of the surrounding environment, allowing applications to calculate spatial relationships, e.g., to avoid obstacles, and achieve realistic occlusions. Developers can specify mesh quality and update intervals for efficient processing.

|  |
| --- |
| // Initialize meshing  MeshManager meshManager = new MeshManager();  meshManager.StartMeshing(MeshQuality.High, updateInterval: 1.0f);  // Retrieve mesh data in update loop  foreach (var mesh in meshManager.CurrentMeshes)  {      Vector3[] vertices = mesh.Vertices;      int[] indices = mesh.Indices;      Debug.Log($"Mesh - Vertices: {vertices.Length}, Indices: {indices.Length}");  } |

In this code snippet, *StartMeshing* initiates the meshing process with specified quality and update intervals. The *CurrentMeshes* property provides access to the latest mesh data, including arrays of *Vertices* and *Indices* for each mesh segment, which can be used for rendering or spatial analysis in the application.

#### 4.1.5.3 Example: Magic Leap AR Cloud

AR Cloud [62] is a connected service that enables Magic Leap 2 devices to access extensive spatial data. It allows digital twin administrators to create Shared Spaces and manage device-side scans through a web-based console. A key feature of AR Cloud is its ability to merge scans from different users into a unified 3D model reconstruction accessible from each client. The service can run on the cloud on providers such as AWS, GCP, and Azure.

\* \* \* Third Change \* \* \* \*

### 4.1.11 Summary of spatial description formats

This section describes the common output data formats for the spatial computing functions defined in clause 4.2. Table 4.1.11-1 provides a list of the output data for each spatial computing function and the corresponding format.

Table 4.1.11-1: Output data of spatial computing functions.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function** | **Data** | **Description** | **Format Example** | **Possible Representation** |
| World Tracking | Feature Map | Features to compute a pose in XR Space. | Array of features |  |
| Feature | 3D point associated with a descriptor, the descriptor format depends on the feature type. | SIFT (Scale Invariant Feature Transform) | Typically, a vector of 128 floats. |
| ORB – Oriented FAST Rotated BRIEF (Binary descriptor) | Typically, a 32-bit descriptor. |
| BRIEF – Binary Robust Independent Elementary Features) | Typically, 32-bit descriptor. |
| FREAK – Fast Retina Keypoint | Typically, 64-bit descriptor. |
| SURF – Speeded-Up Robust Features | Typically, vector of 64-bit or 128-bit floats. |
| Relocalization | Pose | Pose of the AR device in XR Space | Position: 3D vector  Orientation: quaternion | Position: 3 floating point values.  Oriantation: 4 floating point values |
| XR\_space\_id | An identifier for the XR Space used as a reference. | String |  |
| Anchoring | Id | An identifier of the new trackable (when the anchoring function is invoked by a producer). | String |  |
| Pose | Pose of the anchor in relation to the device according to a trackable (when the anchoring function is invoked by a consumer). | Position: 3D vector  Orientation: quaternion | Position: 3 floating point values.  Oriantation: 4 floating point values |
| 3D Model Reconstruction | Model | 3D model(s) of surrounding space. | Non-segmented model (e.g., point cloud, mesh with/without attributes) | OBJ, STL, PLY [41], FBX, glTF |
| Segmented model | glTF, USD |
| Bounding volume | Spatial region of interest indicating desired subset of the 3D model | Oriented box, sphere, frustum |  |
| Level of Detail | Desired level of detail for the 3D model (e.g. number of triangles per volume) | Enum: coarse, medium, fine |  |
| Semantic Perception | Objects | A list of 3D objects resulting from the segmentation and labeling function. | Array or a hierarchical node graph. |  |
| Object | A 3D object in the captured world. |  |  |
| Mesh | Segmented object. | Mesh |  |
| Label | Label of segmented object. | String |  |
| Semantic representation | Representation of objects with semantic relationships (e.g., adjacency, containment) | Graph of objects (Nodes represent objects; edges represent spatial or semantic relationship as textual description) | Graph, List, Array of textual description |
| Collider Generation | Colliders | Set of colliders (not combined with associated objects). | Array of colliders |  |
| Collider | A collider object. | Primitive (e.g., sphere with radius) |  |
| Mesh | OBJ, STL, FBX |
| Colliders | Part of description of the 3D Model (when combined with associated objects). | Hierarchical node graph | MPEG-I SD, USD [46] |
| Segmentation and Labeling | Objects | A list of 3D objects resulting from the segmentation and labeling function. | Array or a hierarchical node graph. |  |
| Object | A 3D object in the captured world. |  |  |
| Mesh | Segmented object. | Mesh |  |
| Label | Label of segmented object. | String |  |
| Light Extraction | Lights | Set of extracted lights | Array |  |
| Light | A description of a light source that includes a set of parameters that depend on the type of the light. Possible types include: point, directional, area, spot, texture-based, or image-based light. | Object | For point light, the parameters include: pose, intensity, color, and range. |
| Real Object Removal | Objects | A set of textured 3D objects. | Mesh + texture | OBJ, STL, PLY, FBX, glTF |
| Images | Images with transparency. |  | PNG, TIFF, TGA, BMP |

\* \* \* End of Changes \* \* \* \*