**3GPP TSG-SA WG4 Meeting #130S4-242020r01**

**Orlando, Florida, USA, 18 – 22 November 2024** *revision of S4aV240067*

**Source: InterDigital Canada**

**Title: [FS\_ARSpatial] pCR on Spatial Description Formats**

**Spec: 3GPP TR 26.819**

**Agenda item: 9.9**

**Document for: Agreement**

**1. Introduction**

The Study on Spatial Computing for AR Services [FS\_ARSpatial] was approved during SA#104 meeting. The core spatial computing functions have been documented in TR 26.819 in the SA4-129e meeting. Spatial computing includes functions such as world tracking, relocalization, anchoring, 3D model construction, collider generation, segmentation and label, light extraction. The resulting output of spatial computing is a set of spatial mapping information that is organized in a data structure called the XR Spatial Description for storing and exchanging the information.

**2. Reason for Change**

Define the format for the output of each of the identified functions in TR 26.819 in a spatial computing service and relevant spatial description information.

**3. Proposal**

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

\* \* \* Begin Changes \* \* \*

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

# 2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non‑specific.

- For a specific reference, subsequent revisions do not apply.

- For a non-specific reference, the latest version applies. In the case of a reference to a 3GPP document (including a GSM document), a non-specific reference implicitly refers to the latest version of that document *in the same Release as the present document*.

[1] 3GPP TR 21.905: "Vocabulary for 3GPP Specifications".

[2] 3GPP TR 26.928: "Extended Reality (XR) in 5G".

[3] 3GPP TR 26.998: "Support of 5G glass-type Augmented Reality / Mixed Reality (AR/MR) devices".

[4] ARCore SLAM, https://developers.google.com/ar/develop/fundamentals

[5] ARKit VIO, https://developer.apple.com/documentation/arkit/arkit\_in\_ios/configuration\_objects/understanding\_world\_tracking

[6] ARCore Cloud Anchor, <https://codelabs.developers.google.com/codelabs/arcore-cloud-anchors#0>

[7] ARKit World Map, <https://developer.apple.com/documentation/arkit/arworldmap>

[8] Meta Quest Spatial Anchor, <https://developer.oculus.com/documentation/unity/unity-spatial-anchors-overview/>

[9] HoloLens, https://learn.microsoft.com/en-us/windows/mixed-reality/design/spatial-mapping

[10] Meta Space Setup, <https://www.uploadvr.com/quest-v64-undocumented-features-furniture-recognition-multimodal/>

[11] Apple RoomPlan, <https://developer.apple.com/augmented-reality/roomplan/>

[12] Google Scene Semantics, <https://developers.google.com/ar/develop/scene-semantics>

[13] Unity Documentation - Collision, <https://docs.unity3d.com/2023.1/Documentation/Manual/collision-section.html>

[14] Blender Documentation – Collisions, https://docs.blender.org/manual/en/latest/physics/rigid\_body/properties/collisions.html

[15] Unity Documentation - Compound Colliders, <https://docs.unity3d.com/Manual/compound-colliders-introduction.html>

[16] Unity Documentation - Mesh Colliders, <https://docs.unity.cn/Manual/mesh-colliders-introduction.html>.

[17] ARCore Ligthing Estimation, <https://developers.google.com/ar/develop/lighting-estimation>

[18] Cast Shadow, <https://ieeexplore.ieee.org/document/9018202>

[19] Macario Barros, A., Michel, M., Moline, Y., Corre, G., & Carrel, F. (2022). A comprehensive survey of visual slam algorithms. Robotics, 11(1), 24.

[x1] SIFT, <https://www.vlfeat.org/api/sift.html>

[x2] E. Rublee, V. Rabaud, K. Konolige and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," 2011 International Conference on Computer Vision, Barcelona, Spain, 2011, pp. 2564-2571.

[x3] A. Alahi, R. Ortiz and P. Vandergheynst, "FREAK: Fast Retina Keypoint," 2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, 2012, pp. 510-517.

[x4] M. Calonder, V. Lepetit, M. Ozuysal, T. Trzcinski, C. Strecha and P. Fua, "BRIEF: Computing a Local Binary Descriptor Very Fast," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 34, no. 7, pp. 1281-1298, July 2012.

[x5] OBJ, https://en.wikipedia.org/wiki/Wavefront\_.obj\_file

[x6] PLY, <http://gamma.cs.unc.edu/POWERPLANT/papers/ply.pdf>

[x7] FBX, <https://www.autodesk.com/products/fbx/overview>

[x8] STL, <https://docs.fileformat.com/cad/stl/>

[x9] glTF, https://registry.khronos.org/glTF/specs/2.0/glTF-2.0.pdf

[x10] USD, <https://openusd.org/dev/api/class_usd_object.html>

[x11] ITMF, <https://www.immersivealliance.org/download/download-itmf/>

[x12] OpenUSD, <https://openusd.org/release/api/usd_lux_page_front.html>

[x13] Unity, <https://docs.unity3d.com/Manual/mesh-colliders-introduction.html>

[x14] X3D, <https://www.web3d.org/documents/specifications/19775-1/V3.3/>

[x15] Visual Feature Descriptors, <https://www.ipb.uni-bonn.de/html/teaching/photo12-2021/2021-pho1-11-features-descriptors.pptx.pdf>

[x16] ARF, <https://forge.etsi.org/rep/arf/arf005/-/blob/1.0.0/API/openapi.yaml>

[x17] ISO/IEC 23090-14, Text of ISO/IEC FDIS 23090-14 2nd edition Scene description, April 2024.

[x18] FAST, <http://www.edwardrosten.com/work/rosten_2006_machine.pdf>

[x19] Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool, Speeded-Up Robust Features (SURF), Computer Vision and Image Understanding, Vol. 110, Issue 3, 2008, pp. 346-359.

[x20] OpenGL Textures, https://registry.khronos.org/OpenGL-Refpages/gl4/html/texture.xhtml

[x] <doctype> <#>[ ([up to and including]{yyyy[-mm]|V<a[.b[.c]]>}[onwards])]: "<Title>".

\* \* \* Next Change \* \* \*

# Definitions of terms, symbols and abbreviations

# Terms

For the purposes of the present document, the terms given in TR 21.905 [1] and the following apply. A term defined in the present document takes precedence over the definition of the same term, if any, in TR 21.905 [1].

**spatial computing:** AR functions which process sensor data to generate information about the world 3D space surrounding the AR user.

**spatial mapping:** The process of mapping real-world surfaces into the virtual world.

**XR spatial description:** a data structure describing the spatial mapping of the real-world using anchors, trackables, camera parameters and visual features.

**XR Space**: as defined in TS 26.119.

**anchor:** as defined in TS 26.119.

**trackable:** as defined in TS 26.119.

\* \* \* Next Change \* \* \*

### 4.2.2 World tracking

Simultaneous Localization and Mapping (SLAM) is a technology that allows an AR device to create a map of its environment while simultaneously determining its own location within that map. The goal of SLAM is to obtain a global, consistent estimate of the camera path. The map of the environment is usually kept for helping localization. SLAM achieves its objectives using two parts: (1) a tracking mechanism, which solves the localization problem by comparing the new input data with the currently existing map, and (2) a mapping mechanism, which supports the tracking by providing, maintaining and expanding the map itself based on the information obtained from the input data obtained by the tracking part. Loop detection is the process of identifying when a device has returned to a previously visited location. When a loop is detected, this information is used to reduce the drift in both the map and pose. Loop closure is the process of correcting the device’s pose and map based on the detected loop. This involves adjusting the device’s trajectory and the map to ensure consistency and eliminate drift. Beside localization, loop detection and loop closure are two main challenges in SLAM.

The visual-based approaches can be divided into three main categories: visual-only SLAM, visual-inertial SLAM, and RGB-D SLAM [19].

Visual-only SLAM refers to the techniques based only on 2D images provided by a monocular or stereo camera. This type of SLAM uses images captured from one or more cameras to detect features in the environment, track their movements across frames, and estimate the motion of the camera and the structure of the environment. The primary challenge in visual-only SLAM is dealing with scenarios where visual information is poor, such as in low-light conditions or in textureless environments. It is also susceptible to accumulating drift over time since it only relies on visual data. Robustness of the algorithms can be increased by adding information from an inertial measurement unit (IMU). Visual-inertial SLAM techniques combine data from cameras and inertial sensors (accelerometers and gyroscopes) to estimate the pose of the device. RGB-D SLAM techniques employ a depth sensor and leverage depth data alongside visual data, which simplifies the mapping process but can be limited by the operational environment of the depth sensors (more suitable to indoor environments).

Another algorithm used for world tracking is the visual inertial odometry which combines information from the device’s motion sensing hardware and computer vision analysis of the scene visible to the device’s camera. The process recognizes notable features in the scene image, tracks differences in the positions of those features across video frames and compares that information with motion sensing data. The result is a high-precision model of the device’s position and motion.

The main difference between visual SLAM and visual odometry lies in considering, or not, the global consistency of the estimated trajectory and map. While visual odometry performs only local optimizations, visual-SLAM algorithms also employ loop closure detection [19].

By aligning the pose of the virtual camera that renders the 3D content with the pose of the device's camera, virtual content can be rendered from the correct perspective. The rendered virtual image can be overlaid on top of the image obtained from the device's camera, making it appear as if the virtual content is part of the real world.

As the AR device moves through the world, SLAM is used to understand where the device is relative to the world around it. The process detects visually distinct features in the captured camera image called *feature points (=feature map)* and uses these points to compute a change in location. The visual information is combined with inertial measurements from the device's IMU to estimate the pose (i.e., position and orientation) of the camera relative to the world over time.

ARCore supports world tracking using visual-inertial SLAM [4], while ARKit uses Visual Inertial Odometry [5].

For world tracking, the following input data may be used:

- Sensor data:

* Images captured by AR Device
* IMU data
* pose of AR Device
* Depth maps

The output of the World Tracking function is a set of features (Feature Map). Features are extracted from images and described using descriptor. Common feature detection algorithms include: the features from accelerated segment test (FAST) [x18] algorithm, the speeded-up robust features (SURF) [x19] algorithm, the oriented FAST and rotated BRIEF (ORB) [x2] algorithm, and the scale invariant feature transform (SIFT) [x1] algorithm.

Common feature descriptors include:

* SIFT – Scale Invariant Feature Transform ([x1])
* ORB – Oriented FAST Rotated BRIEF (Binary descriptor) ([x2])
* FREAK – Fast Retina Keypoint (Binary descriptor) ([x3]]
* BRIEF – Binary Robust Independent Elementary Features (Binary descriptor) ([x4])
* SURF – Speeded-Up Robust Features [x19]

\* \* \* Next Change \* \* \*

### 4.2.3 Relocalization

Relocalization is a function that is used to estimate the pose of the AR device at initialization, when tracking is lost, or regularly to correct the drift of the tracking (TR 26.998 [3] clause 4.2.3). Cameras capture the real world, while sensors (accelerometers, gyroscopes, and depth sensors) contribute additional data for mapping and positioning. Computer vision algorithms process this data to determine the location and orientation of the device relative to its environment.

SLAM, Visual Localization, e.g., Visual SLAM (vSLAM), or Visual Positioning System (VPS) are all algorithms that can be used for mapping unknown environments while also maintaining the localization of the device/user within that environment, as explained in TR 26.928 [2] clause 4.1.4.

For relocalization, the following input data can be used:

- Sensor data:

- images (for SLAM)

- local pose of AR Device

The output of the relocalization function is a pose, which is defined by a position and an orientation.

\* \* \* Next Change \* \* \*

### 4.2.4 Anchoring

Spatial Anchors are a concept in Augmented Reality that enable the persistence and stability of virtual content in the real world. Virtual objects or information are anchored to specific locations in physical space.

A spatial anchor acts as a marker or reference point in the real world that AR devices can recognize and track. These anchors serve as fixed reference points that AR devices can detect and use to position virtual content accurately relative to the real-world coordinates.

Spatial anchors are created using spatial mapping techniques, which involve capturing and analyzing the physical features of the surrounding environment. This can be done through depth-sensing cameras, LiDAR scanners, or other sensors to understand the geometry and spatial characteristics of the space.

Once the spatial mapping is performed, spatial anchors are placed at desired locations within the mapped environment. Anchor can be persistent for the entire duration of the session.

The three main AR SDK (Meta, Google and Apple) offer anchoring solution:

- ARCore, Cloud Anchor [6]

- ARKit, World Map [7]

- Meta Quest, Spatial Anchor [8]

For anchoring, the following input data can be used:

- Anchor pose for creation

- Sensor data:

- images

- pose of AR device

- LiDAR data

The Anchoring function provides several services:

* Creation: when a new trackable is created, information are extracted from trackable, an anchor is created with a relative pose to the trackable. A unique identifier of the newly created trackable is returned.
* Consumption: from a trackable ID, the function retrieves the stored trackable and all their information. Sensor data and the features of the trackable are used to compute the pose of the anchor(s) associated with the trackable. When the trackable is detected, the anchors are positioned, and their pose is returned.
* Synchronization: This function enables spatial anchors to be synchronized across multiple AR devices, allowing for shared AR experiences, e.g. for collaborative applications where multiple users interact with the same virtual content.
* Persistence: This function enables spatial anchors to be saved and retrieved across sessions. This means that virtual objects can remain in the same physical location even after the AR application is closed and reopened.

\* \* \* Next Change \* \* \*

### 4.2.5 3D model construction

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.

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.

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

#### 4.2.5.1 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 2 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 4.2.5.1-1: HoloLens 3D Model construction and spatial mapping (from Unity)

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.

### 4.2.6 Segmentation and labelling

3D Semantic Segmentation is a task that involves dividing a 3D point cloud or 3D mesh into semantically meaningful parts or regions. The goal of 3D semantic segmentation is to identify and label different objects and parts within a 3D scene. This is similar to semantic image segmentation where sections of an image are separated into clusters of pixels relating to corresponding objects, with respective classifications.

The three main AR SDKs (Meta, Google and Apple) support segmentation and labeling solutions. However, the domain in which these functions are performed (i.e., 2D vs. 3D) may differ from one SDK to another.

The output of the segmentation and labeling function is a 3D model that is composed of a number of object models and associated labels. The segmentation function can be part of the 3D model construction function. In this case, the 3D model construction function may directly output a segmented and labeled 3D model.

If the 3D model generated by the 3D model construction function is segmented and labeled, it can be described as a hierarchical graph of objects. Some 3D scene description formats that support a hierarchical graph include: glTF [x9], USD [x10]

Labeling is done by attaching a label (e.g. a string value) to each 3D object .There is no defined format that specifies a list of possible labels. In some 3D scene description formats, fields carrying application-specific data may be used for labeling.

#### 4.2.6.1 Example: Space Setup (Meta)

Space Setup [10] automatically identifies and marks furniture in an indoor room. Segmented objects in Space Setup may assigned one of the following labels: floor (not displayed), ceiling (not displayed), wall (not displayed), door, table, sofa, storage, screen, or bed. Objects are represented with bounding boxes around each identified object and the segmentation can be adjusted manually.

A screenshot of a computer

Description automatically generated

Figure 4.2.6.1-1: Space Setup

#### 4.2.6.2 Example: RoomPlan API (Apple)

RoomPlan [11] is a Swift API that utilizes the camera and LiDAR Scanner on iPhone and iPad devices to create a 3D floor plan of a room, including key characteristics such as dimensions and types of furniture. The 3D object-detection pipeline recognizes 16 object categories directly in 3D.

Segmented objects in the RoomPlan API may be assigned one of the following labels: storage, sofa, table, chair, bed, refrigerator, oven, stove, dishwasher, washer/dryer, fireplace, sink, bathtub, toilet, stairs, or TV. Objects are represented with bounding boxes.

A screen shot of a room

Description automatically generated

Figure 4.2.6.2-1: RoomPlan API

#### 4.2.6.3 Example: Scene Semantics API (Google)

The Scene Semantics API [12] provides real-time outdoor semantic information, which complements existing geometric information in ARCore. Given an image of an outdoor scene, the API returns a label for each pixel across a set of useful semantic classes. ARCore provides a classification with 12 labels, as shown in Figure 4.2.6.3-1.

A screenshot of a video game

Description automatically generated

Figure 4.2.6.3-1: Scene Semantics API in ARCore

\* \* \* Next Change \* \* \*

### 4.2.7 Collider generation

Colliders are used to handle collision between objects. A collider is a component that defines the shape of an object for the purposes of physical collisions [13]. Colliders are invisible, and do not need to be the same shape as the object mesh.

The consistent handling of collisions between virtual and real objects requires real objects colliders. Hence, one of the functions provided by spatial computing platforms is the creation of colliders used by physics simulation engines. These colliders are generated based on the input 3D model of the surrounding environment.

Collision shape of the object may be defined using either primitive shapes or mesh-based shapes.

Primitive shapes, such as a box, sphere, capsule, cylinder, or cone, are best in terms of memory and performance but do not necessarily reflect the actual shape of the object. They are calculated based on the object’s bounding box [14].

Mesh-based shapes (e.g., convex hull or mesh) are calculated based on the geometry of the object and are therefore a better representation of the object.

A computer screen shot of a machine object

Description automatically generated

Figure 4.2.7-1: Combination of primitive colliders [15]

A green banana on a gray surface

Description automatically generated

Figure 4.2.7-2: Mesh collider [16]

Input data can be:

- The 3D Model

The output of the collider generation function is a set of colliders, possibly as part of the description of the 3D Model.

In the case of primitive shapes, the MPEG\_scene\_interactivity extention of ISO/IEC 23090-14 (MPEG-I Scene Description) and the KHR\_collision\_shapes glTF extension support simple geometric collider shapes such as box, capsule, cylinder, and sphere.

In the case of mesh colliders, the object’s mesh and collider’s mesh may be contained in the same descriptive file if the the format of the file supports more than one mesh per node (e.g., MPEG-I Scene Description or USD). Alternatively, the collider mesh shape associated with an object may be described separately using a simple format (e.g., OBJ, STL, or FBX).

\* \* \* Next Change \* \* \*

### 4.2.8 Light extraction

The consistent rendering of virtual and real objects, including occlusion and lighting/shadowing aspects, requires a 3D model and the detection/extraction of light parameters. The 3D model (from a certain point of view) can be used by the rendering engine for compositing.

Lighting estimation is a technique used by augmented reality platforms to match the lighting of the virtual objects in the scene to the lighting of the real world surrounding the viewer. Real light data provide different parameters that represent different characteristics of a light source. Those parameters can be used to instantiate a virtual light (i.e., a virtual representation of a real light) with the same properties, to achieve consistency between real and virtual scene. This virtual light is set with the same pose as the real one.

In ARCore, the Lighting Estimation API provides detailed information about the lighting in a scene [17].



Figure 4.2.8-1: Rendering without light extraction (left) vs. with light extraction (right). Real object shadow highlighted in red and virtual object shadows in green.

Figure 4.2.8-1 shows an example of an inconsistency between the shadows without light extraction (left) and a more consistent rendering with light extraction.

Some algorithms, based on cast shadow [18], use the 3D model of the real scene to estimate light source.

For light estimation, the following input data can be used:

- Sensor data:

- Images captured by AR Device

In XR applications, different types of light, each with specific properties, are possible and may include:

* Point light: is a light that is located at a point in the scene and emits light in all directions equally. Point lights have a specific range and only affects objects within this range.
* Spot light: is a light that is located at a point in the scene and emits light in a cone shape. It emits light in one direction and is constrained to a specific angle, resulting in a cone-shaped region of light.
* Directional light: is a light that is located infinitely far away and emits light in one direction only.
* Area light: a light that is defined by a rectangle or disc in the scene, and emits light in all directions uniformly across its surface area but only from one side of the rectangle or disc.
* Environmental light: is composed of image(s) of the surrounding environment including light sources. Several formats may be used for the image(s):
  + Cube Map: The environment is projected onto the faces of a cube and stored as six square images,
  + Equirectangular: image resulting from the projection of the surface of a spherical environment map.

Formats developed for virtual lights can be used to transport the characteristics of real lights.

\* \* \* Next Change \* \* \*

### 4.2.10 Summary of spatial description formats

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

*Table X – 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 Construction | Model | 3D model(s) of surrounding space. | Non-segmented model (e.g., point cloud, mesh with/without attributes) | OBJ, STL, PLY, FBX, glTF |
| Segmented model | glTF, USD |
| 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 |
| 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. |







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