 ISO/IEC JTC 1/SC 29/WG 3 N1468

**ISO/IEC JTC 1/SC 29/WG 3**

**MPEG Systems   
Convenorship: KATS (Korea, Republic of)**

**Document type:** Output Document

**Title: EE Description for Avatar representation formats**

**Status:** Approved

**Date of document:** 2025-02-10

**Source:** ISO/IEC JTC 1/SC 29/WG 3

**Expected action:** ACT

**Action due date:** 2025-02-14

**No. of pages:** 57 (with cover page)

**Email of Convenor:** young.L@samsung.com

**Committee URL:** <https://isotc.iso.org/livelink/livelink/open/jtc1sc29wg3>

**INTERNATIONAL ORGANISATION FOR STANDARDISATION**

**ORGANISATION INTERNATIONALE DE NORMALISATION**

**ISO/IEC JTC 1/SC 29/WG 3**

**CODING OF MOVING PICTURES AND AUDIO**

**ISO/IEC JTC 1/SC 29/WG 3 N01468**

**Geneva, CH – January 2025**

|  |  |
| --- | --- |
| **Source** | **WG03 (MPEG Systems)** |
| **Title** | **Exploration Experiments Description for MPEG-I Avatar Representation Formats** |
| **Editor** | **Anthony Trioux (Xidian Univ.)** |
| **MPEG number** | **24878** |

Contents

[1 EE1: Compression for Animation Streams and avatar components (ongoing) 1](#_Toc190052557)

[2 EE2: Integrating on Geometry Data Components for Avatar Data (ongoing) 33](#_Toc190052580)

[3 EE3: Other Animation Sample Formats for Animation Stream (ongoing) 44](#_Toc190052594)

[4 EE4: Content Discovery and Partial Access (opened) 49](#_Toc190052620)

[5 EE5: Animation Controllers (opened) 51](#_Toc190052621)

**1 EE1: Compression For Animation Streams and Avatar Components (ongoing)**

EE opened at MPEG #148. Please see WG03 N01468 for the latest description of this EE.

**2 EE2: Integrating On Geometry Data Components For Avatar Data (ongoing)**

EE opened at MPEG #148. Please see WG03 N01468 for the latest description of this EE.

**3 EE3: Other Animation Sample Formats For Animation Stream (ongoing)**

EE opened at MPEG #148. Please see WG03 N01468 for the latest description of this EE.

**4 EE4 ARF Manifest (opened):**

EE opened at MPEG #149. Please see WG03 N01468 for the latest description of this EE.

**5 EE5 Animation Controllers (opened):**

EE opened at MPEG #149. Please see WG03 N01468 for the latest description of this EE.

# EE1: Compression for Animation Streams and avatar components (ongoing)

## **1.1 Introduction**

This Exploration Experiment (EE) aims to evaluate and develop methods for compressing animation streams for avatars, focusing on both facial and body animations. The experiment will explore compression for various animation data types, including blendshapes, facial landmarks, animation controllers, and joint transforms, with the goal of establishing a set of standardized approaches that enable efficient and flexible encoding and decoding of avatar animation data across platforms. In addition, the EE will evaluate and develop methods for compressing avatar components from the ARF container, one example is the compression of blendshapes faces.

## **1.2 Problem statement**

The following statement on Compression for Animation streams are identified:

* Compression of Facial Animation Streams: Develop and test methods for compressing diverse facial animation data types, including blendshapes, landmarks, and animation controllers, focusing on preserving expression fidelity and temporal consistency. The exploration should address various compression levels (both lossless and lossy) to accommodate different bandwidth and latency requirements for real-time applications.
* Compression of Body Animation Streams: Address compression techniques for body animation streams, with a particular emphasis on joint transforms used for skeletal motion. The goal is to preserve the accuracy and natural flow of body movements while reducing the amount of data. This includes evaluating compression mechanisms for different joint hierarchies and motion patterns and examining trade-offs between compression efficiency (both lossless and lossy) and articulation fidelity.
* Standardization of Avatar Animation Data Formats: Propose a standardized format for compressed avatar animation data that is interoperable across platforms and supports a wide range of animation components. The format should be extensible to accommodate new animation parameters as they emerge, and must support compatibility with existing frameworks, such as Mediapipe, ARKit, OpenXR, etc.
* Evaluation of Compression Techniques on Realism and Responsiveness: Establish metrics and testing methodologies to assess the impact of compression on avatar realism and responsiveness, with an emphasis on user perception. The evaluation process will examine how different compression levels affect the perceived quality of facial expressions and body movement, including potential delays, artifacts, or loss of motion fidelity. Objective and subjective testing methods will be developed to validate the compression approaches.

The EE will focus on developing compression tools for avatar animation streams including but not limited to : blendshapes, joints, controllers, landmarks, etc. and support their integration into the new part related to Avatar representation format (Part 39).

## **1.3 Use cases relevant for the EE**

The work of the EE is based on the requirements defined in N00359 and use cases identified in m64008 .

The requirements defined in N00359 are currently organized into five different categories including: Avatar Representation, Coding format, Transport/Synchronization/Carriage, Storage and Privacy and Security requirements.

The requirements related to the “Avatar Representation” are summarized as follows:

* Interoperability: allow converting to and from other avatar representation.
* 3D Scenes: to allow integration into scene description solution.
* Attributes: requirements applicable to the features supported by the avatar, such as, CGI features as textures, geometries, skeletons, accessories, semantic definitions.

The requirements related to the “Coding Format” are summarized as follows:

* General: requirements applicable to the whole format and its use
* Geometry: requirements applicable to the geometry of the base avatar
* Interaction: requirements that relate to the interaction between Avatars and between Avatars and other objects in the scene

* Animation and control: requirements on the facial and body animation

Transport/Synchronization/Carriage: requirements related to the transport of the base avatar model and the associated animation streams

* Storage: requirements that pertain to the storage of the base avatar model
* Security and privacy: requirements on the protection of the avatar in different scenarios

The requirements and use cases explicit mention the interoperability concept to allow converting to and from other representation and the representation of all components of an avatar including but not exclusive to geometries, textures, skeletons, accessories and semantics.

## **1.4 Related (WG2) and Extracted (new) Requirements**

## **1.5 Relation to other activities (EE, requirements, etc…)**

## **1.6 Mandates**

## **1.7 Participants**

|  |  |  |  |
| --- | --- | --- | --- |
| Participant | Contact | Email | Type |
|  |  |  |  |
| Xidian Univ. | Anthony Trioux | anthony\_trioux@xidian.edu.cn | L |
| Qualcomm | Imed Bouazizi | bouazizi@qti.qualcomm.com | P |
| Interdigital | Joao Regateiro | joao.regateiro@interdigital.com | P |

(P = proponent, L = leader)

## **1.8 Information about proposed technologies**

The following contributions on Compression for Animation Stream have been submitted:

**Meeting #148**

m70195 [SD] **Avatar facial animation stream format and coding scheme**

**Meeting #149**

**[m71290 [SD][Avatar][ARF] Animation Bitstream and Compression with Landmarks](https://git.mpeg.expert/MPEG/Systems/SceneDescription/MPEG-Contributions/-/issues/809)**

This proposal has been accepted for inclusion in the EE1 and is documented below.

1. **Introduction**

In the MPEG#148 meeting, a new part (Part 39) of MPEG System workgroup (WG03) has been initiated with an initial output document WG03N01398 [1] and, consequently, three Exploratory Experiments [2] have also been initiated to continue the effort on delivering a new avatar format.

The content of this contribution addresses EE1 on “Compression for Animation Streams” and EE3 on “Other animation sample formats for animation stream” by providing an animation bitstream as the basis support for compression algorithms to be investigated and further proposed.

The initial proposal to address the issue of animation data streaming is present as an alternative in the current specification working draft [1]. While the proposed solution addresses the requirements of the storage and transport of animation streams, it is container-specific (i.e., it is mainly based on ISOBMFF and ZIP containers) and may not be applicable to delay-sensitive use cases.

This contribution presents a generic bitstream format for avatar animation streams that is compatible with the Avatar Representation Format (ARF) and can be encapsulated using different media containers or packetization mechanisms. This is based on our previous contribution m70326 [3] that was presented in the last meeting and included in the EE. It should also be noted that the proposed bitstream is also compatible with the ISOBMFF-based carriage approach with few modifications.

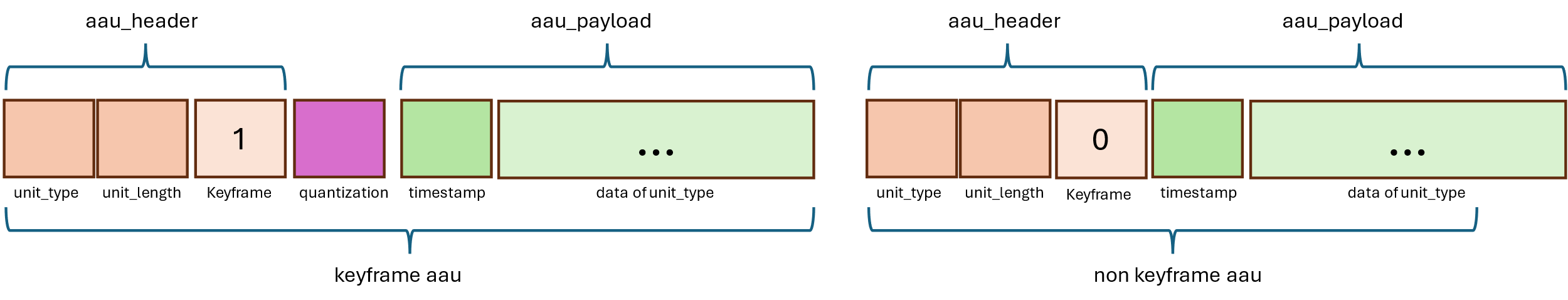
1. **Avatar Animation Bitstream**

An avatar animation bitstream is composed of a sequence of avatar animation units (AAUs), which may also be referred to as NAL units.

The general syntax structure for an AAU is shown in Table 1, where the data types used for the definition of different fields in the syntax structures are as follows.

* **uimsbf**: Unsigned integer with most significant bit first.
* **vlc8**: Variable length character string. Contains string data stored as a character array encoded in UTF-8.
* **boolean**: A single bit that represents a Boolean value.
* **float32**: A 32-bit floating point value represented according to the IEEE 754 specification.

Each avatar animation unit (AAU) contains a header and a payload. An AAU header contains at least a field that indicates the unit type and a field that indicates the AAU payload. The contents of the payload depend on the type of the AAU, where ByteAlignment is a padding with up to seven bits set to 0 for the AAU payload to be byte-aligned.



**Figure 1 - Illustration of binary structure using keyframe and non-keyframe AAUs.**

A keyframe AAU represents the main structure of the bitstream format including detailed information about the frame being transmitted e.g., quantization of the data of unit\_type different from configuration type (AAU\_CONFIG), and other properties of the payload defined in this document. A non-keyframe contains the same properties of the payload as for the keyframe with the exception that the data being transmitted is reconstructed relative to the previous keyframe AAU. To this end, the data of a non-keyframe is different between the current frame and the last keyframe, using the aau\_quantization as a precision parameter to reconstruct the current frame.

**Table 1 – Syntax of avatar\_animation\_unit()**

|  |  |  |
| --- | --- | --- |
| **Syntax** | **No. of bits** | **Mnemonic** |
| avatar\_animation\_unit() |  |  |
| { |  |  |
| aau\_header(); |  |  |
| if (aau\_keyframe) |  |  |
| aau\_quantization | 16 | **uimsbf** |
| aau\_payload(); |  |  |
| **ByteAlignement** | **0-7** | **uimsbf** |
| } |  |  |

* aau\_quantization: Indicates the type of the AAU signaled by aau\_unit\_type is not a keyframe therefore it carries a quantization scale for the carried data.

The syntax structure of the AAU header is as shown in Table 2.

**Table 2 – Syntax of aau\_header().**

|  |  |  |
| --- | --- | --- |
| **Syntax** | **No. of bits** | **Mnemonic** |
| aau\_header() |  |  |
| { |  |  |
| **aau\_unit\_type;** | **7** | **uimsbf** |
| **aau\_unit\_length;** | 32 | **uimsbf** |
| **aau\_keyframe;** | 1 | **boolean** |
| if (aau\_keyframe) |  |  |
| aau\_quantization | 16 | **uimsbf** |
| reserved | 1 | **uimsbf** |
| } |  |  |

The aau\_header() syntax construct contains the following syntax elements:

* aau\_unit\_type: indicates the type of the AAU. The possible values are described in Table 3.
* aau\_unit\_length: indicates the size of the AAU payload in bytes.
* aau\_keyframe: indicates whether the AAU payload is a keyframe.
* aau\_quantization: Indicates the type of the AAU signaled by aau\_unit\_type is not a keyframe therefore it carries a quantization scale for the carried data. The quantization processing model is described in Section 2.5.

**Table 3 – Avatar Animation Unit type codes and corresponding payloads.**

|  |  |  |
| --- | --- | --- |
| **aau\_unit\_type** | **Name of AAU type** | **Content of AAU payload** |
| 0 | AAU\_CONFIG | aau\_config\_unit() |
| 1 | AAU\_ANIMATION | aau\_animation\_unit() |
| 2 | AAU\_JOINT | aau\_joint\_unit() |
| 3 | AAU\_LANDMARK | aau\_landmark\_unit() |
| 4..10 | AAU\_RSV\_4 AAU\_RSV\_10 | Reserved AAU types. |
| 11..127 | AAU\_UNSPEC\_11 AAU\_UNSPEC\_127 | Unspecified AAU types. |

The aau\_payload() is defined as shown in Table 4.

**Table 4 – Syntax of aau\_payload().**

|  |  |  |
| --- | --- | --- |
| **Syntax** | **No. of bits** | **Mnemonic** |
| aau\_payload() |  |  |
| { |  |  |
| aau\_timestamp; | **32** | **uimsbf** |
| if (aau\_unit\_type == AAU\_CONFIG) |  |  |
| aau\_config\_unit() |  |  |
| else if (aau\_unit\_type == AAU\_ANIMATION) |  |  |
| aau\_animation\_unit() |  |  |
| else if (aau\_unit\_type == AAU\_JOINT) |  |  |
| aau\_joint\_unit() |  |  |
| else if (aau\_unit\_type == AAU\_LANDMARK) |  |  |
| aau\_landmark\_unit() |  |  |
| } |  |  |

The aau\_payload() syntax construct contains the following syntax elements:

* aau\_timestamp: is the timestamp of the AAU in ticks. The timestamp in seconds can be calculated as timestamp/timescale, where timescale is signalled in the configuration AAU.
* ByteAlignment: is a padding with up to seven bits set to 0 for the AAU payload to be byte-aligned.
  1. Configuration avatar animation unit

A configuration AAU is an AAU with aau\_unit\_type set to AAU\_CONFIG. The payload of such AAU is defined as shown in Table 5.

**Table 5 – Syntax of aau\_config\_unit().**

|  |  |  |
| --- | --- | --- |
| **Syntax** | **No. of bits** | **Mnemonic** |
| aau\_config\_unit() |  |  |
| { |  |  |
| **acu\_profile\_length;** | **8** | **uimsbf** |
| **acu\_animation\_profile;** | **acu\_profile\_length \* 8** | **vlc8** |
| **acu\_timescale;** | **32** | **uimsbf** |
| **acu\_target\_avatar\_info\_flag;** | **1** | **boolean** |
| if (acu\_target\_avatar\_info\_flag) { |  |  |
| **acu\_avatar\_id;** | **3** | **uimsbf** |
| **acu\_lod\_id;** | **3** | **uimsbf** |
| } |  |  |
| **acu\_animation\_correspondence\_flag;** | **1** | **boolean** |
| if (acu\_animation\_correspondence\_flag) { |  |  |
| **acu\_animation\_set\_id;** | **3** | **uimsbf** |
| **acu\_animation\_count;** | **10** | **uimsbf** |
| for (i=0; i<acu\_animation\_count; i++) { |  |  |
| **acu\_target\_animation\_index**[i]; | **10** | **uimsbf** |
| } |  |  |
| } |  |  |
| **acu\_joint\_correspondence\_flag;** | **1** | **boolean** |
| if (acu\_joint\_correspondence\_flag) { |  |  |
| **acu\_skeleton\_id;** | **3** | **uimsbf** |
| **acu\_joint\_count;** | **10** | **uimsbf** |
| for (i=0; i<acu\_joint\_count; i++) { |  |  |
| **acu\_target\_joint\_index**[i]; | **10** | **uimsbf** |
| } |  |  |
| } |  |  |
| **acu\_landmark\_correspondence\_flag;** | **1** | **boolean** |
| if (acu\_landmark\_correspondence\_flag) { |  |  |
| **acu\_landmarkset\_id;** | **3** | **uimsbf** |
| **acu\_landmark\_count;** | **10** | **uimsbf** |
| for (i=0; i<acu\_landmark\_count; i++) { |  |  |
| **acu\_target\_landmark\_index**[i]; | **10** | **uimsbf** |
| } |  |  |
| } |  |  |
| **acu\_reserved\_correspondence\_flags;** | **6** |  |
| **acu\_reserved\_unspecified\_flags;** | **22** |  |
| } |  |  |

The aau\_config\_unit() syntax construct contains the following syntax elements:

* acu\_profile\_length: is the number of characters in the profile string signalled by acu\_animation\_profile.
* acu\_animation\_profile: is a character string with the name of the profile that generated stream conforms to.
* acu\_timescale: is the number of ticks per second.
* acu\_target\_avatar\_info\_flag: is a flag indicating whether target avatar information is signalled in this configuration AAU. Value 0 indicates that no target avatar information is present. Value 1 indicates that target avatar information is present in the configuration AAU.
* acu\_avatar\_id: is an integer identifying the avatar to animate.
* acu\_lod\_id: is an integer identifying the level-of-detail (LoD) of the avatar to animate.
* acu\_animation\_correspondence\_flag: is a flag indicating whether animation correspondence information is signalled in this configuration AAU. Value 0 indicates that no correspondence information is present. Value 1 indicates that animation correspondence information is present in the configuration AAU.
* acu\_animation\_set\_id: is an integer identifying the animation set of the avatar to animate.
* acu\_target\_animation\_index[i]: is the index of the target animation corresponding to the source avatar’s i-th animation.
* acu\_joint\_correspondence\_flag: is a flag indicating whether joint correspondence information is signalled in this configuration AAU. Value 0 indicates that no correspondence information is present. Value 1 indicates that joint correspondence information is present in the configuration AAU.
* acu\_skeleton\_id: is an integer identifying the skeleton of the avatar to animate.
* acu\_target\_joint\_index[i]: is the index of the target avatar skeleton joint corresponding to the source avatar’s i-th joint.
* acu\_landmarks\_correspondence\_flag: is a flag indicating whether landmark correspondence information is signalled in this configuration AAU. Value 0 indicates that no correspondence information is present. Value 1 indicates that landmarks correspondence information is present in the configuration AAU.
* acu\_landmarkset\_id: is an integer identifying which collection of landmarks is used from the avatar.
* acu\_target\_landmark\_index[i]: is the index of the target avatar skeleton joint corresponding to the source avatar’s i-th joint.
* acu\_reserved\_correspondence\_flags: are reserved flags for the correspondences of future avatar components.
* acu\_reserved\_unspecified\_flags: are flags for unspecified features.
  1. Animation target avatar animation unit

An animation target AAU is an AAU whose aau\_unit\_type field is set to AAU\_ANIMATION. The payload of such AAU is defined as shown in Table 6.

**Table 6 – Syntax of aau\_animation\_unit().**

|  |  |  |
| --- | --- | --- |
| **Syntax** | **No. of bits** | **Mnemonic** |
| aau\_animation\_unit() |  |  |
| { |  |  |
| **amu\_animation\_target\_count\_minus1**; | **10** | **uimsbf** |
| for (i=1; i<amu\_animation\_target\_count\_minus1 + 1; i++) { |  |  |
| **amu\_animation\_source\_index**[i]; | **10** | **uimsbf** |
| **amu\_animation\_target\_index**[i]; | **10** | **uimsbf** |
| **amu\_animation\_target\_weight**[i]; | **32** | **float32** |
| } |  |  |
| } |  |  |

The aau\_animation \_unit() syntax construct contains the following syntax elements:

* amu\_animation\_target\_count\_minus1: plus 1 indicates the number of animation targets whose weights are signalled in this animation target AAU.
* amu\_animation\_source\_index[i]: is the index of the i-th animation source whose weight is signalled in the animation AAU.
* amu\_animation\_target\_index[i]: is the index of the i-th animation target whose weight is signalled in the animation AAU.
* amu\_animation\_target\_weight[i]: is the weight of the i-th animation target whose index is signalled by the field amu\_animation\_target\_index[i].
  1. Joint avatar animation unit

A joint AAU is an AAU whose aau\_unit\_type field is set to AAU\_JOINT. The payload of such AAU is defined as shown in Table 7.

**Table 7 – Syntax of aau\_joint\_unit().**

|  |  |  |
| --- | --- | --- |
| **Syntax** | **No. of bits** | **Mnemonic** |
| aau\_joint\_unit() |  |  |
| { |  |  |
| **aju\_joint\_count\_minus1**; | **10** | **uimsbf** |
| for (i=1; i<aju\_joint\_count\_minus1 + 1; i++) { |  |  |
| **aju\_source\_joint\_index**[i]; | **10** | **uimsbf** |
| **aju\_target\_joint\_index**[i]; | **10** | **uimsbf** |
| **aju\_joint\_transform**[i]; | **16 \* 32** | **float32** |
| } |  |  |
| } |  |  |

The aau\_joint\_unit() syntax construct contains the following syntax elements:

* aju\_joint\_count\_minus1: plus 1 indicate the number of joint transformations signalled in the joint transform AAU.
* aju\_source\_joint\_index[i]: indicates the source joint index for the i-th joint signalled in the joint transform AAU.
* aju\_target\_joint\_index[i]: indicates the target joint index for the i-th joint signalled in the joint transform AAU.
* aju\_joint\_transform[i]: is the transformation matrix for the target joint corresponding to the source joint whose index is signalled by the field aju\_source\_joint\_index[i].

Alternatively, the aju\_joint\_transform[i] field may be replaced by an encoding of optional translation, rotation, scale, velocity, inertia and uncertainty as presented in Table 8.

**Table 8 – Syntax of aau\_joint\_unit() with translation, rotation, and scale.**

|  |  |  |
| --- | --- | --- |
| **Syntax** | **No. of bits** | **Mnemonic** |
| aau\_joint\_unit() |  |  |
| { |  |  |
| **aju\_joint\_count\_minus1**; | **10** | **uimsbf** |
| for (i=1; i<aju\_joint\_count\_minus1 + 1; i++) { |  |  |
| **aju\_source\_joint\_index**[i]; | **10** | **uimsbf** |
| **aju\_target\_joint\_index**[i]; | **10** | **uimsbf** |
| **aju\_joint\_translation\_flag**[i]; | **1** | **boolean** |
| if (aju\_joint\_translation\_flag[i]) { |  |  |
| **aju\_joint\_translation**[i]; | **3 \* 32** | **float32** |
| } |  |  |
| **aju\_joint\_rotation\_flag**[i]; | **1** | **boolean** |
| if (aju\_joint\_rotation\_flag[i]) { |  |  |
| **aju\_joint\_quaternion\_flag**[i]; | **1** | **boolean** |
| if (aju\_joint\_quaternion\_flag[i]) { |  |  |
| **aju\_joint\_quaternion**[i]; | **4 \* 32** | **float32** |
| }else { | **4 \* 32** | **float32** |
| **aju\_joint\_rotation**[i]; | **4 \* 32** | **float32** |
| } |  |  |
| **aju\_joint\_quaternion\_flag**[i]; | **1** | **boolean** |
| if (aju\_joint\_quaternion\_flag[i]) { |  |  |
| **aju\_joint\_quternion**[i]; | **4 \* 32** | **float32** |
| } |  |  |
| **aju\_joint\_scale\_flag**[i]; | **1** | **boolean** |
| if (aju\_joint\_scale\_flag[i]) { |  |  |
| **aju\_joint\_scale**[i]; | **3 \* 32** | **float32** |
| } |  |  |
| **aju\_joint\_velocity\_flag**[i]; | **1** | **boolean** |
| if (aju\_joint\_velocity\_flag[i]) { |  |  |
| **aju\_joint\_velocity**[i]; | **16 \* 32** | **float32** |
| } |  |  |
| **aju\_joint\_inertia\_flag**[i]; | **1** | **boolean** |
| if (aju\_joint\_inertia\_flag[i]) { |  |  |
| **aju\_joint\_inertia**[i]; | **6 \* 32** | **float32** |
| } |  |  |
| **aju\_joint\_uncertainty\_flag**[i]; | **1** | **boolean** |
| if (aju\_joint\_uncertainty\_flag[i]) { |  |  |
| **aju\_joint\_uncertainty**[i]; | **32** | **float32** |
| } |  |  |
| } |  |  |
| } |  |  |

* aju\_joint\_translation\_flag[i]: is a flag indicating whether a translation is signalled for the i-th joint.
* aju\_joint\_translation[i]: is the translation for the target joint corresponding to the source joint whose index is signalled by the field aju\_source\_joint\_index[i].
* aju\_joint\_rotation\_flag[i]: is a flag indicating whether a rotation is signalled for the i-th joint.
* aju\_joint\_ rotation [i]: is the rotation for the target joint corresponding to the source joint whose index is signalled by the field aju\_source\_joint\_index[i].
* aju\_joint\_quaternion\_flag[i]: is a flag indicating whether a quaternion type of rotation is signalled for the i-th joint.
* aju\_joint\_quaternion [i]: is the quaternion type of rotation for the target joint corresponding to the source joint whose index is signalled by the field aju\_source\_joint\_index[i].
* aju\_joint\_scale\_flag[i]: is a flag indicating whether a scale is signalled for the i-th joint.
* aju\_joint\_scale[i]: is the scale for the target joint corresponding to the source joint whose index is signalled by the field aju\_source\_joint\_index[i].
* aju\_velocity\_flag[i]: is a flag indicating whether a velocity is signalled for the i-th joint.
* aju\_joint\_velocity[i]: is the velocity for the target joint corresponding to the source joint whose index is signalled by the field aju\_source\_joint\_index[i].
* aju\_inertia\_flag[i]: is a flag indicating whether an inertia is signalled for the i-th joint.
* aju\_joint\_inertia[i]: is the inertia for the target joint corresponding to the source joint whose index is signalled by the field aju\_source\_joint\_index[i].
* aju\_uncertainty\_flag[i]: is a flag indicating whether a uncertainty is signalled for the i-th joint.
* aju\_joint\_uncertainty [i]: is the uncertainty of the prediction for the animation parameter for the target joint corresponding to the source joint whose index is signalled by the field aju\_source\_joint\_index[i].
  1. Landmark avatar animation unit

A landmark AAU is an AAU whose aau\_unit\_type field is set to AAU\_LANDMARK. The payload of such AAU is defined as shown in Table 9.

**Table 9 – Syntax of aau\_landmark\_unit().**

|  |  |  |
| --- | --- | --- |
| **Syntax** | **No. of bits** | **Mnemonic** |
| aau\_landmark\_unit() |  |  |
| { |  |  |
| **alu\_landmark\_count\_minus1**; | **10** | **uimsbf** |
| for (i=1; i<alu\_landmark\_count\_minus1 + 1; i++) { |  |  |
| **alu\_source\_landmark\_index**[i]; | **10** | **uimsbf** |
| **alu\_target\_landmark\_index**[i]; | **10** | **uimsbf** |
| **alu\_landmark\_transform**[i]; | **16 \* 32** | **float32** |
| } |  |  |
| } |  |  |

The aau\_landmark\_unit() syntax construct contains the following syntax elements:

* alu\_landmark\_count\_minus1: plus 1 indicate the number of landmark transformations signalled in the landmark animation AAU.
* alu\_source\_landmark\_index[i]: indicates the source landmark index for the i-th landmark signalled in the landmark animation AAU.
* alu\_target\_landmark\_index[i]: indicates the target landmark index for the i-th landmark signalled in the landmark animation AAU.
* alu\_landmark\_transform[i]: is the transformation matrix for the target landmark corresponding to the source landmark whose index is signalled by the field aju\_source\_ landmark\_index[i].

Alternatively, the alu\_landmark\_transform[i] field may be replaced by an encoding of optional translation, rotation, scale, velocity, inertia and uncertainty as presented in Table 10.

**Table 10 – Syntax of aau\_landmark\_unit() with translation, rotation, and scale.**

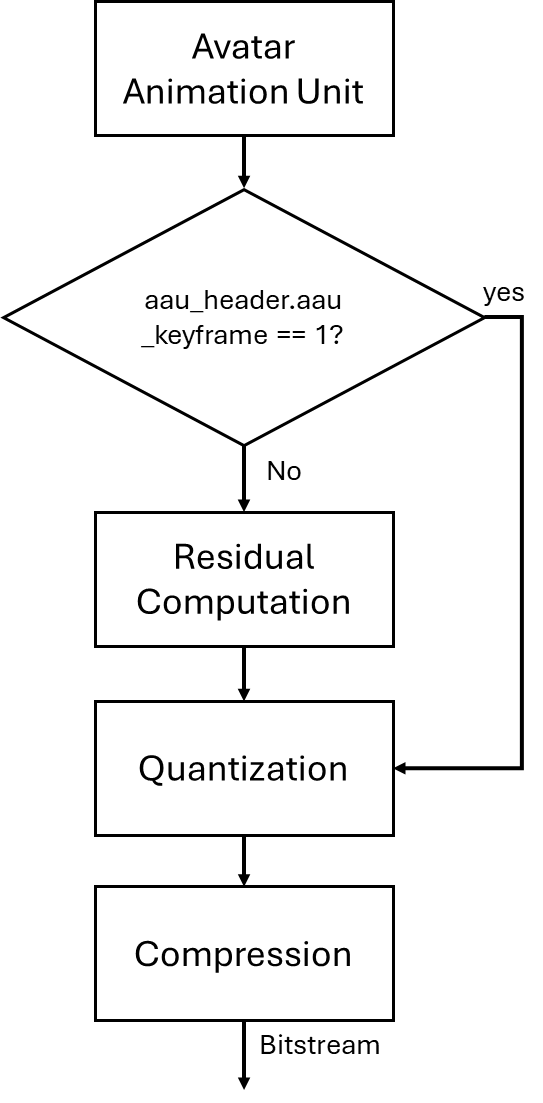
|  |  |  |
| --- | --- | --- |
| **Syntax** | **No. of bits** | **Mnemonic** |
| aau\_landmark\_unit() |  |  |
| { |  |  |
| **alu\_landmark\_count\_minus1**; | **10** | **uimsbf** |
| for (i=1; i<alu\_landmark\_count\_minus1 + 1; i++) { |  |  |
| **alu\_source\_landmark\_index**[i]; | **10** | **uimsbf** |
| **alu\_target\_landmark\_index**[i]; | **10** | **uimsbf** |
| **alu\_landmark\_translation\_flag**[i]; | **1** | **boolean** |
| if (alu\_landmark\_translation\_flag[i]) { |  |  |
| **alu\_landmark\_translation**[i]; | **3 \* 32** | **float32** |
| } |  |  |
| **alu\_landmark\_position\_flag**[i]; | **1** | **boolean** |
| if (alu\_landmark\_position\_flag[i]) { |  |  |
| **alu\_landmark\_position**[i]; | **3 \* 32** | **float32** |
| } |  |  |
| **alu\_landmark\_uncertainty\_flag**[i]; | **1** | **boolean** |
| if (alu\_landmark\_uncertainty\_flag[i]) { |  |  |
| **alu\_landmark\_uncertainty**[i]; | **32** | **float32** |
| } |  |  |
| } |  |  |
| } |  |  |

* alu\_landmark\_translation\_flag[i]: is a flag indicating whether a translation is signalled for the i-th landmark.
* alu\_landmark\_translation[i]: is the translation for the target landmark corresponding to the source landmark whose index is signalled by the field alu\_source\_landmark\_index[i].
* alu\_landmark\_position\_flag[i]: is a flag indicating whether a position is signalled for the i-th landmark.
* alu\_landmark\_position [i]: is the absolute position for the target landmark corresponding to the absolute position of the source landmark whose index is signalled by the field alu\_source\_landmark\_index[i].
* alu\_uncertainty\_flag[i]: is a flag indicating whether an uncertainty is signalled for the i-th landmark.
* alu\_landmark\_uncertainty [i]: is the uncertainty of the prediction for the animation parameter for the target landmark corresponding to the source landmark whose index is signalled by the field alu\_source\_landmark\_index[i].
  1. Quantization processing model

This section describes the process of using quantization to code and compress the bitstream format. The quantization flag in one embodiment can be fixed and normative with a defined default value for all the data parameters transmitted in the payload.

On a different embodiment the quantization flag is transmitted and unique to all data parameters in the payload, which means all data in the payload is quantified to an integer value to the same number of bits e.g., quantization of 10 bits.

On another embodiment the quantization flag is transmitted per keyframe and different for every data parameter transmitted in the payload, making the bitstream size variable depending on the type of data being transmitted in the payload.



**Figure 2 - Processing model for the avatar animation unit using quantization and compression.**

Figure 2 initiates with a complete AAU containing an aau\_header and an aau\_payload function, if the headed contains an aau\_keyframe the process applies the following quantization:

Where “data” can be represented by any data floating value present in the “aau\_payload” with a “aau\_type” equals to “AAU\_ANIMATION”, “AAU\_JOINT” or “AAU\_LANDMARK” e.g., “amu\_animation\_target\_weights[i]” that represent 32 bits of type float32, but not limited to. The QS can be defined by any variable integer value e.g., QS = 5, or QS = 10 or any positive number.

The resulted in one embodiment is used to define the number of bits for the data that needs to be compressed and transmitted, and so the bitstream is either positive or positive the size of .

The residual computation is performed as: if the “aau\_header” contains “aau\_keyframe=1”, then the final result is as follows:

This permits to transform the data to integers type. : if the “aau\_header” contains “aau\_keyframe=0”, then the final result is as follows:

The data of the previous frame is subtracted from the current fame, which results in applying quantization to the difference between two frames, being the previous frame a keyframe. The

Finally with the data’ is passed through the previous quantization function to achieve a positive or negative integer value.

Finally, the last component is to use a compression mechanism of integers, this can typically be entropy encoding, but not limited too.

**Proposal**

It is proposed to adopt the proposed bitstream format in section 2 of this contribution as an MPEG avatar animation bitstream into the working draft (WD) of ISO/IEC 23090-39.

**References**

[1] WG03N01398, “Avatar Representation Format”, MPEG#147, Kemer, Turkey, November 2024.

[2] WG03N01379, “EE Description of Avatar representation formats”, MPEG#147, Kemer, Turkey, November 2024.

[3] m70326, “[SD][Avatar] MPEG Avatar Animation Stream”, MPEG#147, Kemer, Turkey, November 2024.

**m71323 [ARF][EE1] Assessing Compressed Blendshape-Based Avatar Facial Animation: Objective Metrics vs. Subjective Scores**

This proposal has been accepted for inclusion in the EE1 and is documented below.

**2 Introduction**

The MPEG 148 meeting discussed the relevant proposals for the avatar WD (working draft) stage, focusing on two aspects: 1) avatar representation format design and 2) avatar animation stream format. This meeting established three exploratory experiments (EEs) on digital humans, among which EE1[1] was about "Compression for Animation Streams". m70195[2] proposed a compression coding scheme for digital human animation parameters and the corresponding bitstream structure. The compression method used inter-frame prediction (residual calculation of adjacent frames), followed by quantization and entropy coding. Experimental results showed that the compression scheme achieved valuable bitrate reduction (around 10 times reduction) while ensuring good visual quality. However, only subjective scores as well as bitrate computation were provided. Objective scores are essential as they provide a fast, consistent, and scalable alternative to subjective evaluations, which are often time-consuming and resource-intensive.

**3 Experimental setup**

In what follows, we evaluate the ability of widely used objective metrics to predict the observed MOS scores. For this evaluation, we consider popular metrics in the pixel domain, including Peak Signal-to-Noise Ratio (PSNR) [3], PSNR with consideration for the Human Visual System (PSNR-HVS) [4], Structural Similarity Index Measure (SSIM) [5], Multi-Scale Structural Similarity Index Measure (MS-SSIM) [6], Visual Information Fidelity (VIF) [7], Detail Loss Measure (DLM) [8], and Video Multi-method Assessment Fusion (VMAF) [9]. In addition, we also consider direct computation of PSNR in the BS and Head Pose (HP) domain. We first define the equation of the MSE in the BS and HP domain as follows:





where, MSE*BS(t)* and MSE*HP(t)* is the mean square error for each time index (frames) *t*, of the BS and HP weights, respectively. *K* and *L* is the number of BS, HP weights used in the application, respectively. *BSorii(t)*and *BSdisi(t)*are the *ith* original and reconstructed BS at frame *t*.

Then, for each time index, one can obtain two PSNR values:

*RBS* is the peak-to-peak range of the BS weights and *RHP* is the peak-to-peak range of the HP weights.

Finally, the PSNR for the complete sequence can be obtained by averaging all PSNR per frame together:

where *F* is the number of frames in the sequence. Note that one can also get a single value either per frame or per sequence by averaging the PSNR-*BS* and PSNR-*HP* together.



The above metrics are inspiredby the PSNR-FAP [10], which was developed during MPEG-4 FBA. The equation is as follows:

, 

where *N*FAP is the number of FAP's used for a sequence, *N*frames is the number of frames of the sequence, *Ri* the peak-to-peak range of the *i*th FAP, MSE*i* is the mean square error along the sequence for each decoded FAP, and fapori[i][j] and faprecon[i][j] are the original and decoded value of FAP *i* for frame *j*.

However, there is a criticaldistinction between the two metrics: in our case, the MSE computation is performed across all BS/HP foreach time index or frame, rather than per FAP over the temporal axis. Thisadjustment addresses two key issues with PSNR-FAP: (1) it mitigates the disproportionate impact of sparse or inactive BS,whose low MSE values can dominate the metric, and (2) itretains frame-level granularity, capturing temporal variationsin reconstruction fidelity.As a result, PSNR-BS+HP provides amore stable and perceptually aligned measure of blendshapecompression quality as shown below in Table I.

The evaluation considers three key aspects: accuracy, monotonicity, and consistency. These are measured using the following metrics: Pearson Correlation Coefficient (PCC) and Root Mean Squared Error (RMSE) for accuracy, Spearman Rank Order Correlation Coefficient (SROCC) for monotonicity, and Outlier Ratio (OR) for consistency [11]. The metrics are computed following the application of a non-linear regression to the data, as described in [12].

**4 Experimental results**

Table 1: SROCC, PCC, OR and RMSE results. Best values are shown in bold, second-best values in bold and underlined, and third-best values in italic bold.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | PCC↑ | SROCC↑ | OR↓ | RMSE↓ |
| PSNR[3] | 0.548 | 0.538 | **0.146** | 0.822 |
| PSNR HVS[4] | 0.552 | 0.540 | **0.135** | 0.819 |
| SSIM[5] | 0.500 | 0.499 | 0.156 | 0.851 |
| MS-SSIM[6] | 0.507 | 0.499 | 0.167 | 0.847 |
| VIF[7] | 0.561 | 0.547 | **0.146** | 0.813 |
| DLM[8] | | **0.611** | **0.601** | **0.135** | **0.778** |
| VMAF[9] | ***0.598*** | ***0.588*** | **0.135** | **0.787** |
| PSNR-FAP[10] | 0.265 | 0.272 | 0.208 | 0.947 |
| PSNR-BS | 0.323 | 0.287 | 0.198 | 0.930 |
| PSNR-HP | **0.601** | **0.614** | 0.177 | ***0.785*** |
| PSNR-BS+HP | 0.573 | 0.562 | 0.156 | 0.805 |

Results displayed in Table I highlight the limitations of traditional objective metrics in accurately predicting perceived quality after blendshape compression. Among the evaluated metrics, DLM, VMAF, and PSNR-HP demonstrate the highest correlation with the MOS values, with correlation coefficients reaching approximately 0.6 for both Pearson and Spearman correlations scores. This illustrates the needs of developing better objective metrics tailored blendshape-based facial avatar animation.

**5 Conclusion**

In this proposal, we have proposed a comprehensive evaluation of traditional objective metrics and highlighted their resulting correlations scores with subjective evaluations. Results reveal the limitations of traditional metrics to accurately predict perceived quality, with DLM, VMAF, PSNR-HP and PSNR-BS+HP achieving the highest correlation with MOS (∼ 0.6). Relying on the PSNR computed in the BS domain is proven to be a robust choice, as it directly evaluates the accuracy of the blendshape representation without being influenced by potential distortions introduced during the rendering process. Unlike PSNR in the pixel domain, which may account for such distortions, this metric focuses solely on the quality of the blendshape compression. Furthermore, it offers a more targeted evaluation of the encoding process, making it particularly relevant for applications prioritizing efficient transmission and reconstruction of facial animations. We recommend to consider the proposed metrics to evaluate future proposals in the Avatar working group.

**References**

[1] WG03 N01379, “EE Description for Avatar representation formats”, MPEG#148, Kemer, TK, November 2024.

[2] WG03 m70195, “Avatar facial animation stream format and coding scheme”, MPEG#148, Kemer, TK, November 2024.

[3] Q. Huynh-Thu and M. Ghanbari, “Scope of validity of PSNR in image/video quality assessment,” Electronics Letters, vol. 44, no. 13, pp. 800–801, 2008.

[4] Karen Egiazarian, Igor Kovalchuk, and Mikhail M. Gabbouj, “New full reference quality metrics based on hvs,” in Proceedings of the Second International Workshop on Video Processing and Quality Metrics. 2006, vol. 4, pp. 1–6, IST/SPIE.

[5] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli, “Image quality assessment: from error visibility to structural similarity,” IEEE Transactions on Image Processing, vol. 13, no. 4, pp. 600–612, 2004.

[6] Zhou Wang, Eero P. Simoncelli, and Alan C. Bovik, “Multi-scale structural similarity for image quality assessment,” Conference Record of the Thirty-Seventh Asilomar Conference on Signals, Systems and Computers, pp. 1398–1402, 2003.

[7] Hamid R Sheikh and Alan C Bovik, “Image information and visual quality,” IEEE Transactions on image processing, vol. 15, no. 2, pp. 430–444, 2006.

[8] Songnan Li, Fan Zhang, Lin Ma, and King Ngi Ngan, “Image quality assessment by separately evaluating detail losses and additive impairments,” IEEE Transactions on Multimedia, vol. 13, no. 5, pp. 935–949, Oct. 2011.

[9] C. G. Bampis, P. K. Yadav, A. M. B. Barbosa, S. S. R. K. M. K. A., and J. A. V. de Lemos, “Spatiotemporal feature integration and model fusion for full reference video quality assessment,” IEEE Transactions on Circuits and Systems for Video Technology, vol. 29, no. 8, pp. 2256– 2270, 2019.

[10] Hai Tao, Homer H Chen, Wei Wu, and Thomas S Huang, “Compression of MPEG-4 facial animation parameters for transmission of talking heads,” IEEE Transactions on circuits and systems for video technology, vol. 9, no. 2, pp. 264–276, 1999.

[11] International Telecommunication Union, “Methodologies for the subjective assessment of the quality of television images,” Recommendation ITU-R BT.500-15, May 2023.

[12] Video Quality Experts Group (VQEG), “Final report from the video quality experts group on the validation of objective models of video quality assessment, phase ii,” Tech. Rep., Video Quality Experts Group (VQEG), 2003.

**m71324 [ARF][EE1] Compression Methods and Stream Format for Blend Shape-based Avatar Facial Animation**

This proposal has been accepted for inclusion in the EE1 and is documented below.

**Introduction**

The MPEG 148 meeting discussed the relevant proposals for the avatar WD (working draft) stage, focusing on two aspects: 1) avatar representation format design and 2) avatar animation stream format. This meeting established three exploratory experiments (EEs) on digital humans, among which EE1[1] was about "Compression for Animation Streams". m70195[2] proposed a compression coding scheme for digital human animation parameters and the corresponding bitstream structure. The compression method used inter-frame prediction (residual calculation of adjacent frames), followed by quantization and entropy coding. Experimental results showed that the compression scheme achieved valuable bitrate reduction (around 10 times reduction) while ensuring good visual quality. However, the performance of this scheme considering lossless compression and/or all-intra mode was not considered in the last meeting.

In this proposal, we:

1. Add support for lossless coding and provide resulting bitrate for both lossless coding and all-intra mode;

2. Improve the prediction module, by adding an "intra-frame prediction" to further improve the compression rate;

3. Adjust the code stream structure to support more flexible and efficient transmission.

A notable feature of this contribution concerns the correlation analysis of animation parameters within the same frame, which allow to propose an “intra-frame prediction” mode relying on right/left blendshapes correlation to achieve further compression of the bitstream.

The application scenarios met by this contribution are daily conversation scenarios, which are applicable to the "3.1. Video Conferencing / Remote Calls" and "3.2. Telepresence / Immersive Experiences" use cases in N00301[3].

**3 Coding framework**

Compared to m70195, a new "scaling" module is added prior to any prediction mode. The overall coding framework consists of scaling, prediction, quantization, and entropy coding modules. The overall coding process is shown in Figure 1.

* ‘**Original data**’ corresponds to the BlendShape weights value, ranging from 0 to 1, with *p* effective digits.
* ‘**Scaling**’ scales up the original decimal value into an integer and retains all valid digits;
* In the ‘**Prediction module**’, different prediction modes (intra/inter prediction) can be selected to obtain the predicted value of the current one and compute the residual information;
* In the ‘**Quantization**’ module, it is possible to quantize the residual. Quantization will cause loss when rounding off the data. The value directly input to the entropy coding module without quantization is the predicted value itself.
* ‘**Entropy** **coding’** module uses ‘Exponential Golomb’ codes to obtain the final bitstream.

Note: *The dotted border in the block diagram*

Fig.1 BlendShape coding flow *indicates optional modules.*

**3.1 Introduction to scaling**

The scale-up module multiplies the original value (the decimal BS weight value of [0~1]) by an integer power of 10 to enlarge the small data and retain the precision. This operation also satisfies the implementation of lossless coding:

Taking exponential Golomb coding as an example, it requires integers at the input, so the goal of the "scaling" process module is to retain all the decimal digits, and then send the resulting integer to the exponential Golomb encoder for further encoding. The formula is as follows:



is the obtained integer value after scaling, *v* is the original data, and p is the scaling factor, which is determined by the number of digits after the decimal point of the original data.

**3.2 Multiple prediction modes**

**(1) Intra prediction:**

Because each animation parameter in a frame represents the animation of different parts of the digital human, when different animation parameters represent the movement of the same organ, there will be a certain correlation between them. For example, for the two animation parameters "EyeBlinkLeft" and "EyeBlinkRight", when the left eye blinks, the right eye often makes the same movement, and there is a strong correlation between the two. Therefore, for intra-frame prediction, we can use some animation parameters as a reference for the corresponding parameters of other part(s):



*BSi* is the animation parameter to be encoded in the current frame, *BSj* is its corresponding reference animation parameter in the same frame, with i ≠ j and pred\_intra is the resulting residual in all-intra mode.

Intra-frame prediction requires a correspondence table to reference such existing relationships between BlendShapes, as shown in Table 1.

Table 1. Blend shape reference correspondence table

|  |  |
| --- | --- |
| Can be referenced (BS\*) | Need to refer to BS\* |
| *BS1* | *BS2* |
| *BS3* | *BS4* |
| *…* | *…* |
| *BSm* | *BSn* |

Note: *The reference mapping in this table is related to the scene type of facial animation. This proposal considers the "normal dialogue" scene as use-case, which can meet most "non-extreme" other facial animation scenes.*

**(2) Inter prediction:**

Inter-frame prediction is a time domain redundancy removal process that calculates the residual of two consecutive frames:



 is the animation parameter to be encoded in the current frame,  is the animation parameter of the previous frame, and pred\_inter is the resulting residual of the animation parameter *i* in the time domain.

**(3) Joint prediction:**

For the animation parameter *BSi* to be encoded in the current frame, two predictions can also be superimposed at the same time:



pred\_intra\_inter is the residual obtained by intra prediction and then inter prediction.

Table 2. Prediction mode correspondance table

|  |  |
| --- | --- |
| Predictive mode | Description |
| 00 | No prediction |
| 01 | Intra prediction |
| 10 | Inter prediction |
| 11 | Intra-frame prediction followed by inter-frame prediction |

Note: *The solution supports a total of 4 prediction modes.*

**3.3 Quantization module**

The quantization module performs scalar quantization on the "prediction residual" data output by the prediction module according to the quantization step size:



*Y* is the output of the prediction module, *Qstep* is the quantization step size,  is the quantized value that will be input to the Golomb encoding module, and round() is the rounding operation to the nearest integer.

Different compression levels (or different error losses) are determined by the quantization step. The quantization step is generally described by a quantization parameter, similar to the 'QP' used in video coding. The quantization parameter '*QPBS*' of the blend shape can be set. The corresponding relationship between *QPBS* and *Qstep* is as follows:



Quantification can lead to loss, however when *QPBS* = 0, i.e., *Qstep* =1, the *Y* (prediction residual) will not be quantized, the encoded value is therefore lossless.

**3.4 Stream format**

This solution adopts a hierarchical structure design, extracting the common parameters of the entire video sequence frame separately to form a sequence parameter set.



Fig.2 Hierarchical structure of code stream

The parameter set is sent at the beginning of the session; The parameter set and data frame belong to two independent channels, and the parameter set can be reliably transmitted and retransmitted through the connection oriented TCP protocol to ensure the correct reception of the parameter set; Data frames can use UDP protocol to meet real-time requirements.

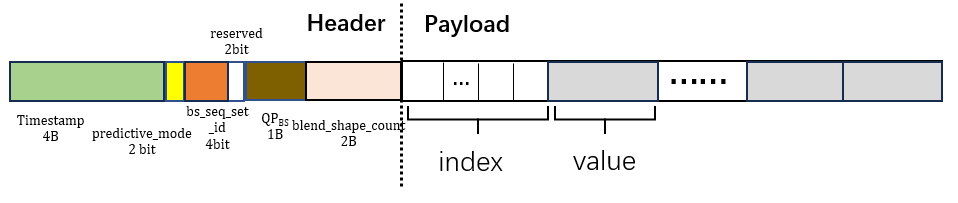


Fig.3 I frame structure of code stream

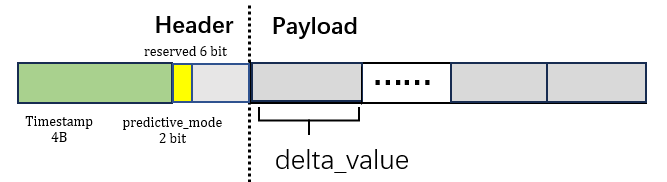


Fig.4 P frame structure of code stream

The header information and index information of the p-frame refer to the previous I-frame.

**4 Animation stream syntax and semantics**

The stream structure includes the syntax and semantics of the sequence set and BlendShape parameters for each frame.

**4.1 Sequence Set syntax and semantics**

The parameter set information includes the common parameters of the video sequence, and there can be many types of parameter sets, which are referenced by video frames based on their IDs.

***Syntax***

|  |  |
| --- | --- |
| aligned(8) class SeqParameterSet(){ | **Descriptor** |
| **bs\_seq\_set\_id**; | u(4) |
| **bs\_scale**; | u(4) |
| **reference\_flag;** | u(1) |
| **reserved;** | f(7) |
| **blend\_shape\_set;** | u(8) |
| if(reference\_flag) |  |
| **BS\_REFERENCE\_ TABLE；** | ue(v) |
| } |  |

***Semantics***

* bs\_seq\_set\_id: 4-bit unsigned integer, specifies the sequence parameter set number to be used.
* bs\_scale: 4-bit unsigned integer, scales up the original BS animation parameters.
* reference\_flag: 1-bit unsigned integer, whether to include the reference BS information table BS\_REFERENCE\_TABLE.
* reserved: A 7-bit general frame information reserved field for future extension of defined parameters.
* blend\_shape\_set: 8-bit unsigned integer, corresponding to the identifier of the BlendShape set. The default value 0 represents the blend shape set of MORGAN.
* BS\_REFERENCE\_ TABLE: Reference BS information table, as described in Table 1. " Blend shape reference correspondence table ".

**4.2 Data Frame syntax and semantics**

***Header***

The header information contains the metadata required to decode each frame of animation, including the frame's timestamp, parameter set number, prediction mode identifier, and some necessary control information identifiers.

***Syntax***

|  |  |
| --- | --- |
| aligned(8) class BlendShapeHeader(){ | **Descriptor** |
| **timestamp**; | u(32) |
| **predictive\_mode;** | u(2) |
| **if(predictive\_mode == 0 || predictive\_mode == 1){** |  |
| **bs\_seq\_set\_id**; | u(4) |
| **blend\_shape\_count\_enabled**; | u(1) |
| **reserved;** | f(1) |
| QPBS | u(8) |
| if (blend\_shape\_count\_enabled) { |  |
| **blend\_shape\_count**; | u(16) |
| } |  |
| } |  |
| **else:** |  |
| **reserved;** | f(6) |

***Semantics***

* timestamp: 32-bit unsigned integer, specifies the timestamp of the current frame, used for synchronization between frames.
* predictive\_mode: 2-bit unsigned integer, the first bit is 0/1 to indicate inter-frame prediction disabled/enabled; the second bit is 0/1 to indicate intra-frame prediction disabled/enabled.
* bs\_seq\_set\_id: 4-bit unsigned integer, specifies the sequence parameter set number used.
* blend\_shape\_count\_enabled: 1-bit Boolean value, specifies whether to enable the selection of the number of blend shapes. If 1, it allows the selection of non-default number of blend shapes.
* QPBS: 8-bit unsigned integer, blend shape weight quantization parameter.
* blend\_shape\_count: 16-bit unsigned integer, when blend\_shape\_count\_enabled is 1, specifies the total number of blend shapes, and the value does not exceed blend\_shape\_set.
* reserved: General frame information reserved bit.

***Payload***

The payload specifies the element values ​​to be encoded according to different prediction modes and control parameters. First, there are 4 cases according to the prediction mode. In each prediction mode, the number of BSs can be freely selected.

***Syntax***

|  |  |
| --- | --- |
| aligned(8) class BlendShapeFormat { | Descriptor |
| if(!predictive\_mode) { // 00, No predictions |  |
| if(blend\_shape\_set & blend\_shape\_count\_enabled) { |  |
| for (unsigned int i = 0; i < blend\_shape\_count; i++){ |  |
| **blend\_shape \_index[i]**; | ue(v) |
| } |  |
| for (unsigned int i = 0; i < blend\_shape\_count; i++) { |  |
| **blend\_shape\_weight [i]**; | se(v) |
| } |  |
| } |  |
| if(blend\_shape\_set & !blend\_shape\_count\_enabled) { |  |
| for (int i = 0; i < num(blend\_shape\_set); i++){ |  |
| **blend\_shape\_weight [i];** | se(v) |
| } |  |
| } |  |
| } |  |
| else if(predictive\_mode == 1){ // 01, Intra prediction |  |
| if (blend\_shape\_set & blend\_shape\_count\_enabled){ |  |
| for (unsigned int i = 0; i < blend\_shape\_count; i++){ |  |
| **blend\_shape \_index[i]**; | ue(v) |
| } |  |
| for (unsigned int i = 0; i < blend\_shape\_count; i++) { |  |
| if(blend\_shape \_index[i] in **BS\_REFERENCE\_TABLE** ){ |  |
| **blend\_shape\_reference\_weight**; | se(v) |
| **pred\_intra;** | se(v) |
| } |  |
| **blend\_shape\_ weight[i]**; | se(v) |
| } |  |
| } |  |
| if(blend\_shape\_set & !blend\_shape\_count\_enabled) { |  |
| for (int i = 0; i < num(blend\_shape\_set); i++){ |  |
| if (blend\_shape \_index[i] in **BS\_REFERENCE\_TABLE** ){ |  |
| **blend\_shape\_reference\_weight**; | se(v) |
| **pred\_intra;** | se(v) |
| } |  |
| **blend\_shape\_weight[i]**; | se(v) |
| } |  |
| } |  |
| } |  |
| else if(predictive\_mode == 2){ // 10, Inter prediction |  |
| if (blend\_shape\_set & blend\_shape\_count\_enabled){ |  |
| for (unsigned int i = 0; i < blend\_shape\_count; i++){ |  |
| **delta\_blend\_shape\_weight[i];** | se(v) |
| } |  |
| } |  |
| if (blend\_shape\_set & !blend\_shape\_count\_enabled){ |  |
| for (int i = 0; i < num(blend\_shape\_set); i++){ |  |
| **delta\_blend\_shape\_weight[i];** | se(v) |
| } |  |
| } |  |
| }else{ // 11, Inter after intra |  |
| if (blend\_shape\_set & blend\_shape\_count\_enabled){ |  |
| for (unsigned int i = 0; i < blend\_shape\_count; i++) { |  |
| for ( blend\_shape \_index[i] in **BS\_REFERENCE\_TABLE** ){ |  |
| **delta\_blend\_shape\_reference\_weight**; | se(v) |
| **pred\_intra\_inter;** | se(v) |
| } |  |
| **delta\_blend\_shape\_weight [i]**; | se(v) |
| } |  |
| } |  |
| if(blend\_shape\_set & !blend\_shape\_count\_enabled) { |  |
| for (int i = 0; i < num(blend\_shape\_set); i++){ |  |
| for ( int i: **BS\_REFERENCE\_TABLE** ){ |  |
| **delta\_blend\_shape\_reference\_weight**; | se(v) |
| **pred\_intra\_inter;** | se(v) |
| } |  |
| **delta\_blend\_shape\_ weight [i]**; | se(v) |
| } |  |
| } |  |
| } |  |
| } |  |

***Semantics***

* blend\_shape \_index[i] : integer array, the index value of the i-th blend shape weight in the current frame, indicating the index represented by the current i-th blend shape.
* blend\_shape\_weight[i]: Float array, the value of the i-th blend shape in the current frame.
* delta\_blend\_shape\_weight[i]: floating point array, the difference between the current frame blend\_shape\_weight[i] and the previous frame blend\_shape\_weight[i] reconstruction value.
* blend\_shape\_reference\_weight: the value of the blend shape weight in the BS\_REFERENCE\_TABLE of the current frame.
* BS\_REFERENCE\_TABLE: Configuration information table - BS reference mapping table.
* pred\_intra: The residual value obtained by intra-frame prediction of the current frame BS according to blend\_shape\_reference\_weight.
* delta\_blend\_shape\_reference\_weight: the temporal difference of the blend shape weight of the current frame BS in BS\_REFERENCE\_TABLE.
* pred\_intra\_inter: The temporal difference of the residual value obtained by intra-frame prediction of the current frame BS according to blend\_shape\_reference\_weight.

**5 Experimental results**

We used a camera to collect a large amount of BlendShape original data from real male and female users considering “dialogue” scenes use-cases, and divide the data into different sub-sequences. Each sub-sequence duration is about 20s and the frame rate is 30fps. According to the compression algorithm designed above, the compression coding of the original data is made, and the rate-distortion performance of one of the sub-sequence of the “maleTalking” type is as follows.

* **Bitrate reduction:**

Table 3. Bitrate results of 4 *QPBS* considering the different prediction modes

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Prediction modes  Different *QPBS* | 00 | 01 | 10 | 11 | IEEE 754  (float32) |
| bit\_rate  (bps) | bit\_rate  (bps) | bit\_rate  (bps) | bit\_rate  (bps) | bit\_rate  (bps) |
| ***QPBS\_*11**\_maleTalking | 15778.86 | 13624.23 | 6346.988 | 5529.088 | 49920 |
| ***QPBS\_*13**\_maleTalking | 11232.91 | 9443.442 | 3914.355 | 3464.923 | 49920 |
| ***QPBS\_*14**\_maleTalking | 9070.43 | 7556.489 | 3086.954 | 2793.666 | 49920 |
| ***QPBS\_*15**\_maleTalking | 7086.816 | 5888.468 | 2500.792 | 2353.735 | 49920 |

The yellow highlight in Table 3 corresponds to the best bitrate reduction, i.e., the "best prediction mode (intra+inter)", which can achieve 9 to 20 times compression at four quantization parameter scales compared with the IEEE 754[4]. The rate-distortion analysis is performed in the following subsections, while the video rendering quality will be shown during the oral presentation.

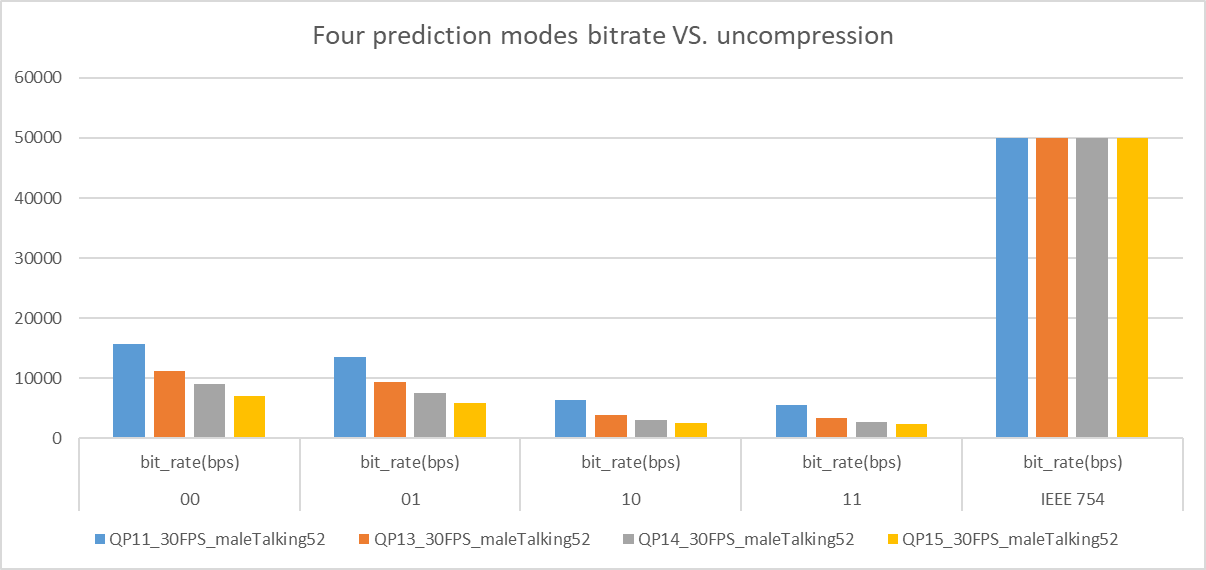
******

Fig.5 Bitrate results of 4 *QPBS* vs IEE754

* **Lossless performance:**

Table 4 shows the lossless compression (*QPBS*=0) performance of the scheme. The data shown in the table is the bit rate when bs\_scale is 6 (that is, the original data is accurate to 6 decimal digits). In the best prediction mode, the compression ratio achieved is nearly twice that of IEEE 754.

Table 4. Lossless results in different prediction modes

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Prediction modes  Different *QPBS* | 00 | 01 | 10 | 11 | IEEE 754  (float32) |
| bit\_rate  (bps) | bit\_rate  (bps) | bit\_rate  (bps) | bit\_rate  (bps) | bit\_rate  (bps) |
| ***QPBS\_*0**\_maleTalking | 41753.803 | 39897.831 | 30156.557 | **28740.516** | 49920 |
| ***QPBS\_*11**\_maleTalking | 15778.86 | 13624.23 | 6346.988 | 5529.088 | 49920 |
| ***QPBS\_*13**\_maleTalking | 11232.91 | 9443.442 | 3914.355 | 3464.923 | 49920 |
| ***QPBS\_*15**\_maleTalking | 7086.816 | 5888.468 | 2500.792 | 2353.735 | 49920 |

* Note1: *The yellow highlight in Table 4 is the bit rate comparison between lossless coding and IEEE 754 coding, and the bold data is the best performance of lossless coding of this scheme.*
* Note2: *When using lower precision values ​​for the capture or renderer, higher lossless compression ratio can be achieved by reducing the value of* bs\_scale*.*

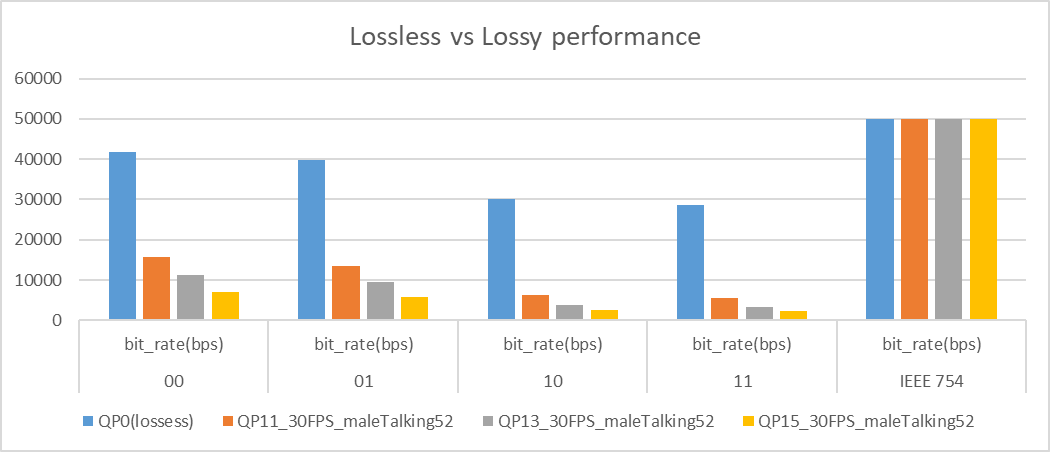
****

Fig.6 Lossless vs Lossy performance

* **Rate-distortion performance**

This section shows the loss of reconstructed data and original data of blend shape weights under the above four *QPBS*, which is reflected by calculating the MSE and PSNR of blend shape data.

The definition and related equation(s) of the BS-PSNR can be found in our other proposal m71323 “[ARF][EE1] Assessing Compressed Blendshape-Based Avatar Facial Animation: Objective Metrics vs. Subjective Scores”[5].

Table 5. Rate-distortion performance under 4 *QPBS* with 4 prediction modes

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Bitrate** | **PSNR** | **MSE** |
| ***QPBS*\_11**\_maleTalking00 | 15778.86 | 65.82 | 2.65E-07 |
| ***QPBS*\_11**\_maleTalking01 | 13624.23 | 65.82 | 2.65E-07 |
| ***QPBS*\_11**\_maleTalking10 | 6346.99 | 65.82 | 2.65E-07 |
| ***QPBS*\_11**\_maleTalking11 | 5529.09 | 65.82 | 2.65E-07 |
| ***QPBS*\_13**\_maleTalking00 | 11232.91 | 53.81 | 4.21E-06 |
| ***QPBS*\_13**\_maleTalking01 | 9443.44 | 53.81 | 4.21E-06 |
| ***QPBS*\_13**\_maleTalking10 | 3914.36 | 53.57 | 4.47E-06 |
| ***QPBS*\_13**\_maleTalking11 | 3464.92 | 53.57 | 4.47E-06 |
| ***QPBS*\_14\_**maleTalking00 | 9070.43 | 47.79 | 1.68E-05 |
| ***QPBS*\_14\_**maleTalking01 | 7556.49 | 47.79 | 1.68E-05 |
| ***QPBS*\_14\_**maleTalking10 | 3086.95 | 47.40 | 1.87E-05 |
| ***QPBS*\_14\_**maleTalking11 | 2793.67 | 47.40 | 1.87E-05 |
| ***QPBS*\_15**\_maleTalking00 | 7086.82 | 41.51 | 7.14E-05 |
| ***QPBS*\_15**\_maleTalking01 | 5888.47 | 41.51 | 7.14E-05 |
| ***QPBS*\_15**\_maleTalking10 | 2500.79 | 41.69 | 7.07E-05 |
| ***QPBS*\_15**\_maleTalking11 | 2353.74 | 41.69 | 7.07E-05 |

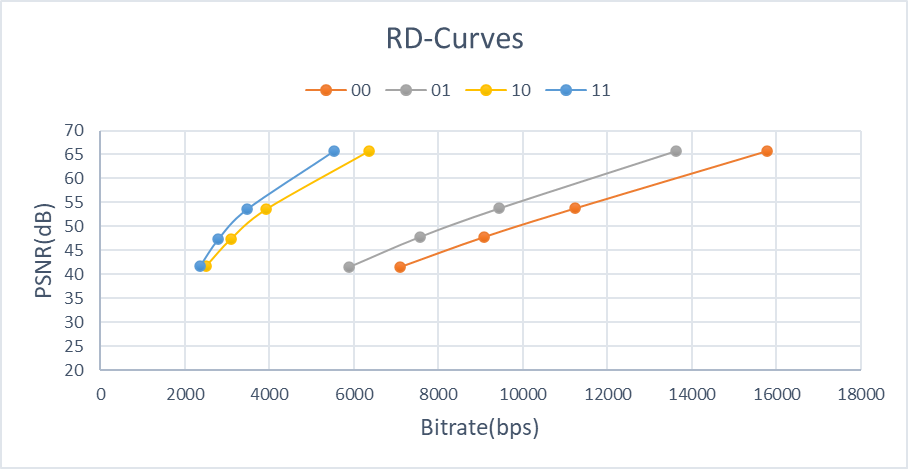


Fig.7 R-D Curves under 4 *QPBS* with 4 prediction modes

**6 Conclusion**

In this proposal, we have further explored the compression of Blend shape data for facial animation. Specifically, we proposed and implemented a coding scheme with the relevant bitstream structure for Blend shape data facial animation, presented detailed experimental results, and analyzed the performance comparison between the different proposed prediction modes. This coding scheme can be applied to the application scenarios of "3.1. Video Conferencing / Remote Calls" and "3.2. Telepresence / Immersive Experiences" in N00301, and other facial animation scenarios can be adapted accordingly by adjusting the Blend shape reference correspondance table. The experimental results show that this scheme achieves good compression performance while preserving good quality of the animated avatar.

**References**

[1] WG03 N01379, “EE Description for Avatar representation formats”, MPEG#148, Kemer, TK, November 2024.

[2] WG03 m70195, “Avatar facial animation stream format and coding scheme”, MPEG#148, Kemer, TK, November 2024.

[3] WG02 N00301, “Use Cases for MPEG Avatar,” MPEG#143, Geneva, Switzerland, July 2023.

[4] Kahan W. IEEE standard 754 for binary floating-point arithmetic[J]. Lecture Notes on the Status of IEEE, 1996, 754(94720-1776): 11.

[5] WG03 m71323, “[ARF][EE1] Assessing Compressed Blendshape-Based Avatar Facial Animation: Objective Metrics vs. Subjective Scores”, MPEG#149, Geneva January 2025.

**m71325 [SD] Storage of blendshapes in ARF container**

This proposal has been accepted for inclusion in the EE1 and is documented below.

The Avatar Representation Format (ARF) as defined in ISO/IEC 23090-39 has adopted a blendshape-based approach for facial animation. The user’s facial expressions are tracked, e.g. by an XR runtime, as a vector of weights, each of which represents the amount a particular blendshape contributes to the user’s current facial expression.

To enable receivers to properly animate the user’s avatar independently of which blendshape set is being used, the ARF container stores the base avatar model’s blendshapes, potentially also in different levels of detail.

The following figures depict some examples of these blendshapes:

A close up of a person's face

Description automatically generatedA white mannequin head with a grey background

Description automatically generatedA white mannequin head

Description automatically generated

The following table shows some selected set of facial tracking frameworks and their associated number of blendshapes:

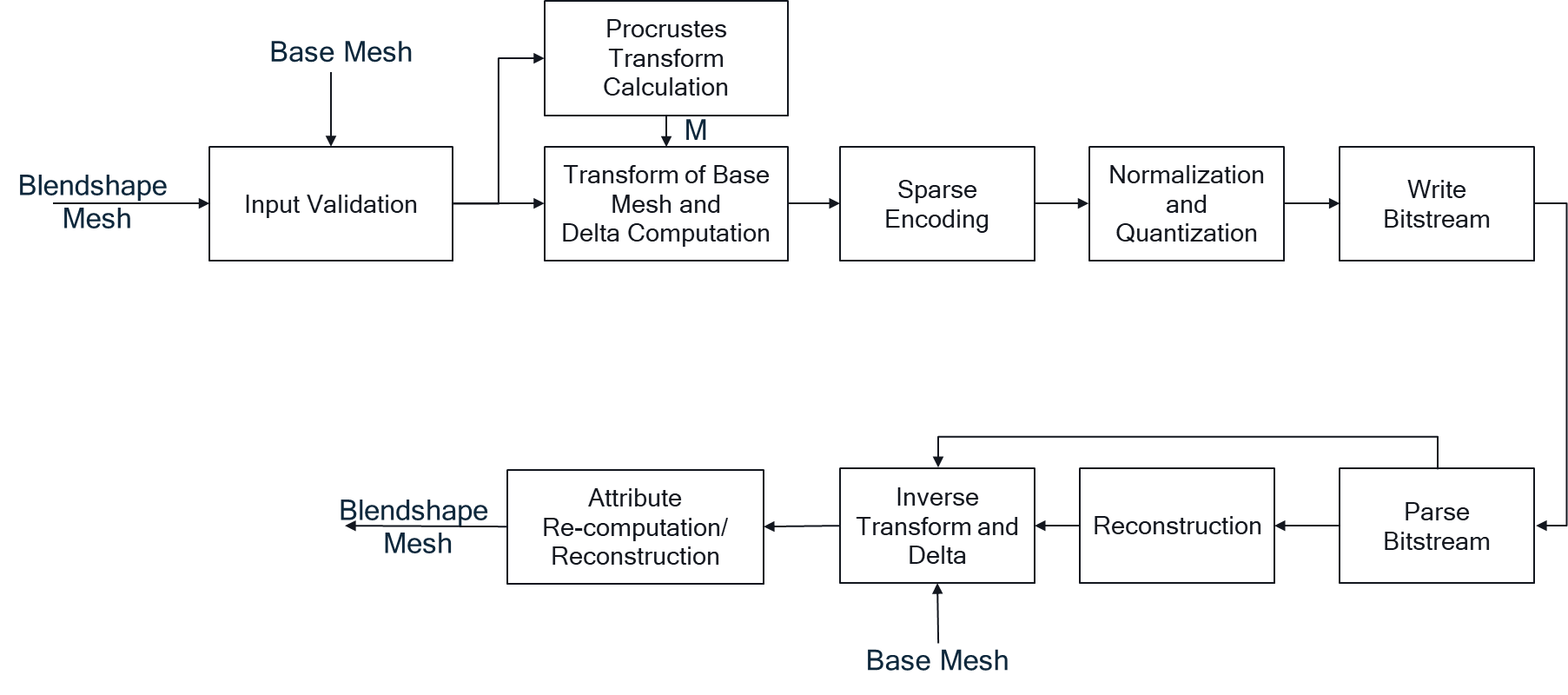
|  |  |
| --- | --- |
| **Framework** | **Number of Blendshapes** |
| [Apple ARKit](https://developer.apple.com/documentation/arkit/arfaceanchor/blendshapelocation) | 52 |
| [HTC Vive OpenXR](https://registry.khronos.org/OpenXR/specs/1.1/html/xrspec.html" \l "XR_HTC_facial_tracking) Facial Tracking | 38 |
| [Meta (Facebook) OpenXR](https://registry.khronos.org/OpenXR/specs/1.1/html/xrspec.html" \l "XR_FB_face_tracking2) Face Tracking | 63 and 70 |
| [Android OpenXR](https://developer.android.com/develop/xr/openxr/extensions/XR_ANDROID_face_tracking) Face Tracking Extension | 70 |
| [Qualcomm QFace](https://www.qualcomm.com/developer/blog/2024/12/driving-photorealistic03d-avatars-in-real-time-on-device-3d-gaussian-splatting) | 84 |

As can be noted from the table, a range of 52-84 blendshapes would be required to achieve accurate facial expressions in Avatar communications. This poses significant challenges on the size of the ARF container, especially when multiple levels of detail are also supported.

**Blendshape compression**

To address this issue, we propose to introduce a simple blendshape compression scheme to be supported in the Avatar Representation Format specification. This scheme is specifically designed for blendshapes and takes into account that the blendshapes only contain updated vertex positions compared to the base mesh (which is the face in neutral expression).

The ARF blendshape compressor follows the following architecture:



The following sections give details about each of these blocks.

Input Validation

Input validation is the first step of the compression pipeline, ensuring that the base mesh and the target blendshape meshes are compatible for processing. This involves loading the base mesh and blendshape meshes and verifying that their topologies match. Topology preservation is confirmed by checking that both meshes have the same number of vertices, the same number of faces, and identical face indices. If any discrepancies are found during these checks, the compression process stops.

Procrustes Transformation

The Procrustes transformation step involves the computation of an optimal transformation for aligning the base and target blendshape meshes to minimize positional differences. First, the centroids of both meshes are calculated by averaging their vertex positions. The centroid is then subtracted from each vertex to center the meshes at the origin.

Next, the scales of the meshes are computed by taking the square root of the sum of squared distances of vertices from the centroid. The scale ratio between the meshes is calculated and applied to normalize them as follows:

Subsequently, the rotation matrix is computed using the covariance matrix derived from the centered and scaled vertices. Singular Value Decomposition (SVD) is performed on the covariance matrix to extract the optimal rotation matrix as follows:

Transform of Base Mesh and Delta Computation

The determined transform operations (translation, scale, and rotation) are applied to the base mesh to align it with the target blendshape.

The difference or delta computation is then performed. After applying the optimal transformation, the vertex positions of the transformed base mesh are subtracted from the corresponding vertex positions of the target blendshape.

These differences are stored as an Nx3 matrix, where N represents the number of vertices. This matrix represents the deltas, or displacements, that define the deformation of the base mesh into the target shape.

Sparse Encoding

Sparse encoding is used to achieve compression efficiency by encoding only significant vertex deltas. This is driven by the fact that only a small subset of the vertices from the base mesh are deformed to achieve a specific facial expression of the target blendshape.

A threshold, typically set to a small value such as , is used to determine which deltas are considered significant. Deltas below this threshold are treated as negligible and discarded. The indices of vertices with significant deltas are then stored using a predictive encoding scheme to further reduce storage overhead.

Optional Quantization

Quantization is an optional step that reduces data precision to achieve higher compression ratios. The desired bit depth, such as 8, 12, or 16 bits, is chosen based on a tradeoff between compression efficiency and fidelity to the original mesh. Per-coordinate statistics, such as minimum, maximum, and range values, are calculated to normalize the data. The deltas are scaled to fit within the quantization range, and the scaling parameters are stored for reconstruction. Currently, we use a linear quantization process.

Bitstream Writing and Parsing

Bitstream writing and parsing involves writing and reading the compressed blendshape bitstream and extracting compressed data and metadata required for the reconstruction of the blendshape. Metadata includes the number of vertices, the transformation matrix, and quantization parameters, if applicable. The compressed data consists of indices and values of the deltas. This step prepares the data for the reconstruction process by ensuring all necessary information is correctly interpreted.

In the writing process, a Zlib encoding (Huffman entropy coding) algorithm is applied to the sparse encoded position data to further reduce the size of the bitstream.

Attribute Re-computation

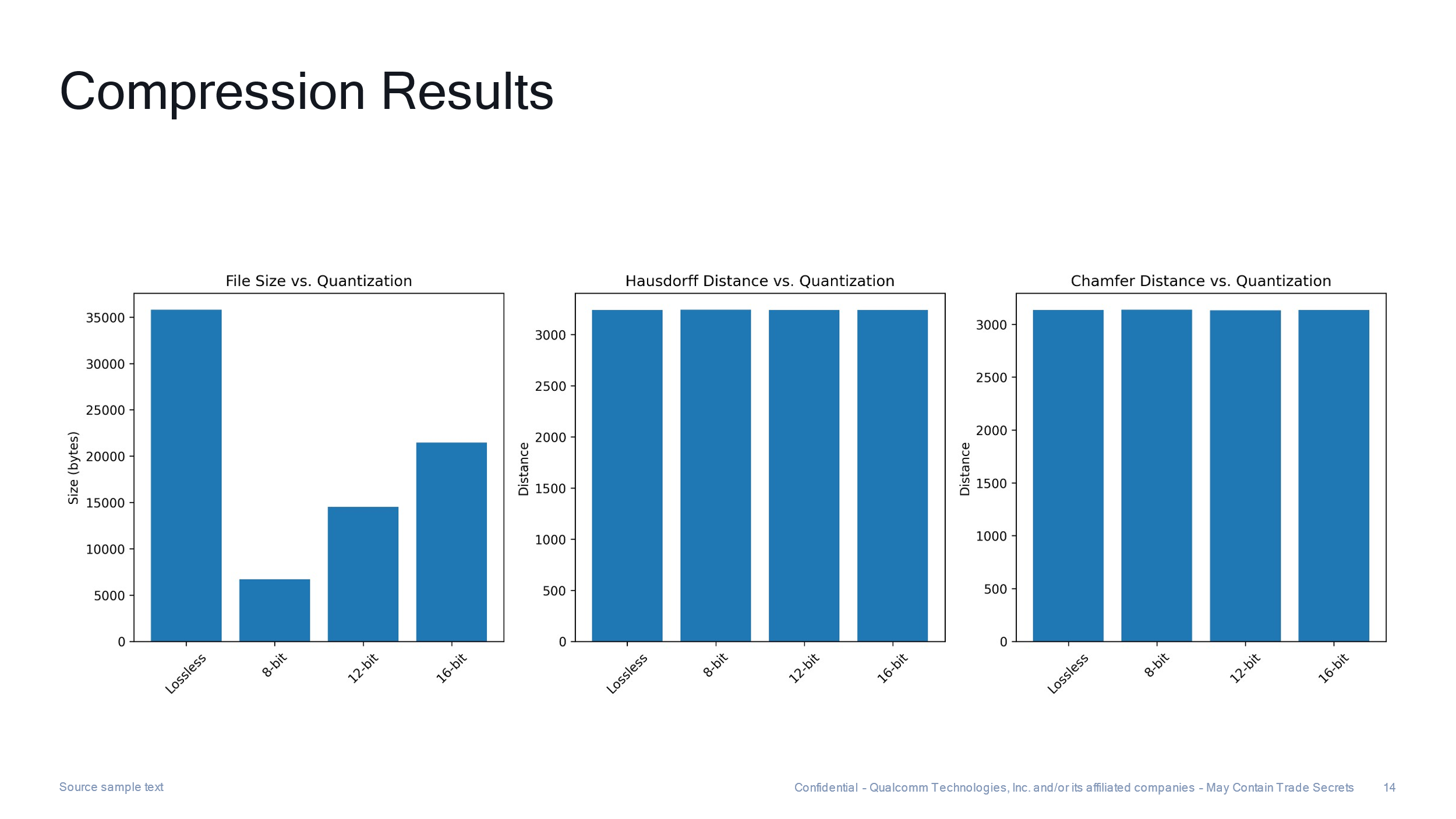
Certain attributes of the blendshape are changed as part of the deformation from the base mesh. The main example of such an attribute are the normals. Fortunately, the normals of the blendshape don’t have to be carried as part of the compressed blendshape. Instead, they can be easily recomputed after the decompression process.

The process for recomputing normal is as follows:

* Average the normals of all adjacent faces for each vertex.
* Normalize these vectors to unit length.

**Results**

The following results have been generated using the Alexis blendshape set:



The size savings are depicted in the following figure:

**Source Code**

The source code will be made available on GitLab. It includes the MeshCompressor as well as scripts for encoding/decoding and evaluating the performance of the compression algorithm.

**Proposal**

We propose to consider adding a basic compression scheme for blendshapes to reduce the total size of the ARF container to a reasonable size.

## **1.9 Extracted from TuC**

## **1.10 Test cases**

The EE may define test cases for which the evaluation criteria will be analyzed. For instance, a first test case can be with live content while another in the on-demand content.

## **1.11 Evaluation criteria**

List of criteria that will allow to compare the different technical solutions and converge to a unique solution. Criteria can be objective like memory efficiency, bitrate or subjective flexibility, compatibility with legacy solution, etc.

|  |  |  |
| --- | --- | --- |
| **Criteria** | **Description** | **Evaluation** |
| Criteria #1 | Description | The technical solution should minimize/optimize … |

## **1.12 Timeline**

MPEG #150: Document solutions and evaluation results and recommend changes to the CD text.

## **1.13 References**

[1] WG03 N01316, “Procedures and Test Scenario for MPEG Avatar Representation”, MPEG#147, Sapporo, Japan, July 2024.

[2] WG02 N00359, “Draft of MPEG-I Phase 3 requirements”, MPEG#146, Rennes, France, April 2024.

[3] WG03 N0XXXX, “WD of ISO/IEC 23090-38 Avatar representation formats”, Kemer, Turkey, November 2024.

[4] WG02 m64008 “Use Cases for MPEG Media Avatar”, Hanover, Germany, October 2023.

# EE2: Integrating on Geometry Data Components for Avatar Data (ongoing)

## **1.1 Introduction**

Currently, there are several formats (FBX, USD or glTF) for encoding 3D content including mesh rigs, e.g., deformable 3D objects with coefficient parametrizations. None of these formats are dedicated to avatar data representation. In most cases the definition and parsing of the avatar data structure is given to the application has a task, such as parsing existing formats.

The non existing of a standardized avatar format motivate for the initiation of the current work and the combined effort to produce a format that is interoperable between application and compatible with existing frameworks.

The goal of the EE is thus to specify the integration of avatar data information into a container format that is accessible in the avatar data model. The objective is to improve interoperability, avoid the exclusive used of external limited formats and provide a complete representation of an avatar format in MPEG-I. The EE will study the integration of the data components and based on this integration and the possible identified technical gaps; the EE aims at defining the necessary data components to the data structure of the avatar representation format.

## **1.2 Problem statement**

An interoperable avatar representation format should be capable of allowing other formats to decode, understand, map and utilize its data model structures for individual applications. In the current days an avatar format is a combination of several formats (illustrated in Figure X).

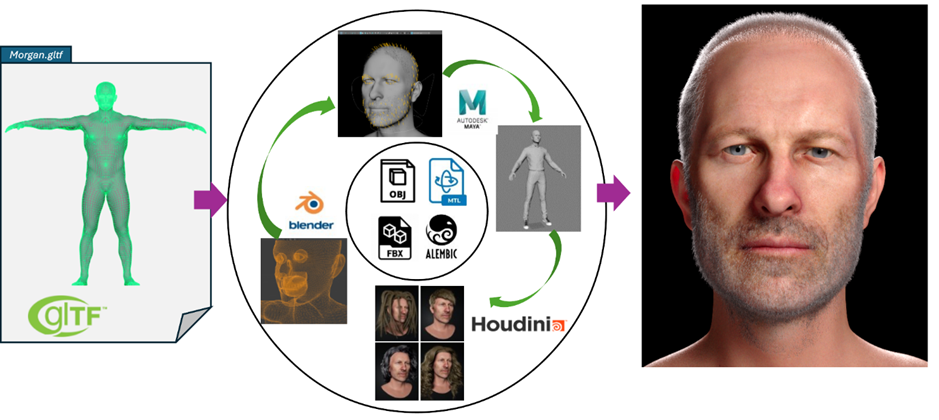


Figure 1 - Ecosystem of avatar asset creation.

This is impractical and not conformed with the claim for an interoperable format.

An interoperable format requires:

* Representation and definition of basic structures e.g., meshes, joints, textures and external file references.

## **1.3 Use cases relevant for the EE**

The work of the EE is based on the requirements defined in N00359 and use cases identified in m64008 .

The requirements defined in N00359 are currently organized into five different categories including: Avatar Representation, Coding format, Transport/Synchronization/Carriage, Storage and Privacy and Security requirements.

The requirements related to the “Avatar Representation” are summarized as follows:

* Interoperability: allow converting to and from other avatar representation.
* 3D Scenes: to allow integration into scene description solution.
* Attributes: requirements applicable to the features supported by the avatar, such as, CGI features as textures, geometries, skeletons, accessories, semantic definitions.

The requirements related to the “Coding Format” are summarized as follows:

* General: requirements applicable to the whole format and its use
* Geometry: requirements applicable to the geometry of the base avatar
* Interaction: requirements that relate to the interaction between Avatars and between Avatars and other objects in the scene

* Animation and control: requirements on the facial and body animation

Transport/Synchronization/Carriage: requirements related to the transport of the base avatar model and the associated animation streams

* Storage: requirements that pertain to the storage of the base avatar model
* Security and privacy: requirements on the protection of the avatar in different scenarios

The requirements and use cases explicit mention the interoperability concept to allow converting to and from other representation and the representation of all components of an avatar including but not exclusive to geometries, textures, skeletons, accessories and semantics.

## **1.4 Related (WG2) and Extracted (new) Requirements**

## **1.5 Relation to other activities (EE, requirements, etc…)**

## **1.6 Mandates**

## **1.7 Participants**

|  |  |  |  |
| --- | --- | --- | --- |
| Participant | Contact | Email | Type |
|  |  |  |  |
| Interdigital | Joao Regateiro | joao.regateiro@interdigital.com | L |
| Qualcomm | Imed Bouazizi | bouazizi@qti.qualcomm.com | P |
| Xidian Univ. | Anthony Trioux | anthony\_trioux@xidian.edu.cn | P |

(P = proponent, L = leader)

## **1.8 Information about proposed technologies**

The following contributions on XXXX have been submitted:

**Meeting #148**

m69421 **Avatar JSON Interchangeable Format**

**Meeting #149**

**m71292 [SD][Avatar][ARF] Avatar Representation Format Data Types.**

This proposal has been accepted for inclusion in the EE2 and is documented below.

1. **Introduction**

In the MPEG#148 meeting, a new part 39 of the ISO/IEC 23090 standard on Coded Representation of Immersive Media was initiated in output document N01398 [1], targeting the definition of an Avatar Representation Format (ARF). In this scope, three Exploratory Experiments (EE) [2] were launched to investigate potential improvements to the initial version of the standard Working Draft (WD).

This contribution addresses EE2 on “Integrating on Geometry Data components for Avatar Data”. It proposes to add enrich the *Data* component with new types that encode the representation of low-level geometry and texture features of the avatar within ARF. It thereby addresses the following high-priority requirements in [3]:

|  |  |  |  |
| --- | --- | --- | --- |
| **Category** | **Req #** | **Description** | **Priority** |
| **Avatar Representation** | **194** | A suitable exchange format for conversion between avatar representation formats in market | High |
|  | **196a** | Representation format includes description of body, skeletal, etc. | High |
|  | **199** | Mesh-based format for representation and animation | High |
| **Storage** | **235** | Base avatar stored in single file | High |
| **236** | Partial access | High |

*Table 1: requirements from [3] addressed by the contribution*

1. **Rationale for embedding a mesh representation format in ARF**

In the current version of the ARF WD [1], the data that define the geometry and texture of the avatar model, *e.g.*, vertex coordinates and UV maps, are represented within the Mesh component as *Data* items. Essentially, as shown on Table 2, a *Data* item references through the *uri* property an encoding of the geometry and texture of the avatar in an external representation format such as glTF binary. This external encoding format is specified as a MIME type by the *type* property.

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Use | Description |
| name | string | M | a string that defines the name of this data. |
| type | string | M | a string that provides the mime type of the data. |
| uri | string | M | a string that defines the data content or reference to the data content depending on type. |
| offset | integer | O | defines the number of bytes used as offset into the data content as pointed to by uri. |
| byteLength | integer | O | defines the number of bytes to use in data content. |
| compression | string | O | an identifier of the compressor used to compress this LoD representation of the mesh. |
| protection | number | O | an identifier of the protection configuration that is applied to encrypt this LoD representation of the mesh. |

*Table 2: Data component specification in the current ARF WD [1]*

**Platforms**

First, updates of the external avatar geometry and texture encoding formats used in the ARF description, such as glTF, currently needs to be managed by the platforms receiving the description. This puts a burden on the platforms by requiring that they can cope with several signaling ways of all the geometry and texture encoding formats they support.

**Not Avatar-Format**

Second, a mesh representation relying on external encoding formats that are not avatar-specific are likely to lack avatar modeling attributes that will be needed in future versions of ARF to improve on the functionality and realism of the avatar representation.

**Skinning**

Third, the ARF WD includes a *skin.weights* property that references an array of floating-point weights associated with every vertex of the mesh. It is unclear how this array of weights can be represented as a Data item. A candidate MIME type would be application/octet-stream, but this type does not provide any information on the structure or typing (e.g., float32, float64) of the data. It is also unclear how a GLB buffer could be used to represent ARF skinning weights, since in glTF the skinning weights and the joints to which they refer are attributes of a mesh primitive. Hence, the GLB buffer would need to contain both mesh, skeleton and skinning information.

**Indexing**

Fourth, there is no robust solution when indexing is required between external data and ARF content. For instance, if the vertices are expected to be ordered in a specific way (like for the storage of skin weights), there are no rule nor guarantee that the vertex order of the mesh in an external file (like .glb) is as expected.

For these reasons, we propose to enrich ARF with an internal representation format, e.g., for the avatar geometry and texture, as an alternative to referencing an external representation format.

1. **Proposed ARF representation format for avatar geometry and texture**

To maintain consistency with the current ARF WD scheme, we propose to represent avatar geometry and texture properties as a hierarchy of JSON objects and encode them as ARF *Data* items. To this purpose, these objects will be assigned types that will be registered by the Internet Assigned Numbers Authority (IANA) as subtypes of a new *model/arf* MIME type.

**Data types**

The objects representing the geometry and appearance of the avatar reference data buffers for storing, *e.g*., the 3D coordinates of the mesh vertices or the pixel intensities of the avatar texture images. To comply with the current design in the ARF WD, these buffersmust be encoded as ARF *Data* objects with dedicated *type* values. Accordingly, we propose to extend the ARF Data component with new data types with additional type-specific properties, highlighted in green in Table 3 below.

The *dense* data type represents regular multi-dimensional arrays.

The *image* data type represents an image, *i.e.*, a 2D array of pixel intensities encoded in a given color space.

The *tuples* data type represents a list of tuples, each tuple having 0 or more values. The tuples in the list may have different value counts, but the data type of all values in all tuples *e.g.*, 16-bit IEEE 754 floating point numbers, must be the same. The *tuples* type could be used, for instance, to encode the faces of a mesh composed of a mix of triangles and quads, as a list containing 3-tuples (for vertex indices of triangular faces) and 4-tuples (for vertex indices of quad faces).

The *sparse* data type represents a sparse tensor encoded as a list of 2-*tuples* (index, value), where index is the location of the value in the tensor. An index is the 1D indexing of cells in a tensor. For tensors with more than one dimension, the 1D indexing is obtained by “flattening” the tensor to a 1D tensor. Sparse tensors could be used, for instance, to encode the skinning weights of the mesh vertices. Indeed, many vertices are bound to fewer than the maximum allowed number of joints, hence have 0 weight entries. As a result, the skinning weight tensor has a large number of 0 entries.

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | Use | Description |
| name | string | M | a string that defines the name of this data. |
| type | string | M | a string that provides the mime type of the data. |
| uri | string | M | a string that defines the data content or reference to the data content depending on type. |
| offset | integer | O | defines the number of bytes used as offset into the data content as pointed to by uri. |
| byteLength | integer | O | defines the number of bytes to use in data content. |
| compression | string | O | an identifier of the compressor used to compress this LoD representation of the mesh. |
| protection | number | O | an identifier of the protection configuration that is applied to encrypt this LoD representation of the mesh. |
| If type is “*model/arf.dense*” { | | | |
| dims | integer [1-\*] | M | A list of integers that define the dimension of the tensor e.g., dimension of [2, 7, 4] refers to a tensor with 2 \* 7 \* 4 = 56 values, where the first element of the tensor has dimension 2, the second element has dimension 7 and the last element has dimension 4. |
| dtype | string | M | A string that defines the type of values e.g., “f32”, “f64”, “i8”. |
| } | | | |
| If type is “*model/arf.sparse*” { | | | |
| dims | integer [1-\*] | M | A list of integers that define the dimension of the tensor e.g., dimension of [2, 7, 4] refers to a tensor with 2 \* 7 \* 4 = 56 values, where the first element of the tensor has dimension 2, the second element has dimension 7 and the last element has dimension 4. | |
| count | integer | M | Number of a 2-tuple (index, value) in data content | |
| dtype | string | M | A string that defines the type of values e.g., “f32”, “f64”, “i8”. | |
| itype | string | M | A string that defines the type of values e.g., “i32”, “i64”. | |
| } | | | |
| If type is “*model/arf.image*” { | | | |
| mimeType | string | O | Image mime type, such as, “image/png” | |
| dims | integer [2] | O | An array of integers with size of 2 that represents the image size (width, height). | |
| ctype | string | O | A string that defines the type of colors e.g., “RGB” or “YUV”. | |
| dtype | string |  | A string that defines the type of values e.g., “f32”, “f64”, “i8”. | |
| } | | | |
| If type is “*model/arf.tuples*” { | | | |
| count | integer | O | An integer that defines the total number of tuples. | |
| dtype | string | M | A string that defines the type of values e.g., “f32”, “f64”, “i8”. | |
| itype | string | M | A string that defines the type of values e.g., “i32”, “i64”. | |
| } | | | |

*Table 3: proposed extension of the Data component for the representation of data types. The added properties are highlighted in green.*

**Representation of avatar geometry and appearance**

Similarly to the encodings as ARF *Data* objects of the new data types presented in the previous section, we propose to introduce in the ARF description new properties for describing the geometry and appearance of an avatar as ARF *Data* objects with dedicated MIME types. For clarity, these new properties are described below as objects. However, it should be understood that **these objects are meant to be encoded, similarly to the new data types in the previous section, as ARF *Data* items with additional object-specific properties that are listed below**. This encoding of new object properties as ARF *Data* items is illustrated in the JSON ARF example description provided in section 4 of this contribution.

At the top of the object hierarchy describing the geometry and appearance of an avatar is the *Geometry* object. This object is encoded as an ARF *Data* component with *type* set to *“model/arf.geometry+json”* and appended to the properties of the ARF *Mesh*, as illustrated on Table 4. The added geometry property is highlighted in green.

Similarly to the encodings as ARF *Data* objects of the new data types presented in the previous section, we propose to introduce in the ARF Data new objects for describing the geometry and appearance of an avatar as ARF *Data* objects with dedicated MIME types.

For clarity, these new properties are described below as objects. However, it should be understood that **these objects are meant to be encoded, similarly to the new data types in the previous section, as ARF *Data* items with additional object-specific properties that are listed below**. This encoding of new object properties as ARF *Data* items is illustrated in the JSON ARF example description provided in section 4 of this contribution.

At the top of the object hierarchy describing the geometry and appearance of an avatar is the *Geometry* object. This object is encoded as an ARF *Data* component with *type* set to *“model/arf.geometry+json”* and appended to the properties of the ARF *Mesh*.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Use** | **Description** |
| name | string | M | The name of the mesh. |
| id | number | M | The identifier of the mesh. |
| path | string | M | A string that represents a hierarchical path that can be used to associate the mesh with a node in the external scene graph. |
| data | array(Data) | M | A reference into a data item that contains the mesh data. |

*Table 4: proposed extension of the Mesh component for the representation of avatar geometry and texture data. The added geometry property is highlighted in green.*

Table 5 defines the semantics of the *Geometry* object. The *vertices* and *normals* properties reference tensors holding respectively the 3D coordinates of the mesh vertices and the vertex normals. The *uvs* property is a list of tensors holding the UV coordinates of the mesh vertices in the set of UV maps associated with the textures that model the avatar appearance.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Description** | **Required** |
| vertices | integer | Index of the corresponding item in the “*data”* collection. The referenced item must be a float tensor [number of vertices, 3]. | ü Yes |
| normals | integer | Index of the corresponding item in the “*data”* collection. The referenced item must be a float tensor [number of vertices, 3]. | No |
| uvs | integer [1-\*] | List of indexes to corresponding items in the “*data”* collection. The ith referenced item must be a float tensor [number of uvs(i), 2]. | No |
| faces | Faces | A *Faces* object. | No |
| textures | Texture [1-\*] | List of *Texture.* | No |

*Table 5: specification of the Geometry object*

The *faces* property in Table 5 references a *Faces* object that defines the vertex indices for each face of the mesh, as well as the corresponding normal vectors and UV coordinates. The *Faces* object is described in Table 6. It is encoded in the ARF description as a *Data* object with type set to *“model/arf.faces+json”*.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Description** | **Required** |
| vertices | integer | References an item in the “*data”* collection. The referenced item must be a float tensor [number of faces, 3-256] or a list of tuples, where tuples must have between 3 and 256 values. | No |
| normals | integer | References an item in the “*data”* collection. The referenced item must be a float tensor [number of faces, 3-256] or a list of tuples, where tuples must have between 3 and 256 values. | No |
| uvs | integer [1-\*] | List of references of items in the “*data”* collection. Each referenced item must be a float tensor [number of faces, 3-256] or a list of tuples, where tuples must have between 3 and 256 values. | No |

*Table 6: specification of the Faces object*

The *textures* property in Table 5 references a list of *Texture* objects that define the textures mapped to the avatar mesh geometry. The specification of the *Texture* object is shown on Table 7. The *Texture* object is encoded in the ARF description as a *Data* object with type set to *“model/arf.texture+json”*.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type** | **Description** | **Required** |
| type | string | Texture type, such as “albedo”, “specular”, etc. | ü Yes |
| uvs | integer | References one of the uvs item in the *“Data”* collection. | ü Yes |
| image | integer | References an item in the “*data”* collection. The referenced item must be an image. | ü Yes |

*Table 7: specification of the Texture object*

**Mapping to MIME types**

The new ARF types introduced above for the encoding of the avatar geometry and appearance will be mapped to subtypes of a new *model/arf* MIME type, as described in Table 8.

|  |  |
| --- | --- |
| **Object type** | **MIME type** |
| Geometry | *model/arf.geometry+json* |
| Faces | *model/arf.faces+json* |
| Texture | *model/arf.texture+json* |
| dense | *model/arf.dense* |
| sparse | *model/arf.sparse* |
| image | *model/arf.image* |
| tuples | *model/arf.tuples* |

*Table 8: mapping of object types introduced in this contribution to future MIME types.*

1. **Example JSON description**

An example ARF avatar description building on the new geometry and texture *Data* items proposed in this contribution is provided below. For conciseness, the parts of the ARF description that do not pertain to the elements introduced in the contribution have been omitted.

|  |
| --- |
| {  […]  **“components”**:  {  **“skeletons”**: [],  **“skins”**: [],  **“meshes”**: [  {  **“name”**: **“full\_body”**,  **“id”**: 42,  **“path”**: **“”**,  **“data”**: [5]  }  ],  **“nodes”**: […],  **“blendshapes”**: […]  },  **“data”**: [  {  **“name”**: **“my\_avatar\_mesh\_vertices”**,  **“type”**: **“model/arf.dense”**,  **“uri”**: **“my\_avatar\_mesh\_vertices.bin”**,  **“offset”**: 0,  **“byteLength”**: 180000,  **“dims”**: [15000, 3],  **“dtype”**: **“f32”**  }, {  **“name”**: **“my\_avatar\_vertex\_normals”**,  **“type”**: **“model/arf.dense”**,  **“uri”**: **“my\_avatar\_vertex\_normals.bin”**,  **“offset”**: 0,  **“byteLength”**: 180000,  **“dims”**: [15000, 3],  **“dtype”**: **“f32”**  }, {  **“name”**: **“my\_avatar\_uvs0”**,  **“type”**: **“model/arf.dense”**,  **“uri”**: **“my\_avatar\_uvs0.bin”**,  **“offset”**: 0,  **“byteLength”**: 120000,  **“dims”**: [15000, 2],  **“dtype”**: **“f32”**  }, {  **“name”**: **“my\_avatar\_faces”**,  **“type”**: **“model/arf.dense”**,  **“uri”**: **“my\_avatar\_faces.bin”**,  **“offset”**: 0,  **“byteLength”**: 252000,  **“dims”**: [21000, 3],  **“dtype”**: **“u32”**  }, {  **“name”**: **“my\_avatar\_albedo”**,  **“type”**: **“model/arf.image”**,  **“uri”**: **“my\_avatar\_albedo.png”**,  **“offset”**: 0,  **“byteLength”**: 414720,  **“mimeType”**: **“image/png”**,  **“dims”**: [720, 576],  **“ctype”**: **“RGB”**,  **“dtype”**: **“i8”**  }, {  **“name”**: **“my\_avatar\_geometry\_description”**,  **“type”**: **“model/arf.geometry+json”**,  **“uri”**: **“my\_avatar\_geometry\_description.json”**,  **“offset”**: 0,  **“byteLength”**: 680,  **“vertices”**: 0,  **“normals”**: 1,  **“uvs”**: 2,  **“faces”**: 6,  **“textures”**: [7]  }, {  **“name”**: **“my\_avatar\_faces\_description”**,  **“type”**: **“model/arf.faces+json”**,  **“uri”**: **“my\_avatar\_faces\_description.json”**,  **“offset”**: 0,  **“byteLength”**: 148,  **“vertices”**: 3  }, {  **“name”**: **“my\_avatar\_texture\_description”**,  **“type”**: **“model/arf.texture+json”**,  **“uri”**: **“my\_avatar\_texture\_description.json”**,  **“offset”**: 0,  **“byteLength”**: 312,  **“type”**: **“albedo”**,  **“uvs”**: 2,  **“image”**: 4  }  ]  } |

The listing starts with the *”components”* component of the ARF description. This component holds a *“meshes”* array that is composed of a single *Mesh* item. The *“data”* item of this single *Mesh* item contains a single ARF *Data* item that is item 5 in the *“data”* array, *i.e.*, the *Data* item with type set to *“model/arf.geometry+json”* that describes the avatar geometry and texture properties.

This *geometry Data* item references *Data* items 6 and 7 that correspond to the *Faces* and *Texture* objects introduced in the contribution. The *geometry*, *Faces* and *Texture Data* items reference dense tensors encoded as *Data* items with indices 0 through 3 that represent, respectively, the coordinates of the mesh vertices in the avatar geometry, the components of the vertex normals, the UV coordinates of the mesh vertices in the avatar texture image, and the indices of the vertices that define each face of the mesh. The *Texture Data* item references the albedo texture image encoded as the *Data* item with index 4 in the *”data”* array.

**5. Conclusion**

It is proposed to add the descriptive elements of avatar geometry and appearance proposed in section 3 of this contribution to the Working Draft of ISO/IEC 23090-39.

**6. References**

[1] N01398, “Avatar Representation Format”, MPEG#148, Kemer, Turkey, November 2024.

[2] N01379, “EE Description of Avatar representation formats”, MPEG#148, Kemer, Turkey, November 2024.

[3] N01370, “Procedures and Tests Formats for Avatars Representation Formats”, MPEG#148, Kemer, Turkey, November 2024.

## **1.9 Extracted from TuC**

## **1.10 Test cases**

The EE may define test cases for which the evaluation criteria will be analyzed. For instance, a first test case  can be with live content while another in the on-demand content.

## **1.11 Evaluation criteria**

List of criteria that will allow to compare the different technical solutions and converge to a unique solution. Criteria can be objective like memory efficiency, bitrate or subjective flexibility, compatibility with legacy solution, etc.

|  |  |  |
| --- | --- | --- |
| **Criteria** | **Description** | **Evaluation** |
| Criteria #1 | Description | The technical solution should minimize/optimize … |

## **1.12 Timeline**

## MPEG #150: Document solutions and evaluation results and recommend changes to the CD text.**1.13 References**

[1] WG03 N01316, “Procedures and Test Scenario for MPEG Avatar Representation”, MPEG#147, Sapporo, Japan, July 2024.

[2] WG02 N00359, “Draft of MPEG-I Phase 3 requirements”, MPEG#146, Rennes, France, April 2024.

[3] WG03 N0XXXX, “WD of ISO/IEC 23090-38 Avatar representation formats”, Kemer, Turkey, November 2024.

[4] WG02 m64008 “Use Cases for MPEG Media Avatar”, Hanover, Germany, October 2023.

# EE3: Other Animation Sample Formats for Animation Stream (ongoing)

## **1.1 Introduction**

This Exploration Experiment (EE) aims to develop structures for animation streams for avatars, focusing on both facial and body animations. The experiment will explore animation sample to be streamed and transmitted for various animation data types, including blendshapes, facial landmarks, animation controllers, and joint transforms, with the goal of establishing a set of standardized approaches that enable efficient and flexible encoding and decoding of avatar animation data across platforms.

## **1.2 Problem statement**

The following statement for Animation samples are identified:

* Standardization of Avatar Animation Data Formats: Propose a standardized format for avatar animation data that is interoperable across platforms and supports a wide range of animation components. The format should be extensible to accommodate new animation parameters as they emerge, and must support compatibility with existing frameworks, such as Mediapipe, ARKit, OpenXR, etc.

## **1.3 Use cases relevant for the EE**

The work of the EE is based on the requirements defined in N00359 and use cases identified in m64008 .

The requirements defined in N00359 are currently organized into five different categories including: Avatar Representation, Coding format, Transport/Synchronization/Carriage, Storage and Privacy and Security requirements.

The requirements related to the “Avatar Representation” are summarized as follows:

* Interoperability: allow converting to and from other avatar representation.
* 3D Scenes: to allow integration into scene description solution.
* Attributes: requirements applicable to the features supported by the avatar, such as, CGI features as textures, geometries, skeletons, accessories, semantic definitions.

The requirements related to the “Coding Format” are summarized as follows:

* General: requirements applicable to the whole format and its use
* Geometry: requirements applicable to the geometry of the base avatar
* Interaction: requirements that relate to the interaction between Avatars and between Avatars and other objects in the scene

* Animation and control: requirements on the facial and body animation

Transport/Synchronization/Carriage: requirements related to the transport of the base avatar model and the associated animation streams

* Storage: requirements that pertain to the storage of the base avatar model
* Security and privacy: requirements on the protection of the avatar in different scenarios

The requirements and use cases explicit mention the interoperability concept to allow converting to and from other representation and the representation of all components of an avatar including but not exclusive to geometries, textures, skeletons, accessories and semantics.

## **1.4 Related (WG2) and Extracted (new) Requirements**

## **1.5 Relation to other activities (EE, requirements, etc…)**

## **1.6 Mandates**

## **1.7 Participants**

|  |  |  |  |
| --- | --- | --- | --- |
| Participant | Contact | Email | Type |
|  |  |  |  |
| Qualcomm | Imed Bouazizi | bouazizi@qti.qualcomm.com | L |
| Xidian Univ. | Anthony Trioux | anthony\_trioux@xidian.edu.cn | P |
| Interdigital | Joao Regateiro | joao.regateiro@interdigital.com | P |

(P = proponent, L = leader)

## **1.8 Information about proposed technologies**

The following contributions on XXXX have been submitted:

**Meeting #148**

**m70326 [SD][Avatar] MPEG Avatar Animation Stream**

**m70195 Avatar facial animation stream format and coding scheme**

**Meeting #149**

**m71326 [SD] Support for eye animation in the Avatar Representation Format**

This proposal has been accepted for inclusion in the EE3 and is documented below.

**1 Introduction**

In this contribution, we provide an overview of the existing eye and gaze tracking functionality in popular XR frameworks and APIs.

**2 Overview of Eye and Gaze Tracking**

Eye and gaze tracking technologies play a crucial role in modern augmented reality (AR) and virtual reality (VR) systems, enabling enhanced user interaction and social presence. Several key extensions and frameworks provide tools for developers to incorporate these functionalities into their applications. This paper delves into the XR\_EXT\_eye\_gaze\_interaction, XR\_FB\_eye\_tracking\_social, XR\_ANDROID\_avatar\_eyes, and ARKit frameworks, focusing on their sample syntax and semantics to illustrate their implementation.

2.1 XR\_EXT\_eye\_gaze\_interaction

The XR\_EXT\_eye\_gaze\_interaction extension in OpenXR facilitates the integration of eye-gaze data into applications, allowing developers to create natural and intuitive interaction mechanisms. By tracking the user's eye movement, applications can infer the user's focus point and enable gaze-based actions, such as selection and cursor positioning.

**2.1.1 Sample Syntax and Semantics**

The core structure of this extension includes:

typedef struct XrEyeGazeSampleTimeEXT {

XrStructureType type; // Specifies the type of this structure.

const void\* next; // Pointer to the next structure in a chain.

XrTime time; // Timestamp of the eye-gaze sample.

} XrEyeGazeSampleTimeEXT;

* **type**: Identifies the structure type (XR\_TYPE\_EYE\_GAZE\_SAMPLE\_TIME\_EXT).
* **next**: A pointer for extension-specific structures or NULL.
* **time**: Represents the precise time of the eye gaze sample.

To retrieve gaze data, the function xrGetEyeGazeSampleTimeEXT is used. This function queries the latest available eye gaze sample time and returns it in the specified structure. Developers use this data to synchronize eye movements with system events or other inputs.

2.2 XR\_FB\_eye\_tracking\_social

The **XR\_FB\_eye\_tracking\_social** extension, created by Meta, is specifically tailored for social VR experiences. It provides detailed data on each eye's gaze, enabling lifelike avatar animations and realistic eye movements.

**2.2.1 Sample Syntax and Semantics**

The key structure for this extension is:

typedef struct XrEyeGazesFB {

XrStructureType type; // Type of this structure.

void\* next; // Pointer to the next structure in a chain.

XrEyeGazeFB gaze[XR\_EYE\_POSITION\_COUNT\_FB]; // Array holding gaze data for each eye.

XrTime time; // Timestamp for the gaze data.

} XrEyeGazesFB;

* **type**: Indicates the structure type (XR\_TYPE\_EYE\_GAZES\_FB).
* **next**: Pointer for extension-specific structures or NULL.
* **gaze**: An array of XrEyeGazeFB structures, holding gaze direction and other data for each eye.
* **time**: A timestamp indicating when the gaze data was captured.

The XrEyeGazeFB structure includes the gaze direction and position for each eye, which are crucial for animating avatars or enabling gaze-based interactions in social VR.

2.3 XR\_ANDROID\_avatar\_eyes

The **XR\_ANDROID\_avatar\_eyes** extension supports avatar eye representation by providing information on eye pose, tracking state, and non-tracked states like blinking. It is designed to make avatars more lifelike in AR/VR applications.

**2.3.1 Sample Syntax and Semantics**

The key structures include:

typedef struct XrEyeStateANDROID {

XrBool32 isTracked; // Indicates if the eye is being tracked.

XrBool32 isBlinking; // Indicates if the eye is blinking.

} XrEyeStateANDROID;

typedef struct XrEyePoseANDROID {

XrPosef pose; // Pose of the eye in the base space.

XrEyeStateANDROID state; // State of the eye (tracked, blinking, etc.).

} XrEyePoseANDROID;

* **isTracked**: A boolean indicating if the eye tracking is active.
* **isBlinking**: A boolean indicating if the eye is blinking.
* **pose**: Contains the position and orientation of the eye within the base space.

Applications use these structures to query eye state and pose data, enabling realistic animation of avatar eyes, including blinking and gaze tracking.

2.4 ARKit Eye Tracking

Apple’s ARKit framework offers robust facial and eye tracking capabilities, enabling developers to create immersive applications that respond to real-time facial expressions and gaze direction. This framework is deeply integrated with Apple’s hardware, ensuring high fidelity.

**Sample Syntax and Semantics**

ARKit provides high-level APIs to access eye tracking data. A key component is the ARFaceAnchor class, which includes blend shape coefficients and eye tracking properties.

if let faceAnchor = anchor as? ARFaceAnchor {

let leftEyeTransform = faceAnchor.leftEyeTransform

let rightEyeTransform = faceAnchor.rightEyeTransform

let lookAtPoint = calculateLookAtPoint(leftEye: leftEyeTransform, rightEye: rightEyeTransform)

}

* **leftEyeTransform** and **rightEyeTransform**: Matrices representing the position and orientation of each eye.
* **lookAtPoint**: A calculated point representing the user’s gaze direction based on eye transforms.

ARKit also supports blend shapes for eye openness, allowing developers to track whether the eyes are open or closed. This data can be used for avatar animation or gaze-based interaction.

**3 Summary and Proposal**

By investigating the existing frameworks for eye and gaze tracking, we can identify the key features that the eye animation sample should contain. All these formats share a common pose for each eye. Additionally, a state of the eye, e.g. blinking, may also be provided.

We propose to work as part of the EE#3 core experiment on defining a sample format for eye animation based on the existing tracking frameworks described above.

**4 References**

[1] Khronos, OpenXR 1.1 Specification, <https://registry.khronos.org/OpenXR/specs/1.1/html/xrspec.html>

[2] Android XR, XR\_ANDROID\_avatar\_eyes OpenXR extension, [XR\_ANDROID\_avatar\_eyes OpenXR extension  |  Android XR  |  Android Developers](https://developer.android.com/develop/xr/openxr/extensions/XR_ANDROID_avatar_eyes)

[3] Apple ARKit, [Tracking and Visualizing Faces | Apple Developer Documentation](https://developer.apple.com/documentation/arkit/arkit_in_ios/content_anchors/tracking_and_visualizing_faces)

Interactivity in Scene Description

## **1.9 Extracted from TuC**

## **1.10 Test cases**

The EE may define test cases for which the evaluation criteria will be analyzed. For instance, a first test case  can be with live content while another in the on-demand content.

## **1.11 Evaluation criteria**

List of criteria that will allow to compare the different technical solutions and converge to a unique solution. Criteria can be objective like memory efficiency, bitrate or subjective flexibility, compatibility with legacy solution, etc.

|  |  |  |
| --- | --- | --- |
| **Criteria** | **Description** | **Evaluation** |
| Criteria #1 | Description | The technical solution should minimize/optimize … |

## **1.12 Timeline**

## MPEG #150: Document solutions and evaluation results and recommend changes to the CD text.**1.13 References**

[1] WG03 N01316, “Procedures and Test Scenario for MPEG Avatar Representation”, MPEG#147, Sapporo, Japan, July 2024.

[2] WG02 N00359, “Draft of MPEG-I Phase 3 requirements”, MPEG#146, Rennes, France, April 2024.

[3] WG03 N0XXXX, “WD of ISO/IEC 23090-38 Avatar representation formats”, Kemer, Turkey, November 2024.

[4] WG02 m64008 “Use Cases for MPEG Media Avatar”, Hanover, Germany, October 2023.

# EE4: Content Discovery and Partial Access (opened)

**1.1 Introduction**

This Exploration Experiment (EE) aims to evaluate solutions to address the issue of content discovery and partial access.

**1.2 Problem statement**

MPEG is currently specifying an Avatar Representation Format (ARF) for the storage and animation of 3D avatars. The ARF specifies an ARF document that provides a detailed description of all the content of the ARF container. The ARF document itself is stored inside the ARF container.

When retrieving or handling ARF containers, e.g. to animate the avatar of another participant in an AR call, the user needs to retrieve only a subset of that ARF container.

Partial access is required for the following reasons:

* ARF container contains more assets than the ones needed and authorized for access during an Avatar session
* ARF container stores the assets in multiple levels of detail, which makes the file size relatively large. A user only needs a selected set of assets at a specific level of detail and wouldn’t need to download all levels of detail.

However, because the ARF document contains a complete description of the assets and levels of detail and it is stored inside the container, there is no way to know and select what subset is needed prior to downloading the full container.

**1.3 Use cases relevant for the EE**

The work of the EE is based on the requirements defined in N00359 and use cases identified in m64008.

**1.4 Related (WG2) and Extracted (new) Requirements**

The following requirements are relevant for this EE:

* Total or partial delivery and presentation of the avatar attributes, depending on the client network and device for instance.
* Store and access the different components and assets of the base model independently in the file,
* enable the protection of parts of the base Avatar against tampering,
* enable the carriage and access to authentication features as part of the supported media streams.

**1.5 Relation to other activities (EE, requirements, etc…)**

**1.6 Mandates**

The mandate of this EE is to develop and evaluate a solution for the discovery of the content of an ARF container and for partial access to it.

**1.7 Participants**

|  |  |  |  |
| --- | --- | --- | --- |
| **Participant** | **Contact** | **Email** | **Type** |
| **Qualcomm** | **Imed Bouazizi** | **bouazizi@qti.qualcomm.com** | **L** |
| **Interdigital** | **Joao, Quentin** | **Joao.Regateiro@InterDigital.com Quentin.Avril@InterDigital.com** | **P** |
| Xidian Univ. | Anthony Trioux | anthony\_trioux@xidian.edu.cn | P |

**(P = proponent, L = leader)**

**1.8 Information about proposed technologies**

The following contributions on Content Discovery and Partial Access have been submitted:

Meeting #149

m71327 ARF Manifest

**1.9 Extracted from TuC**

**1.10 Test cases**

The EE may define test cases for which the evaluation criteria will be analyzed. For instance, a first test case can be with live content while another in the on-demand content.

**1.11 Evaluation criteria**

List of criteria that will allow to compare the different technical solutions and converge to a unique solution. Criteria can be objective like memory efficiency, bitrate or subjective flexibility, compatibility with legacy solution, etc.

|  |  |  |
| --- | --- | --- |
| **Criteria** | **Description** | **Evaluation** |
| Criteria #1 | Discovery | The technical solution should facilitate the discovery of the content of an ARF container. |
| Criteria #2 | Partial Access | The technical solution should enable the partial access to the ARF container. |
| Criteria #3 | Authorization | The technical solution should enable the ARF owner to grant partial access to ARF container. |

**1.12 Timeline**

MPEG #150: Document solutions and evaluation results and recommend changes to the CD text.

**1.13 References**

[1] WG03 N01316, “Procedures and Test Scenario for MPEG Avatar Representation”, MPEG#147, Sapporo, Japan, July 2024.

[2] WG02 N00359, “Draft of MPEG-I Phase 3 requirements”, MPEG#146, Rennes, France, April 2024.

[3] WG03 N01398, “WD of ISO/IEC 23090-39 Avatar representation formats”, Kemer, Turkey, December 2024.

[4] WG02 m64008 “Use Cases for MPEG Media Avatar”, Hanover, Germany, October 2023.

# EE5: Animation Controllers (opened)

**1.1 Introduction**

This Exploration Experiment (EE) aims to evaluate and develop avatar animations based on controllers. This animation technique is an extension of blendshapes (or morph targets) which allows the deformation of the avatar mesh given a set of weights.

Controllers are widely used in the content creation industry, since they can allow complex animations given few parameters. This parameter reduction compared to other techniques like blendshapes also reduces the memory footprint or bandwidth of animations parameters.

Controllers can be encoded in the MPEG Avatar Representation Format (ARF) [1] using widely used and proven approaches based on channels encoding.

## Deformation function depending on a weight

Controllers are features widely used in industry to animate rigs, including humanoid rigs. They are, to some extent, enhanced morph targets (or blendshapes). Like morph targets, a weight is attached to each controller: a zero weight has usually no effect, a positive weight starts the modification of the shape and a negative weight can have an inverse modification. There is also a weight range for each controller: the amount of modification is controlled and cannot go beyond predefined values.

Mathematically speaking, a controller can be expressed as a function which, given a mesh *M* and a weight *w*, returns a new mesh *M’*:

However, contrary to morph targets, modifications are not limited to a weighted addition of vertex displacements (or a linear combination of meshes in the case of blendshapes). In fact, a controller can perform any modification on a mesh given a weight. These modifications are not limited to the vertex coordinates and can also modify the texture for instance. Furthermore, the effect of the weight can be non-linear, contrary to morph targets, which only supports linear weight combinations.

In a way, a controller looks like an animation, except that time is replaced by weight, and a weight can be negative. An animation can be seen as a (possibly non-linear) function of time which maps a modification of the mesh. An animation can, depending on the encoding format, make many different modifications, like changing the morph target weights or changing the transform of a node. However, animations are made to do modifications from time 0 to the end of its time range. They are not supposed to be run with random time values. Furthermore, animations are not made to be combined, and especially with different random time positive values. On the other hand, controllers, like morph targets, are made to be combined with different random weight values (positive or negative).

Figure 1 illustrates the combination of two controllers:

1. The first controller updates two morph targets and two joint transforms:
   1. The first morph target moves eyelids to the left
   2. The second morph target moves eyelids to the right
   3. The first joint transform rotates the left eye around axis Y
   4. The second joint transform rotates the right eye around axis Y.
2. The second controller also updates two morph targets and two joint transforms:
   1. The first morph target moves eyelids to the up
   2. The second morph target moves eyelids to the bottom
   3. The first joint transform rotates the left eye around axis X
   4. The second joint transform rotates the right eye around axis X.

Note that each controller performs four updates (two morph targets and two rotations) given a single weight value. Without a controller, 10 values are required: one for each morph target and four for each rotation.

A collage of a person's face

Description automatically generated

Figure 1 - Combining two controllers to animate the avatar gaze.

The first controller animates the eyes from left to right using weight .  
The center row of Figure 1 shows the avatar state when weight (left), when weight (center) and when weight (right).

The second controller animates the eyes from bottom to top using weight .  
The center column of Figure 1 shows the avatar state when weight (bottom), when weight (center) and when weight (top).

These two controllers can be used together to create numerous gazes. For example, the top-right avatar in the figure shows a state when both controllers have the same weight, e.g., and . The top-left avatar in the figure is for first controller weight and second controller weight .

## **Use in Asset Creation**

Controllers are widely used and more than commonplace in the industry of visual effects, animation and gaming. They provide additional control levels to artists. Their use has now become mandatory for a better, simpler and quicker animation process.

A close-up of a head

Description automatically generatedA grey statue of a person

Description automatically generated A grey statue of a person with his mouth open

Description automatically generated A close-up of a person's head

Description automatically generated A grey statue of a person

Description automatically generated

Figure 2: Professional facial rig used in VFX industry. Facial controllers panel is on the right and quickly enables to create realistic facial expressions as depicted in the bottom pictures.

When making a picture film, a game or any virtual experience, characters’ assets go through a long and complex asset creation pipeline. Excluding character designers and storytelling artists, modelers (modeling artists) are, first, sculpting blendshapes, considered as the lowest level of control for animation. An industrial consensus has converged for a few decades to the use of the FACS (Facial Action Coding System) for the creation of facial blendshapes because movements of individual facial muscles are encoded by the FACS. Manipulating FACS blendshapes to create facial expressions is time-consuming, not intuitive and not ergonomic. This is the reason why riggers (rigging artists) are, in a second step, setting up the asset rig on top of the mesh and blendshapes created previously. The rig contains all the elements that will be used to deform the assets’ mesh (skeleton + skinning, controllers etc.). This is where the controllers play a key role in aggregating blendshapes, defining new interpolations, linking skeleton joint and blendshapes etc.

## **Benefits of Controllers**

Compared to blendshapes and joint transforms, controllers have the following advantages:

* **Reduced bandwidth**. Transmitting the parameters of an animation using controllers requires less memory than blendshapes and joint transforms, since a single controller can update several blendshapes and joint transforms. In the previous example, each controller (1 parameter) updates two blendshapes (2 parameters) and 2 rotations (8 parameters if using quaternions, 32 if using matrix transforms).
* **Genericity**. Controllers can be tuned to control high-level animations and their behavior be updated without the need for changing the input parameters. For example, controllers can be defined to rotate the elbow. In a usual case, they only update a joint transform. But if, for some reason, additional modifications are required, there is no need to change the user input and signal to transmit.   
  For example, controllers can also use blendshapes to show the activity of muscles if the avatar lifts something heavy. Without controllers, the user must send updates for the muscles blendshapes: the joint transform is not generic enough to know how to tune such blendshapes without additional knowledge. With controllers, the user only has to send the parameters for the controllers, and the avatar can show additional update.
* **Semantics**. Controllers can be designed for high level changes of an avatar which requires the combination of several updates. For example, controllers can be defined to change the physical properties of an avatar (weight, size, etc.). The meaning of such controllers is understandable by any user which can use them to update their avatar. For example, a user who wishes to reflect the fact that (s)he has gained or lost weight could use a dedicated controller.

**1.2 Problem statement**

The following statements for animation controllers are identified:

* Providing precision and flexibility needed to create realistic and engaging avatar movements
* Providing high-level control of avatar’s animations
* Providing high-level description of human characteristics (emotions, motions, appearance, morphology etc.)
* Providing use cases and needs for real-time animation

**1.3 Use cases relevant for the EE**

The work of the EE is based on the requirements defined in N00359 [2] and use cases identified in m64008 [4].

**1.4 Related (WG2) and Extracted (new) Requirements**

* The following requirements, based on the document N0347 [5] are relevant for this EE:
* Facial and body animation, including facial expressions due to communication or social interactions and avatar motion in scenes.
* High and low-level control mechanisms to efficiently animate the avatar and efficiently transmit animation parameters over the network (between server and client).

**1.5 Relation to other activities (EE, requirements, etc…)**

**1.6 Mandates**

**1.7 Participants**

|  |  |  |  |
| --- | --- | --- | --- |
| **Participant** | **Contact** | **Email** | **Type** |
| Interdigital | Quentin Avril | quentin.avril@interdigital.com | L |
| Qualcomm | Imed Bouazizi | bouazizi@qti.qualcomm.com | P |
| Xidian Univ. | Anthony Trioux | anthony\_trioux@xidian.edu.cn | P |

(P = proponent, L = leader)

**1.8 Information about proposed technologies**

The following contributions have been submitted:

* **Meeting #147**
  + **m68986** [42.2] [SD] – Avatar JSON Interchangeable Format
* **Meeting #148**
  + **m69421** Avatar JSON Interchangeable Format
  + **m69474** Avatar JSON Interchangeable Format high level overview
  + **m69936** Draft Procedures and Test Scenarios for MPEG Avatar Representation Format as part of MPEG-I
* **Meeting #149**
  + **m71239** Animation Controllers in ARF (Avatar Representation Format)

**1.9 Extracted from TuC**

**1.10 Test cases**

The EE may define test cases for which the evaluation criteria will be analyzed. For instance, a first test case can be with live content while another in the on-demand content.

**1.11 Evaluation criteria**

List of criteria that will allow to compare the different technical solutions and converge to a unique solution. Criteria can be objective like memory efficiency, bitrate or subjective flexibility, compatibility with legacy solution, etc.

|  |  |  |
| --- | --- | --- |
| **Criteria** | **Description** | **Evaluation** |
| Criteria#1 | Description | The technical solution should provide a high-level description of human characteristics (motion, emotion etc.) |
| Criteria#2 | Control | The technical solution should provide a high-level control of human animation |

**1.12 Timeline**

**2022-03-30**: MPEG #150: Evaluation documents and changes to the Committee Draft text.

**1.13 References**

[1] WG03 N01316, “Procedures and Test Scenario for MPEG Avatar Representation”, MPEG#147, Sapporo, Japan, July 2024.

[2] WG02 N00359, “Draft of MPEG-I Phase 3 requirements”, MPEG#146, Rennes, France, April 2024.

[3] WG03 N0XXXX, “WD of ISO/IEC 23090-38 Avatar representation formats”, Kemer, Turkey, November 2024.

[4] WG02 m64008 “Use Cases for MPEG Media Avatar”, Hanover, Germany, October 2023.

[5] ISO/IEC JTC 1/SC 29/WG02 N0347 “Requirements and state of the art for MPEG media avatar”, Online, January 2024