1 Scope

This document presents recommended practices regarding the TV 3.0 Closed Signing, defined in [1].

2 References

The following documents are cited in the text in such a way that their contents, in whole or in part, constitute requirements for this document. For dated references, only the editions cited apply. For undated references, the most recent editions of that document (including amendments) apply.

ABNT NBR 25606, *TV 3.0 – Closed Signing.*

3 Abbreviations

For the purposes of this Document, the following abbreviations apply.

BVH	BioVision Hierarchy
DASH	Dynamic Adaptive Streaming over HTTP
DTT	Digital Terrestrial Television
glTF	graphics library Transmission Format
IMSC	Internet Media Subtitles and Captions
ISOBMFF	International Organization for Standardization Base Media File Format
JSON	JavaScript Object Notation

4 Guideline organization

This guideline covers four strategies for sign language transmission. The operational guidelines corresponding to the technologies used in the TV 3.0 Closed Signing are contained in the following Annexes:

- Annex A contains the Sign Language Video Stream guidelines (future development);
- Annex B contains the Sign Language Gloss guidelines;
- Annex C contains the Closed Caption Streaming for automatic translation and adaptation to Sign Language guidelines;
- Annex D contains the Sign Language Motion Stream guidelines;.

In order to streamline the documentation and prevent repetition, all sign language representation guidelines are hosted in **Annex R**. Both Annex B and Annex C refer to this central document for linguistic and structural specifications.

Annex R

Sign Language Representation

This Annex defines the standardized framework for the digital representation and structuring of sign language data within the TV 3.0 ecosystem. Its primary objective is to bridge the gap between linguistic nuances and technical implementation, ensuring that sign language content—whether human-captured or synthetic—maintains its grammatical integrity during transmission and rendering. By establishing a uniform syntax for glosses and metadata, this document ensures interoperability across different platforms and decoding devices, ultimately guaranteeing an accessible and high-quality experience for the deaf community.

R.1 Rules for Representing Sign Languages Glosses

This section specifies the rules for representing sign language glosses. These rules are necessary to specify grammatical aspects essential for sign languages and to allow for proper decoding of glosses in the sign language player.

It is essential to mention that these rules were developed for Brazilian Sign Language, but they are easily adaptable to any other sign language.

R.1.1 Homonyms and homographs

In this representation, words that have identical spelling in the written language (for example, Brazilian Portuguese) and different signs in the sign language, which are called homonyms, must receive a descriptive nomenclature at the end of the glosses, representing the context of the sign. The character “&” shall be used to represent the disambiguation.

Tables 1 and 2 illustrate the homonyms COLAR and APAGAR in Brazilian Portuguese, and the variations in the Brazilian Sign Language using the proposed rule. Tables 3 and 4 illustrate some examples of glosses using these variations.

Table 1 - Homonyms COLAR and its variations in Brazilian Sign Language

HOMONYMS	**VARIATIONS (GLOSS REPRESENTATION)**	
COLAR	COLAR&ACESSÓRIO (NECKLACE&ACCESSORY)	COLAR&GRUDAR (PASTE&STICK)
	COLAR&FILAR (CHEAT&CHEAT)	COLAR&INFORMÁTICA (PASTE&INFORMATICS)

Table 2 - **Homonyms APAGAR and its variations in Brazilian Sign Language**

HOMONYMS	**VARIAÇÕES**		
APAGAR	APAGAR&EXTINGUIR (DELETE&EXTINGUIS)	APAGAR&FOGO (PUT_OUT&FIRE)	APAGAR&INFORMÁTICA (DELETE&INFORMATICS)
	APAGAR&ESCREVER (ERASE&WRITE)	APAGAR&LOUSA (ERASE&BLACKBOARD)	APAGAR&LUZ (TURN_OFF&LIGHT)
	APAGAR&VELA (TURN_OFF&CANDLE)		

Table 3 - Examples of using the homonym COLAR and its variations.

INPUT SENTENCE	**RULE APPLIED**
Ganhei um colar da minha vó. (I got a necklace from my grandmother.)	GANHAR COLAR&ACESSÓRIO MEU VÓ [PONTO] (GET NECKLACE&ACCESSORY MY GRANDFATHER [PERIOD])
É errado colar na prova. (It's wrong to cheat on the test.)	ERRADO COLAR&FILAR PROVA [PONTO] WRONG CHEAT&CHEAT TEST [PERIOD]
O papel está todo colado. (The paper is all glued together.)	PAPEL COLAR&GRUDAR [PONTO] (PAPER PASTE&STICK [PERIOD])
Copie e cole o texto. (Copy and paste the text.)	COPIAR COLAR&INFORMÁTICA TEXTO [PONTO] (COPY PASTE&NFORMATICS TEXT [PERIOD])

Table 4 - Examples of using the homonym APAGAR and its variations

INPUT SENTENCE	GLOSS REPRESENTATION
Cuidado, apague o fogo! (Be careful, put out the fire!)	CUIDADO APAGAR&FOGO [EXCLAMAÇÃO] (CAREFUL, PUT_OUT&FIRE [EXCLAMATION])
Apague essa sentença. (Delete this sentence.)	APAGAR&INFORMÁTICA SENTENÇA [PONTO] (DELETE&INFORMATICS SENTENCE [PERIOD])
Apague meu nome da lista. (Delete my name from the list.)	APAGAR&ESCREVER NOME LISTA[PONTO] (ERASE&WRITE NAME LIST [PERIOD])
Posso apagar o quadro? (Can I erase the painting?)	EU PODER APAGAR&LOUSA [INTERROGAÇÃO] (I CAN ERASE&BLACKBOARD [QUESTION])
Entre e apague a luz. (Come in and turn off the light.)	ENTRAR APAGAR&LUZ [PONTO] (COME IN TURN_OFF&LIGHT)
Apague as velas e faça um pedido. (Extinguish the candles and make a wish.)	APAGAR&VELA FAZER PEDIDO&PEDIR [PONTO] (TURN_OFF&CANDLE MAKE ORDER&ORDER [PERIOD])

This rule enables correct translation considering the semantics and pragmatics of sign languages, as it allows the disambiguation of homonyms and homographs since, in the translation process, some signs may be more specific. Therefore, the translation must consider the meaning of the lexical item of the written language for a corresponding lexical item in sign language.

R.1.2 Punctuation marks

In the notation used for representing the sign language glosses, the punctuation identifying characters (“.”, “!” and “?”) shall be replaced by the spelling of the name in full between square brackets. Table 5 shows the representation of punctuation marks in this representation.

Table 5 - Punctuation marks

PUNCTUATION MARK	GLOSS REPRESENTATION
Period: .	[PONTO] OR [PERIOD]
Exclamation Mark: !	[EXCLAMAÇÃO] OR [EXCLAMATION]
Question Mark: ?	[INTERROGAÇÃO] OR [INTERROGATION]

R.1.3 Adjectives: Unification of the gender of adjectives

In sign languages, adjectives are generally in the neutral form. Therefore, there is no marker for gender (masculine and feminine) nor for number (singular and plural) (BRITO, 2013, p. 63). Thus, in this gloss representation, the adjectives shall be inflected in the masculine to simplify the computational process.

Table 6 presents one example of the use of this rule involving adjectives.

Table 6 - Adjectives

INPUT SENTENCE	GLOSS REPRESENTATION
Essa flor é cheirosa! (This flower is smelly!)	ESSE FLOR CHEIROSO [EXCLAMAÇÃO] (THIS FLOWER SMELLY [EXCLAMATION])

R.1.4 Numerical Representations

In sign languages, there are different ways to present numerals when used as cardinals, ordinals, quantity, measurement, age, days of the week or month, hours, and monetary values. (FELIPE, 2007, p. 43). Concerning cardinal numbers, the rule is that notation should always use the numeral (algarism) as a reference (see Table 9). It is important to note that the signing of the sentence presented in Table 7 will be performed using the sign corresponding to the number 4, , not its fingerspelling the name of the number.

Table 7 - Cardinal Numbers

INPUT SENTENCE	GLOSS REPRESENTATION

Eu preciso comprar 4 cadernos. (I need to buy 4 notebooks.)	EU PRECISAR COMPRAR 4 CADERNO [PONTO] (I NEED TO BUY 4 NOTEBOOKS [PERIOD])

Ordinal numbers are expressed in their full notation with the addition of the disambiguation identifier (“&”) to the ordinal numerals (from the first to the fifth number) since they are homonymous words in some written languages (e.g., Brazilian Portuguese). In addition, ordinal numbers are signed similarly to cardinal numbers starting from the 10th (tenth). Table 8 shows the application of the rule for ordinal numbers.

Table 8 - Ordinal numbers

SYMBOL	GLOSS REPRESENTATION
1º	PRIMEIRO&ORDINAL (FIRST&ORDINAL)
2º	SEGUNDO&ORDINAL (SECOND&ORDINAL)
3º	TERCEIRO&ORDINAL (THIRD&ORDINAL)
4º	QUARTO&ORDINAL (FOURTH&ORDINAL)
5º	QUINTO&ORDINAL (FIFTH&ORDINAL)
6º	SEXTO (SIXTH)
7º	SÉTIMO

	(SEVENTH)
8º	OITAVO (EIGHTH)
9º	NONO (NINETH)
10º	10
11º	11
12º	12
20º	20

When a number refers to a quantity, specific signs must be used in sign language. Regarding time, for example, there are two different ways of referring to time in sign languages: chronological or duration (FELIPE, 2007, p. 78). Thus, specific signs must be used for duration. Thus, it is possible to disambiguate the numeral, differentiating a quantitative numeral from a representative one (see Table 9).

Table 9 - Numbers and quantity

INPUT SENTENCE	GLOSS REPRESENTATION
1 hora (1 hour)	UM_HORA (ONE_HOUR)
1 pessoa (1 person)	UM_PESSOA (ONE_PERSON)

To represent time, common abbreviations present in written languages must be treated and described using the “HORA” (HOUR), “MINUTO” (MINUTE), “SEGUNDO” (SECOND), “MANHÃ” (MORNING), “TARDE”

(AFTERNOON) and “NOITE” (NIGHT) glosses, among others. Table 10 presents two examples of time treatment in gloss.

Table 10 - Time period

INPUT SENTENCE	GLOSS REPRESENTAION
São 3h30. (It is 3:30 am.)	3 HORA 30 MINUTO MANHÃ [PONTO] (3 HOUR 30 MINUTE MORNING [PERIOD])
A reunião será às 14h30. (The meeting will be at 2:30 pm.)	REUNIÃO 2 HORA 30 MINUTO TARDE [PONTO] (MEETING 2 HOUR 30 MINUTES AFTERNOON [PERIOD])

It is also common in sign languages to have a different numerical representation for fractions, percentages, and currency. In this notation, the following representation will be adopted:

- **Fraction:** The fraction symbol will be replaced by the “FRAÇÃO” (FRACTION) gloss;
- **Percentage:** The percentage symbol % will be replaced by the “PORCENTAGEM” (PERCENTAGE) gloss;
- **Games or disputes:** The X symbol will be replaced by the “VERSUS” (VERSUS) gloss;
- **Currencies:** The currency symbol will be replaced by its full representation of the currency (e.g.: R$ -> “REAL&MOEDA” (REAL&COIN) and US$ = “DOLLAR”).

Table 11 presents some examples of usage involving these numerical representations.

Table 11 - Examples of different numerical representations

INPUT SENTENCE	GLOSS REPRESENTATION
Temos ⅔ de aprovação.	TER 2 FRAÇÃO 3 APROVAR [PONTO]

(We have ⅔ approval.)	(WE HAVE 2 FRACTION 3 APPROVE [PERIOD])
A música teve 25,5% de rejeição. (The song had 25.5% rejection.)	MÚSICA TER 25 VÍRGULA 5 PORCENTAGEM REJEIÇÃO [PONTO] (MUSIC HAVE 25 COMMA 5 PERCENTAGE REJECTION [PERIOD])
Fluminense ganhou de 4x2 do Vasco. (Fluminense won 4x2 against Vasco.)	FLUMINENSE GANHAR 4 VERSUS 2 VASCO [PONTO] (FLUMINENSE WIN 4 VERSUS 2 VASCO [PERIOD])
Eu preciso de R$ 4,99. (I need R$4.99.)	EU PRECISAR 4 REAL&MOEDA 99 CENTAVOS [PONTO] (I NEED 4 REAL&COIN 99 CENTS [PERIOD])

R.1.5 Compound words or Expressions

Compound words or expressions represented by a single sign must be separated by _ (underscore). For some words, it is important to mention that words in written languages are composed of more than one element. However, when in sign languages, they are considered as simple signs. (NASCIMENTO, 2011, p. 52). As a rule, this notation will also be applied to greetings which are represented as a simple sign (see Table 12).

Table 12 - Compound words

INPUT SENTENCE	GLOSS REPRESENTATION
Comprei um novo guarda-roupa. (I bought a new wardrobe.)	COMPRAR GUARDA_ROUPA NOVO [PONTO] (BUY NEW WARDROBE [PERIOD])
Bom dia! (Good morning!)	BOM_DIA [EXCLAMAÇÃO] (GOOD_MORNING [EXCLAMATION])
Boa tarde! (Good afternoon!)	BOA_TARDE [EXCLAMAÇÃO] (GOOD_AFTERNOON!)
Boa noite! (Good Night!)	BOA_NOITE [EXCLAMAÇÃO] (GOOD_NIGHT [EXCLAMATION])

R.1.6 Verbs with number-personal agreement

In directional verbs (or verbs with gender agreement for the person) in sign languages, the agreement between sender and receiver must be respected. In this sense, the notation of verbs in their infinitive form receives the characters *S (“*” identifies the number and “S” identifies the singular) or *P (“*” identifies the number and “P” identifies the plural), as nomenclatures of direction. Thus, we have the following representation for personal pronouns:

- EU (I) => 1S;
- TU/VOCÊ (YOU) => 2S;
- ELE/ELA (HE/SHE) => 3S;
- NÓS (WE) => 1P;
- VÓS/VOCÊS (YOU) => 2P;
- ELES/ELAS (THEY) => 3P.

Tables 13 present examples of the treatment of verbs with gender agreement in the singular and plural, respectively.

Table 13 - Examples of using verbs with number-person agreement

INPUT SENTENCE	GLOSS REPRESENTATION
Eu perguntei a sua idade. (I asked your age.)	1S_PERGUNTAR_2S IDADE [PONTO] (1S_ASK_2S AGE [PERIOD])
O que você me perguntou? (What did you ask me?)	2S_PERGUNTAR_1S QUE [INTERROGAÇÃO] (2S_ASK_1S WHAT [QUESTION])
Ela perguntou meu nome. (She asked my name.)	3S_PERGUNTAR_1S NOME [PONTO] (3S_ASK_1S NAME [PERIOD])
Ele perguntou a ela sua idade. (He asked her age.)	3S_PERGUNTAR_3S IDADE [PONTO] (3S_ASK_3S AGE [PERIOD])
Nós perguntamos a sua idade. (We asked your age.)	1P_PERGUNTAR_2S IDADE [PONTO] (1P_ASK_2S AGE [PERIOD])

Vocês perguntam a minha idade. (You ask my age.)	2P_PERGUNTAR_1S IDADE [PONTO] (2P_ASK_1S AGE [PERIOD])
Eles perguntaram a sua idade. (They asked your age.)	3P_PERGUNTAR_2S IDADE [PONTO] (3P_ASK_2S AGE [PERIOD])

R.1.7 Incorporation of Negation

The incorporation of negation initially applies to verbs when the negation adverb is modified in the direct or indirect object. In this case, it is possible to simplify by saying that if the NEGATION signing can be done simultaneously with a word (verb), this specific rule must be applied. In negative sentences, the movement of the head (denying) and facial expressions are mandatory to mark negative sentences, as they are directly linked to syntactic issues; otherwise, the sentence will become ungrammatical. (ALMEIDA; ALMEIDA, 2012, p. 628).

In this gloss notation, the incorporation of negation is done by incorporating the prefix “NÃO_” (NOT_) into the sign, to facilitate the reading of the description. Table 14 presents two examples of the application of this rule.

Table 14 - Examples of using incorporation of negation

INPUT SENTENCE	**GLOSS REPRESENTATION**
Eu não posso mais esperar. (I can't wait anymore.)	EU NÃO_PODER ESPERAR [PONTO] (I NOT_CAN WAIT [PERIOD])
Ele não consegue ganhar. (He can't win.)	ELE NÃO_CONSEGUIR GANHAR [PONTO] (HE NOT_CAN WIN [POINT])

R.1.8 Amplification and reduction of intensity

Intensifiers are used when the word, verb or expression is close to some adverb of intensity. This intensity is characterized by the sign's duration, energy, variance, and average speed (PASSOS, 2014, p.19). In this representation, the markers "(+)" or "(-)" should be inserted at the end of the word to represent the intensifier. In this case, the adverbs of intensity "muito" (very), "mais" (more), "muitíssimo" (very much) and their synonyms will be represented by the marker "(+)" at the end of the word. In contrast, the adverbs "pouco" (little), "menos" (less), “pouquíssimo" (very little) and their synonyms will be represented by the marker "(-)" at the end of the word.

Table 15 presents some examples of the application of the rule in sentences with increased and reduced intensity.

Table 15 -Examples of using amplification and reduction of intensity

INPUT SENTENCE	RULE APPLIED
Ele está muito deprimido. (He is very depressed.)	ELE DEPRIMIDO(+) [PONTO] (HE DEPRESSED(+) [PERIOD])
Ela é tão bonita! (She is so pretty!)	ELE BONITO(+) [EXCLAMAÇÃO] (HE BEAUTIFUL(+) [EXCLAMATION])
Você está nervoso demais. (You are too nervous.)	VOCÊ NERVOSO(+) [PONTO] (YOU NERVOUS(+) [PERIOD])
Estou um pouco doente. (I am a little sick.)	EU DOENTE(-) [PONTO] (I SICK(-) [DOT])

R.2 Accessing Sign Animations via URL

This section outlines the standard URL format to access and download sign animations from the public dictionary. The link is constructed using the sign's unique gloss, allowing applications to programmatically retrieve the correct animation file for integration and playback.

As an example, the animation for the CASA gloss can be found at: https://dicionario2.vlibras.gov.br/2018.3.1/WEBGL/CASA

The full list of signs available in the dictionary can be viewed here: https://dicionario2.vlibras.gov.br/bundles

Annex A

Sign Language Video Stream

[Annex A is under development in a separate document. This is just a placeholder to add its contents when completed.]

Annex B

Sign Language Gloss

B.1 Scope

This document specifies the workflow for sign language integration when translation is performed **prior to transmission** (Source-side). In this workflow, the sign language gloss content is generated and packaged at the broadcast station.

B.2 Sign Language Representation

The sign language representations and glossing rules used in this technology are governed by the specifications detailed in **Annex R (Sign Language Representation)**.

B.3 Operational Workflow

The broadcaster operates by multiplexing and transmitting audio, video, and a sequence of sign language glosses in **IMSC1** format via **Over-the-Top (OTT)** or **Over-the-Air (OTA)** signals. Upon reception, a registered application accesses the stream through the API mechanism specified in **ABNT NBR 25608, Annex C**.

In this **Receiver-side (Client-side)** architecture, the application performs the **rendering for each sentence** of glosses contained within tags, which are identified by the attribute xml:lang="bzs". The sign dictionary and the sign player components are fully embedded and executed within the application, running either on the **TV 3.0 receiver** or a **connected companion device**.

Annex C

Closed Caption Streaming for automatic translation and adaptation to Sign Language

C.1 Scope

This document specifies the workflow for sign language generation when translation is performed after reception (Receiver-side). In this workflow, the translation process-often involving synthetic sign language generation (avatars) or dynamic data rendering-is executed directly Digital Television Receiver or on a connected second-screen device.

C.2 Sign Language Representation

The sign language representations and glossing rules used in this technology are governed by the specifications detailed in **Annex R (Sign Language Representation)**.

C.3 Operational Workflow

The broadcaster operates by multiplexing and transmitting audio, video, and written-language closed captioning via **Over-the-Top (OTT)** or **Over-the-Air (OTA)** signals. Upon reception, a registered application accesses the closed caption stream through the API mechanism specified in **ABNT NBR 25608**.

At **Receiver-side (Client-side)** architecture, the application performs the automatic translation into sign language locally. Following this translation, the application handles the **rendering for each generated sentence**. The sign language translator, the sign dictionary, and the sign player components are fully embedded and executed within the application, running either on the **TV 3.0 receiver** or a **connected companion device**.

Annex D

Sign Language Motion Stream

D.1 Scope

This document defines the operational guidelines for transmission of sign language motion stream through TV 3.0 broadcasting signal, aiming to provide sign language interpretation to DTT receiver using a three-dimensional humanoid avatar. This transmission is specified in ABNT NBR 25606 , Annex D.

D.2 Motion file format conversion

D.2.1 Scope

This guideline assumes that the SLMB format defined in ABNT NBR 25606 , D.5, is not supported by any known animation software in the market. So, it proposes a detailed description of the methods to convert a SLMB file to some known motion file formats, and vice-versa.

D.2.2 Parameters

In the conversion pseudo-codes related to SLMB motion blocks, it is assumed that

- the parameters starting with `geo` are from body geometry
  - the `x`/`y`/`z` components of the vectors `geo.refpose_from_parent` (position of joint from parent in reference pose) and `geo.refpose_end` (position of end of joint in reference pose) in a joint can be gotten from ABNT NBR 25606 , Table D.3.
  - the `x`/`y`/`z` components of the vectors `geo.RX`, `geo.RY` and `geo.RZ` in type-2 and type-3 joints (coordinates of the rotation axes) can be gotten from ABNT NBR 25606 , Table D.5.
- the parameters starting with `bmb` are from the `BodyMotionBlock` structure, as defined in ABNT NBR 25606 , Table D.8.
- the parameters starting with `fmb` are from the `FaceMotionBlock` structure, as defined in ABNT NBR 25606 , Table D.10.

D.2.3 Rotation format conversion

As stated in ABNT NBR 25606 , D.3.4, joint rotation can be defined by Euler angles or by quaternions. Each motion file format declares joint rotation using one of these definitions. SLMB files applies quaternions in some joints and Euler angles in other joints, using x/y/z rotation sequence. So, for some joint rotation movements, conversion between body motion formats requires conversion from Euler angles to quaternion and vice-versa.

For these conversion codes the following parameters and functions are assumed:

- `Ex`, `Ey` and `Ez` are Euler angles around x, y and z rotation axes, respectively.

- `RX`, `RY` and `RZ` are the normalized coordinates of the x, y and z rotation axes, respectively. They are represented by the `x`, `y` and `z` elements, referring to the x/y/z coordinates of the rotation axes.
- `qw`, `qx`, `qy` and `qz` are the elements of the quaternion.
- `wrap_to_degree()` function ensures that the angles are converted to degrees and stays in (-180°, 180°] interval.

Previously, it is necessary to define rotation matrix from coordinate axis to rotation axis.

```
rotationaxisToquaternion(Rx, Ry, Rz)

{

    qw = sqrt(1 + Rx.x + Ry.y + Rz.z) / 2

    qx = (Ry.z - Rz.y) / (4*qw)

    qy = (Rz.x - Rx.z) / (4*qw)

    qz = (Rx.y - Ry.z) / (4*qw)

    return (qw, qx, qy, qz)

}
```

The conversion from Euler angles in y/x/z sequence (adopted in most BVH formats) to quaternion can be done by applying three Euler rotations in y/x/z sequence.

```
euler2quaternion_yxz(Ex, Ey, Ez, RX, RY, RZ)

{

    Qr = rotationaxisToquaternion(Rx, Ry, Rz)

    Qex = (cos(Ex/2), sin(Ex/2), 0, 0)

    Qey = (cos(Ey/2), 0, sin(Ey/2), 0)

    Qez = (cos(Ez/2), 0, 0, sin(Ez/2))

    (qw, qx, qy, qz) = Qez * Qex * Qey * Qr

    return (qw, qx, qy, qz)

}
```

The conversion from Euler angles in x/y/z sequence (adopted in SLMB file format) to quaternion can be done by applying three Euler rotations in x/y/z sequence.

```
euler2quaternion_xyz(Ex, Ey, Ez, RX, RY, RZ)

{

    Qr = rotationaxisToquaternion(Rx, Ry, Rz)

    Qex = (cos(Ex/2), sin(Ex/2), 0, 0)
```

```
    Qey = (cos(Ey/2), 0, sin(Ey/2), 0)

    Qez = (cos(Ez/2), 0, 0, sin(Ez/2))

    (qw, qx, qy, qz) = Qez * Qey * Qex * Qr

    return (qw, qx, qy, qz)

}
```

The algorithm to convert quaternion to Euler angles is described in the paper *Quaternion to Euler angles conversion: A direct, general and computationally efficient method*.

The conversion from quaternion to Euler angles in y/x/z sequence can be done with the following pseudo-code:

```
quaternion2euler_yxz(qw, qx, qy, qz, RX, RY, RZ)

{

    Q = (qw, qx, qy, qz)

    Qr = rotationaxisToquaternion(Rx, Ry, Rz)

    (qRw, qRx, qRy, qRz) = Q * inverse(Qr)

    a = qRw - qRx

    b = qRy - qRz

    c = qRx + qRw

    d = -qRz - qRy

    θ+ = atan2(b, a)

    if (d==0 && c==0) {

        θ- = θ+

    } else {

        θ- = atan2(d, c)

    }

    θ1 = θ+ - θ-

    θ2 = acos(2 * (a^2 + b^2) / (a^2 + b^2 + c^2 + d^2) - 1)

    θ3 = θ+ + θ-

    Ex = wrap_to_degree (θ2 - π/2);

    Ey = wrap_to_degree (θ1);

    Ez = wrap_to_degree (-θ3);

    return (Ex, Ey, Ez)

}
```

The conversion from quaternion to Euler angles in x/y/z sequence can be done with the following pseudo-code.

```
quaternion2euler_xyz(qw, qx, qy, qz, RX, RY, RZ)

{

    Q = (qw, qx, qy, qz)

    Qr = rotationaxisToquaternion(Rx, Ry, Rz)

    (qRw, qRx, qRy, qRz) = Q * inverse(Qr)

    a = qRw - qRy

    b = qRx + qRz

    c = qRy + qRw

    d = qRz - qRx

    θ+ = atan2(b, a)

    if (d==0 && c==0) {

        θ- = θ+

    } else {

        θ- = atan2(d, c)

    }

    θ1 = θ+ - θ-

    θ2 = acos(2 * (a^2 + b^2) / (a^2 + b^2 + c^2 + d^2) - 1)

    θ3 = θ+ + θ-

    Ex = wrap_to_degree (θ1);

    Ey = wrap_to_degree (θ2 - π/2);

    Ez = wrap_to_degree (θ3);

    return (Ex, Ey, Ez)

}
```

D.2.4 BVH format conversion

D.2.4.1 Description

BVH is a file format that defines body motion. This format is widely known by professionals in animation area, used by movie/game studios and supported in most motion capture software. The specification of this format is described in sites such as:

- Motion Capture File Formats explained [3] (section 3.1)
- BVH Motion Capture Data Animated [4]
- Biovision BVH [5]

The BVH file has two sections. The first section is named `HIERARCHY` and describes the skeleton in its reference pose. The second section is named `MOTION` and describes the joint movements along the frames.

The syntax of a SLMB-compatible BVH file is defined in Table D.1:

Table D.1 - BVH file format

Syntax element	Note	Example
`BodyMotionBVH () {`		
`HIERARCHY`		`HIERARCHY`
`ROOT` `root_joint`	`root_joint` = `hips_JNT`	`ROOT hips_JNT`

`{`
`  joint_declaration(root_joint)`
`}`
	`joint_declaration(root_joint)` = declaration of the `root_joint`. The syntax of this declaration is defined in Table D.2	`{`
		`  OFFSET 0.000000 0.000000 (…)`
		`}`

`MOTION`		`MOTION`
`Frames:` `num_frames`	`num_frames` = number of frames in animation	`Frames: 5`
`Frame Time:` `frame_time`	`frame_time` = time between frames, in seconds (floating point format).	`Frame Time: 0.03333334`

`for (i = 0; i < num_frames; i++) {`
`  for (j = 0; j < num_joints; j++) {`
`  for (k=0; k < num_channels(j); k++) {`
`  <movement[i][j][k]>`
`  }`
`  }`
`}`
	`num_frames` = number of frames in animation
	
	`num_joints` = number of joints of the skeleton
	
	`num_channels(j)` = number of channels defined in `j`-th joint
	
	`movement[i][j][k]` = value of the `k`-th channel of the `j`-th joint movement in `i`-th frame.
		`0.280514 0.525651 -0.232078`
		`0.000000 -4.000000 0.000000`
		`0.000003 (...)`
		`0.280495 0.525651 -0.232078`
		`0.000000 -4.000000 0.000000`
		`-0.000081 (...)`
		`0.280514 0.525649 -0.232078`
		`0.000000 -4.000000 0.000000`
		`-0.000255 (...)`
		`0.280495 0.525651 -0.232078`
		`0.000000 -4.000000 0.000000`
		`-0.000512 (...)`
		`0.280514 0.525653 -0.232078`
		`0.000000 -4.000000 0.000000`
		`-0.001154 (...)`

`}`		

The syntax of `joint_declaration(joint)` is defined in Table D.2.

Table D.2 - Syntax of `joint_declaration(joint)`

Syntax element	Note	Example
`joint_declaration (joint) {`		
`OFFSET` `ref[joint].x ref[joint].y ref[joint].z`	`ref[joint]` = x/y/z coordinates of reference pose of `joint` from its parent	`OFFSET 0.000000 0.000000 0.000000`

`CHANNELS` `num_channels[joint]`
`for (i = 0; i < num_channels[joint]; i++) {`
`  channel_name[joint][i]`
`}`
	`num_channels[joint]` = number of channels of the `joint`
	
	`channel_name[joint][i]` = name of the `i`-th channel of the `joint`
		`CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation`

Syntax element	Note	Example

`if (num_children[joint] == 0) {`
`  End Site`
`  {`
`  OFFSET refend[joint].x refend[joint].y refend[joint].z`
`  }`
`} else {`
`  for (i = 0; i < num_children[joint]; i++) {`
`  child = child_joint[i][joint]`
`  JOINT child`
`  {`
`  joint_declaration(child)`
`  }`
`  }`
`}`
	`num_children[joint]` = number of children of `joint`
	
	`refend[joint]` = x/y/z coordinates of reference pose from `joint` to end of bone
	
	`child_joint[i][joint]` = i-th child of the `joint`.
	
	`joint_declaration(child)` = declaration of the `child`. The syntax of this declaration is defined in this table
		`JOINT spine_JNT`
		`{`
		`OFFSET 0.000000 4.130385 -0.008512`
		`CHANNELS 3 Zrotation Xrotation Yrotation`
		`End Site`
		`{`
		`OFFSET -2.8224037 0.000000 0.000000`
		`}`
		`}`

`}`		

The channels defined for joint movement may be:

- `Xposition`, `Yposition`, `Zposition`: joint position in x/y/z axis.
- `Xrotation`, `Yrotation`, `Zrotation`: rotation movement in Euler angles around x/y/z axis. The order of rotation channels defines the reverse order in which axis rotation should be done.

In `MOTION` section, the joints are ordered according to the order in which they are presented in `HIERARCHY` section. Also, the joint movement channels are ordered according to the order presented in the field `CHANNELS` of the joint.

In a SLMB-compatible BVH file, it is necessary to ensure that:

- Skeleton and reference pose defined in `HIERARCHY` section are compliant with ABNT NBR 25606 . The values of `num_children[joint]` and `child_joint[i][joint]` can be gotten from Table D.1, and the values of `ref[joint]` can be gotten from Table D.3.
- Channels in all joints follow the order `Zrotation Xrotation Yrotation`, which means that rotation will be done around Y, X and Z axis, in this order.
- Joint movements obey the restrictions defined in ABNT NBR 25606 , Table D.4.

D.2.4.2 Parameters

In the conversion pseudo-codes related to BVH files, it is assumed that the parameters starting with `bvh` are from the BVH file format as defined in Tables D.1 an D.2.

D.2.4.3 Conversion from BVH to SLMB body motion block

The conversion from a SLMB-compatible BVH file to SLMB body motion block is described by pseudo-code below. The functions `euler2quaternion_yxz` and `quaternion2euler_xyz` are defined in D.2.3.

```
/** Load joint information from HIERARCHY section of BVH file */

joint_list = <list of joints, using the same order in which they are declared in HIEARRCHY section>

for (j = 0; j < <size of joint_list>; j++) {

    joint_order[j] = <order of the joint of index j, as defined in ABNT NBR 25606 , Table D.9>

    joint_type[j] = <type of the joint of index j, as defined in ABNT NBR 25606 , Table D.9>
```

```
    channel_list[j] = <list of channels in the joint of index j, using the same order in which they are declared>

}


bmb.number_of_frames = bvh.num_frames

bmb.frame_time = bvh.frame_time


for (f = 0; f < bvh.num_frames; f++) {

    for (j = 0; j < <size of joint_list>; j++) {

        /** load movement information from MOTION section of BVH file */

        for (c = 0; c < <size of channel_list[j]>; c++) {

            if (channel_list[j][c] == "XPosition") XPosition = bvh.movement[f][j][c]

            if (channel_list[j][c] == "YPosition") YPosition = bvh.movement[f][j][c]

            if (channel_list[j][c] == "ZPosition") ZPosition = bvh.movement[f][j][c]

            if (channel_list[j][c] == "XRotation") XRotation = bvh.movement[f][j][c]

            if (channel_list[j][c] == "YRotation") YRotation = bvh.movement[f][j][c]

            if (channel_list[j][c] == "ZRotation") ZRotation = bvh.movement[f][j][c]

        }

        if (joint_type[j] == 0) {

            /** conversion code of translation movement in type-0 joint */

            bmb.Tx[f][joint_order[j]] = (Xposition + 0.5) * 65535

            bmb.Ty[f][joint_order[j]] = (Yposition + 0.5) * 65535

            bmb.Tz[f][joint_order[j]] = (Zposition + 0.5) * 65535

        }

        if (joint_type[j] == 0 || joint_type[j] == 1) {

            /** conversion code of rotation movement in type-0 and type-1 joint */

            (qw, qx, qy, qz) = euler2quaternion_yxz (Xrotation, Yrotation, Zrotation, (1,0,0), (0,1,0), (0,0,1))

            if (qw < 0) then qx = -qx, qy = -qy, qz = -qz

            bmb.Qx[f][joint_order[j]] = qx * 32767

            bmb.Qy[f][joint_order[j]] = qy * 32767

            bmb.Qz[f][joint_order[j]] = qz * 32767

        } else if (joint_type[j] == 2) {

            /** conversion code of rotation movement in type-2 joint */
```

```
            (qw, qx, qy, qz) = euler2quaternion_yxz (Xrotation, Yrotation, Zrotation, (1,0,0), (0,1,0), (0,0,1))

            (Ex, Ey, Ez) = quaternion2euler_xyz (qw, qx, qy, qz, geo.RX[joint_order[j]], geo.RY[joint_order[j]],
geo.RZ[joint_order[j]])

            E2x = (Ex + 90) / 180 * 1023

            E2y = (Ey + 90) / 180 * 1023

            E2z = (Ez + 180) / 360 * 4095

            bmb.E2[f][joint_order[j]] = (E2x << 22) + (E2y << 12) + E2z

        } else if (type == 3) {

            /** conversion code of rotation movement in type-3 joint */

            (qw, qx, qy, qz) = euler2quaternion_yxz (Xrotation, Yrotation, Zrotation, (1,0,0), (0,1,0), (0,0,1))

            (Ex, Ey, Ez) = quaternion2euler_xyz (qw, qx, qy, qz, geo.RX[joint_order[j]], geo.RY[joint_order[j]],
geo.RZ[joint_order[j]])

            bmb.E3[f][joint_order[j]] = (Ez + 180) / 360 * 255

        } else if (type == 4) {

            /** conversion code of rotation movement in type-4 joint */

            E4x = (Xrotation + 90) / 180 * 255

            E4y = (Yrotation + 90) / 180 * 255

            bmb.E4[f][joint_order[j]] = (E4x << 8) + E4y

        }

    }

}
```

D.2.4.4 Conversion from SLMB body motion block to BVH

The `HIERARCHY` section of the BVH file can be built using pre-defined joint information in ABNT NBR 25606 , Tables D.1 and D.3. This section may be also gotten from `avatarModel.bvh` file, stored at https://drive.google.com/file/d/1lO30145dpupPgHvbyNfc5mPlWWPuIeEy/view?usp=sharing .

The conversion from SLMB body motion block to the `MOTION` section of the BVH file is described by pseudo-code below. The functions `euler2quaternion_xyz` and `quaternion2euler_yxz` are defined in D.2.3.

```
/** Load joint information from HIERARCHY section of BVH file */

joint_list = <list of joints, using the same order in which they are declared in HIERARCHY section of BVH file>

for (j = 0; j < <size of joint_list>; j++) {

    joint_order[j] = <order of the joint of index j, as defined in ABNT NBR 25606 , Table D.9>

    joint_type[j] = <type of the joint of index j, as defined in ABNT NBR 25606 , Table D.9>
```

```
    channel_list[j] = <list of channels in the joint of index j, using the same order in which they are declared>

}


bvh.num_frames = bmb.number_of_frames

bvh.frame_time = bmb.frame_time


for (f = 0; f < bmb.number_of_frames; f++) {

    for (j = 0; j < <size of joint_list>; j++) {

    if (joint_type[j] == 0) {

        /** conversion code of translation movement in type-0 joint */

        Xposition = bmb.Tx[f][joint_order[j]] / 65535 - 0.5

        Yposition = bmb.Ty[f][joint_order[j]] / 65535 - 0.5

        Zposition = bmb.Tz[f][joint_order[j]] / 65535 - 0.5

    } else {

        Xposition = geo.refpose_from_parent[joint_order[j]].x

        Yposition = geo.refpose_from_parent[joint_order[j]].y

        Zposition = geo.refpose_from_parent[joint_order[j]].z

    }

    if (joint_type[j] == 0 || joint_type[j] == 1) {

        /** conversion code of rotation movement in type-0 and type-1 joint */

        qx = bmb.Qx[f][joint_order[j]] / 32767

        qy = bmb.Qy[f][joint_order[j]] / 32767

        qz = bmb.Qz[f][joint_order[j]] / 32767

        qw = sqrt(1 - qx^2 - qy^2 - qz^2)

        (Yrotation, Xrotation, Zrotation) = quaternion2euler_yxz (qw, qx, qy, qz, (1,0,0), (0,1,0), (0,0,1))

    } else if (joint_type[j] == 2) {

        /** conversion code of rotation movement in type-2 joint */

        E2x = bmb.E2[f][joint_order[j]] >> 22

        E2y = (bmb.E2[f][joint_order[j]] >> 12) & 0x03FF

        E2z = bmb.E2[f][joint_order[j]] & 0x0FFF

        Ex = E2x * 180 / 1023 - 90

        Ey = E2y * 180 / 1023 - 90
```

```
        Ez = E2z * 360 / 4095 - 180

        (qw, qx, qy, qz) = euler2quaternion_xyz (Ex, Ey, Ez, geo.RX[joint_order[j]], geo.RY[joint_order[j]],
geo.RZ[joint_order[j]])

        (Xrotation, Yrotation, Zrotation) = quaternion2euler_yxz (qw, qx, qy, qz, (1,0,0), (0,1,0), (0,0,1))

    } else if (joint_type[j] == 3) {

        /** conversion code of rotation movement in type-3 joint */

        Ez = bmb.E3[f][joint_order[j]] * 360 / 255 - 180

        (qw, qx, qy, qz) = euler2quaternion_xyz (0, 0, Ez, geo.RX[joint_order[j]], geo.RY[joint_order[j]],
geo.RZ[joint_order[j]])

        (Xrotation, Yrotation, Zrotation) = quaternion2euler_yxz (qw, qx, qy, qz, (1,0,0), (0,1,0), (0,0,1))

    } else if (joint_type[j] == 4) {

        /** conversion code of rotation movement in type-4 joint */

        E4x = bmb.E4[f][joint_order[j]] >> 8;

        E4y = bmb.E4[f][joint_order[j]] & 0xFF

        Xrotation = E4x * 180 / 255 - 90

        Yrotation = E4y * 180 / 255 - 90

        Zrotation = 0

    }

    /** fill movement information to MOTION section of BVH file */

    for (c=0; c< <size of channel_list[j]>; j++) {

        if (channel_list[j][c] == "XPosition") then bvh.movement[f][j][c] = XPosition

        if (channel_list[j][c] == "YPosition") then bvh.movement[f][j][c] = YPosition

        if (channel_list[j][c] == "ZPosition") then bvh.movement[f][j][c] = ZPosition

        if (channel_list[j][c] == "XRotation") then bvh.movement[f][j][c] = XRotation

        if (channel_list[j][c] == "YPosition") then bvh.movement[f][j][c] = YRotation

        if (channel_list[j][c] == "ZRotation") then bvh.movement[f][j][c] = ZRotation

    }

}
```

D.2.5 JSON format conversion

D.2.5.1 Description

The AR Emoji SDK software [6] can animate the face of an avatar that pre-defines the same meshes and blend shapes as defined in ABNT NBR 25606 , Tables D.6 and D.7, using a file in JSON format.

The syntax of this file is defined in table D.3.

Table D.3: Content of face movement JSON file

Syntax	Note	Example
`FaceMotionJSON () {`		
{		{
"name": name,	name = name of animation	"name": "MAL",
"version": version,	version = version of animation	"version": "1.2.3",

"blendShapes": [
for (i = 0; i < num_meshes; i++) {
{mesh_declaration(i)},
}
],
	num_meshes = number of meshes
	
	mesh_declaration(i) = declaration of i-th mesh, as defined in Table D.4.
		"blendShapes": [
		{"name": "head_GEO", (...)},
		{"name": "eyelash_GEO", (...)},
		{"name": "mouth_GEO", (...)},
		{"name": "eyebrow_l_GEO", (...)},
		{"name": "eyebrow_r_GEO", (...)}
		],

"shapesAmount": num_meshes,	num_meshes = number of meshes	"shapesAmount": 5,

"time": [
for (i = 0; i < num_frames; i++) {
time(i),
}
],
	num_frames = number of frames in animation
	
	time(i) = time stamp of i-th frame, in milliseconds.
		"time": [0, 30, 60, 90, 120],

"frames": num_frames	num_frames = number of frames in animation	"frames": 5
}		}
}		

The syntax of `mesh_declaration(mesh)` is defined in Table D.4:

Table D.4: Syntax of `mesh_declaration(mesh)`

Syntax	Note	Example
`mesh_declaration(mesh) {`		
"name": "mesh"	mesh = mesh name	"name": "eyebrow_l_GEO",
"fullName": "mesh"		"fullName": "eyebrow_l_GEO",
"blendShapeVersion": "3.1"		"blendShapeVersion": "3.1",
"morphTarget": "num_targets(mesh)",	num_targets(mesh) = number of blend shapes that can be applied to mesh, as defined in ABNT NBR 25606 , Table D.7.	"morphTarget": 9,

Syntax	Note	Example

"morphName": [
for (i = 0; i < num_targets(mesh); i ++) {
"target_name(mesh)(i)>",
}
],
	num_targets(mesh) = number of blend shapes that can be applied to mesh.
	
	target_name(mesh)(i) = name of i-th blend shape of the mesh
		"morphName": [
		"BrowsDown_Left",
		"BrowsDown_Right",
		"BrowsUp_Center",
		"BrowsUp_Left",
		"BrowsUp_Right",
		"Sneer",
		"HAPPY_51",
		"ANGRY_55",
		"SAD_58"
		],

"key": [
for (j = 0; j < num_frames; j++) {
[
for (i = 0; i< num_targets(mesh); i ++) {
weight(mesh)(i)(j),
}
],
}
]
	num_frames = number of frames in animation
	
	num_targets(mesh) = number of blend shapes of mesh
	
	weight(mesh)(i)(j) = weight of the i-th blend shape to mesh in j-th frame
		"key": [
		[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
		[0.0, 0.00258091744, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
		[0.0, 0.0104243588, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
		[0.0, 0.0234753173, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
		[0.0, 0.0415892377, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
		]

}		}
}		

D.2.5.2 Parameters

In the pseudo-codes that will be used for conversion for BVH files, it is assumed that the parameters starting with `json` are from the face movement JSON file format as defined in Tables D.3 an D.4.

D.2.5.3 Conversion from JSON to SLMB face motion block

The conversion code from JSON file to SLMB face motion block is described by the following pseudo-code:

```
fmb.number_of_frames = json.frames

for (f = 0; f < json.frames; f++) {

    fmb.frame_time[f] = json.time[f]

}

/** load blend shapes from JSON file. Face Motion Block will start empty and be filled while JSON file is being parsed */

fmb.number_of_blend_shapes = 0

for (m = 0; m < json.shapesAmount; m++) {

    for (b = 0; b < json.blendShapes[m].morphTarget; b++) {

        hasWeight = false

        for (f = 0; f < json.frames; f++) {

            weight = json.blendShapes[m].key[b][f]
```

```
            if (weight != 0) {

                if (not hasWeight)

                {

                    hasWeight = true

                    fmb.number_of_blend_shapes++;

                    target_index = fmb.number_of_blend_shapes - 1

                    fmb.blend_shape_id[target_index] = <get blend shape id from mesh defined in blendShapes[m].name and blend shape defined in blendShapes[m].morphName[b], according to ABNT 25606 , Table D.11 >

                    /** build frame list with frames whose weight is different than 0 */

                    number_of_frames = 0

                }

                number_of_frames++

                frame_index = number_of_frames - 1

                frame_list[frame_index] = f

                /** conversion formula for blend shape weight */

                fmb.blend_shape_weight [target_index][frame_index] = weight * 65535

            }

        }



        if (not hasWeight)

        {

            /** convert the list of frames into a list of frame ranges */

            current_frame_defined = false

            fmb.number_of_frame_ranges[target_index] = 0

            for (k = 0; k < number_of_frames; k++) {

                if (not current_frame_defined) {

                    current_frame_defined = true

                    first_frame = frame_list[k];

                } else if (frame_list[k] != current_frame + 1) {

                    last_frame = current_frame

                    fmb.number_of_frame_ranges[target_index]++

                    range_index = fmb.number_of_frame_ranges[target_index] - 1

                    fmb.frame_range_first[target_index][range_index] = first_frame
```

```
                    fmb.frame_range_size[target_index][range_index] = last_frame - first_frame + 1
                    first_frame = frame_list[i];
                }
                current_frame = frame_list[k]
            }
            if (current_frame_defined) {
                last_frame = current_frame
                fmb.number_of_frame_ranges[target_index]++
                range_index = fmb.number_of_frame_ranges[target_index] - 1
                fmb.frame_range_first[target_index][range_index] = first_frame
                fmb.frame_range_size[target_index][range_index] = last_frame - first_frame + 1
            }
        }
    }
}
```

D.2.5.4 Conversion from SLMB Face Motion Block to JSON

The conversion code from SLMB face motion block to JSON file is described by the following pseudo-code:

```
json.frames  = fmb.number_of_frames
for (f = 0; f < fmb.number_of_frames; f++) {
    json.time[f] = fmb.frame_time[f]
}

json.shapesAmount = 0
for (b = 0; b < fmb.number_of_blend_shapes; b++) {
    mesh_name, target_name = <get mesh and blend shape names from fmb.blend_shape_id[b], according to ABNT 25606 , Table D.11>
    /** check if mesh of name mesh_name is already registered and create an entry if it is not */
    mesh_index_defined = false
    for (m = 0; m < json.shapesAmount; m++) {
        if (json.blendShapes[m].name = mesh_name) {
            mesh_index = m
```

```
            mesh_index_defined = true

        }

    }

    if (not mesh_index_defined){

        json.shapesAmount++

        mesh_index = json.shapesAmount - 1

        json.blendShapes[mesh_index].name = mesh_name

        json.blendShapes[mesh_index].morphTarget = 0

    }

    /** add target_name in the list of blend shapes */

    json.blendShapes[mesh_index].morphTarget++

    target_index = json.blendShapes[mesh_index].morphTarget - 1

    json.blendShapes[mesh_index].morphName[target_index] = target_name


    /** initialize list of weights with 0 for all frames */

    for (f = 0; f < fmb.number_of_frames; f++) {

        json.blendShapes[mesh_index].key[target_index][f] = 0

    }


    /** fill list of weights according to data collected from Face Motion Block */

    frame_list_size = 0

    for (r = 0; r < fmb.number_of_frame_ranges[b]; r++) {

        for (f = 0; f < fmb.frame_ranges_size[b][r]; f++) {

            frame = fmb.frame_ranges_first[b][r] + f

            /** conversion formula for blend shape weight */

            json.blendShapes[mesh_index].key[target_index][frame] = fmb.number_of_blend_shapes[i][j] / 65535

        }

    }

}
```

D.2.6 glTF format conversion

D.2.6.1 Description

The `avatarModel.zip` file from https://drive.google.com/file/d/1yZjdmS832-SDF8N7SQWQ2sR4UWELVvVA/view?usp=sharing provides an avatar sample with the skeleton, reference poses, face meshes, and blend shapes as defined in ABNT NBR 25606 , D.3 and D.4. Body and face motion information may be added to these files to produce movement in that avatar. The animation data should be defined according to the glTF™ 2.0 Specification [7].

Though this avatar is presented in full body, the body movement produced by the SLMB file do not apply to lower member joints (legs and feet).

The picture D.1 exemplifies the glTF animation data structure:

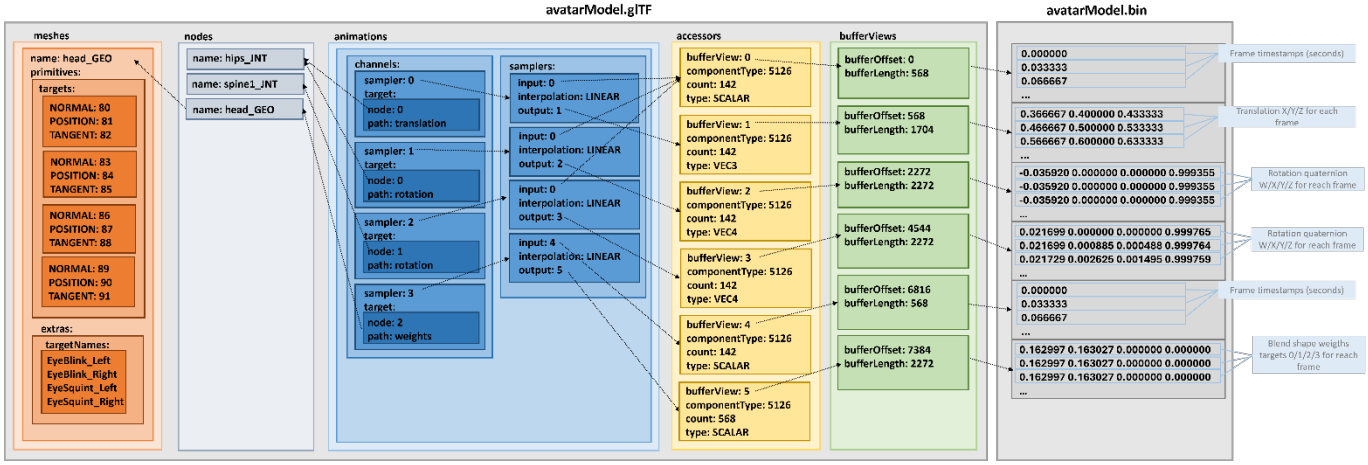

Figure D.1: Animation data in glTF file format

The binary file is composed of several sequential buffers that are mapped in the `bufferViews` array in glTF file. Each buffer contains `bufferOffset` and `bufferLength` attributes, referring to position and size of buffer in binary file.

The `accessors` array maps the accessors that compose the animation content. Each accessor refers to a `bufferView` attribute by its index in `bufferViews` array when large amount of data is required (index `0` refers to the first entry).

The glTF file may have an `animations` array, which is a list of animations to be applied to the model. Each item in `animations` array provides a `samplers` array and a `channels` array.

The `samplers` array is a list of samplers, each one containing three attributes:

- `input` - accessor that provides the list of frame times.
- `interpolation` - method to transition from one frame to other.
- `output` - accessor that provides the values of animated property for each frame.

The `channels` array is a list of channels. Each channel connects a `sampler` to a `target` to be animated. The `target` structure contains the `node` and the `path` attributes.

The nodes of the avatar model - which may be a skeleton joint or a mesh - are listed in the `nodes` array. So, the `node` attribute refers to the index of the node in `nodes` array.

The `path` attribute refers to the property to be animated, which can be:

- `translation` - for joint nodes.
- `rotation` - for joint nodes.
- `weights` - for mesh node.

The `translation` property value consists of 3(three) floating point numbers that define the X/Y/Z translation vector.

The `rotation` property value consists of 4(four) floating point numbers that define the elements of the quaternion (W/X/Y/Z) that describes the rotation movement.

The `weights` property value consists of a list of floating-point numbers that define the weights applied by each blend shape in the mesh. The order of the blend shapes in that list is the same as the `targets` array in the `meshes` array item.

D.2.6.2 Parameters

In the pseudo-codes that will be used for conversion for BVH files, it is assumed that the parameters starting with `glTF` are from the face movement glTF file format as defined in D.2.6.1.

D.2.6.3 Conversion from SLMB to glTF

The conversion code from SLMB body and face motion blocks to glTF animation file is described by the following pseudo-code:

```
number_of_channels = 0

number_of_samplers = 0



for (j = 0; j < bmb.number_of_joints; j++) {

    joint_type, joint_name = <joint type and name corresponding to order j, according to ABNT 25606, Table D.9>

    if (joint_type == 0) {

        /** register translation movement for type-0 joint */

        number_of_samplers++

        sampler_index = number_of_samplers - 1

        for (f = 0; f < bmb.number_of_frames; f++)

            glTF.animations.samplers[sampler_index].input.time[f] = f * bmb.frame_time

            /** conversion formula for translation of type-0 joints */

            tx = bmb.Tx[j][f] / 65535 - 0.5

            ty = bmb.Ty[j][f] / 65535 - 0.5

            tz = bmb.Tz[j][f] / 65535 - 0.5

            glTF.animations.samplers[sampler_index].output.translation[f][0] = tx
```

```
            glTF.animations.samplers[sampler_index].output.translation[f][1] = ty
            glTF.animations.samplers[sampler_index].output.translation[f][2] = tz
        }
        number_of_channels++
        channel_index = number_of_channels - 1
        glTF.animations.channels[channel_index].sampler = sampler_index
        glTF.animations.channels[channel_index].node = <index in glTF.nodes corresponding to joint_name>
        glTF.animations.channels[channel_index].path = "translation"
    }

    /** register rotation movement */
    number_of_samplers++
    sampler_index = number_of_samplers - 1
    for (f = 0; f < bmb.number_of_frames; f++)
        glTF.animations.samplers[sampler_index].input.time[f] = f * bmb.frame_time
        if (joint_type == 0 || joint_type == 1) {
            /** conversion formula for rotation of type-0 and type-1 joints */
            qz = bmb.Qx[j][f]/32767
            qy = bmb.Qy[j][f]/32767
            qz = bmb.Qz[j][f]/32767
            qw = sqrt(1 - qx^2 - qy^2 - qz^2)
        } else if (joint_type == 2) {
            /** conversion formula for rotation of type-2 joints */
            E2x = bmb.E2[j][f] << 22
            E2y = (bmb.E2[j][f] << 12) & 0x3FF
            E2z = bmb.E2[j][f] & 0xFFF
            Ex = E2x * 180 / 1023 - 90
            Ey = E2y * 180 / 1023 - 90
            Ez = E2z * 360 / 4095 - 180
            (qx, qy, qz, qw) = euler2quaternion_xyz (Ex, Ey, Ez, geo.RX, geo.RY, geo.RZ)
        } else if (joint_type == 3) {
            /** conversion formula for rotation of type-3 joints */
```

```
            Ez = bmb.E3[j][f] * 360 / 255 - 180
            (qx, qy, qz, qw) = euler2quaternion_xyz (0, 0, Ez, geo.RX, geo.RY, geo.RZ)
        } else if (joint_type == 4) {
            /** conversion formula for rotation of type-4 joints */
            E4x = bmb.E4[j][f] << 8
            E4y = bmb.E4[j][f] && 0xFF
            Ex = E4x * 180 / 255 - 90
            Ey = E4y * 180 / 255 - 90
            (qx, qy, qz, qw) = euler2quaternion_xyz (Ex, Ey, 0, (1,0,0), (0,1,0), (0,0,1))
        }
        glTF.animations.samplers[sampler_index].output.rotation[f][0] = qw
        glTF.animations.samplers[sampler_index].output.rotation[f][1] = qx
        glTF.animations.samplers[sampler_index].output.rotation[f][2] = qy
        glTF.animations.samplers[sampler_index].output.rotation[f][3] = qz
    }
    number_of_channels++
    channel_index = number_of_channels - 1
    glTF.animations.channels[channel_index].sampler = sampler_index
    glTF.animations.channels[channel_index].node = <index in glTF.nodes corresponding to joint_name>
    glTF.animations.channels[channel_index].path = "rotation"
}

/** construct list of meshes, blend shapes and frames defined in SLMB motion block */
number_of_meshes = 0
for (b = 0; b < fmb.number_of_blend_shapes; b++) {
    mesh_name, target_name = <get mesh and blend shape names from fmb.blend_shape_id[b], according to ABNT 25606 , Table D.11>

    /** check if mesh was already registered in the list, create a new entry otherwise */
    mesh_index_defined = false
    for (j=0; j<number_of_meshes; j++) {
        if (mesh_list[j].name = mesh_name) {
            mesh_index = j
```

```
            mesh_index_defined = true

        }

    }

    if (not mesh_index_defined) {

        number_of_meshes++

        mesh_index = munber_of_meshes - 1

        mesh_list[mesh_index].name = mesh_name

        mesh_list[mesh_index].index = <index of glTF.meshes array whose name attribute corresponds to mesh_name>

        mesh_list[mesh_index].number_of_targets = 0

    }


    /** add a new entry for blend shape */

    mesh_list[mesh_index].number_of_targets++

    target_index = mesh_list[mesh_index].number_of_targets - 1

    mesh_list[mesh_index].targets[target_index].name = target_name

    mesh_list[mesh_index].targets[target_index].number_of_frames = 0


    /** fill the list of frames whose weight is different than 0 */

    for (r = 0; r < fmb.number_of_frame_ranges[b]; r++) {

        for (f = 0; f < fmb.frame_range_size[b][r]; f++) {

            mesh_list[mesh_index].targets[target_index].number_of_frames++

            frame_index = mesh_list[mesh_index].targets[target_index].number_of_frames - 1

            mesh_list[mesh_index].targets[target_index].frames[frame_index].frame = fmb.frame_range_first[b][r] + f

            mesh_list[mesh_index].targets[target_index].frames[frame_index].weight =
fmb.blend_shape_weight[b][frame_index]

        }

    }

}


/** register face movements */

for (m = 0; m < number_of_meshes; m++) {

    mesh_index = meshes_list[m].index

    mesh_name = meshes_list[m].name
```

```
    number_of_samplers++

    sampler_index = number_of_samplers - 1

    /** initialize weights to 0 */

    for (f = 0; f < fmb.number_of_frames; f++) {

        glTF.animations.samplers[sampler_index].input.time[f] = f * fmb.frame_time

        for (b = 0; b < <size of glTF.meshes[mesh_index].extras.targetNames>; b++) {

            glTF.animations.samplers[sampler_index].output.weights[f][b] = 0

        }

    }

    /** fill weights with information collected in mesh list */

    for (b = 0; b < meshes_list[m].number_of_targets; b++) {

        target_name = meshes_list[m].targets[b].name

        target_index = <index of glTF.meshes[mesh_index].extras.targetNames array that corresponds to target_name>

        for (f = 0; f < meshes_list[m].targets[b].number_of_frames; f++) {

            frame = meshes_list[m].targets[b].frames[f].frame

            /** conversion formula for blendshape weights */

            weight = meshes_list[m].targets[b].frames[f].weight / 65535

            glTF.animations.samplers[sampler_index].output.weights[target_index][frame] = weight

        }

    }

    number_of_channels++

    channel_index = number_of_channels - 1

    glTF.animations.channels[channel_index].sampler = sampler_index

    glTF.animations.channels[channel_index].node = <index in glTF.nodes corresponding to mesh_name >

    glTF.animations.channels[channel_index].path = "weights"

}
```

D.3 Concept

The Figure D.2 exemplifies the process of transmitting sign language motion data through TV 3.0 system.

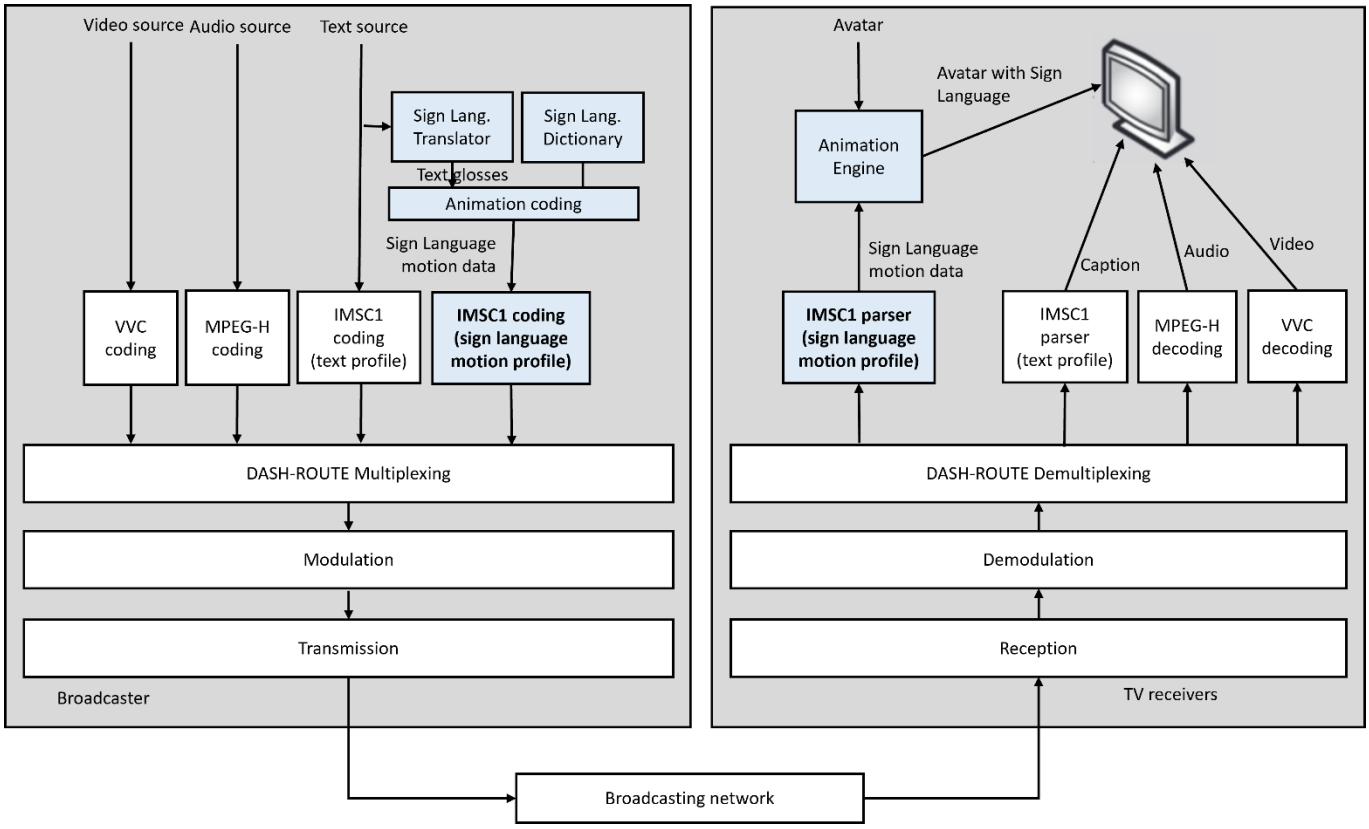

Figure D.2 – Process of broadcasting sign language motion data

This operational guideline is focused on a solution for sign language motion encoding based on a sign language dictionary. There are other possible solutions – for example, the face and body movements can be captured from a human sign language interpreter in real time. It also focuses on avatar engines that can read some of the known animation file formats, as described in D.2.

From a TV broadcasting station, a sign language translator defines the sequence of glosses that expresses the audio/video context in sign language. Through a sign language dictionary, the body and face movements related to that sequence of signs are coded and a file is generated. This file is declared in an IMSC1 document and multiplexed with audio, video, and captions. The multiplexed stream is modulated and transmitted by the broadcaster.

From TV receiver, the content received from broadcaster is demultiplexed, and the SLMB files are extracted from the TTML segments. These files feed an avatar animation engine, which animates a pre-defined avatar using the file contents. The result is the avatar moving its body and face to interpret the context in sign language.

D.4 Transmission

D.4.1 Sign language dictionary

A sign language dictionary consists of a list of pairs composed of a gloss and the information needed to produce the sign that corresponds to that gloss.

In the model proposed by this guideline for a sign language dictionary, each gloss is mapped to two files. One file contains information about body movement, and the other one contains information about face movement.

These movements allow a 3D avatar with the geometry defined in ABNT NBR 25606 , D.3 and D.4, to produce the sign related to the gloss. The body motion file will be stored in BVH format, as described in D.2.4.1. The face motion file will be stored in JSON format, as described in D.2.5.1. These file formats can be interpreted by the AR Emoji SDK software [6].

D.4.2 Sign language sentence construction

The sentence construction process starts with a sequence of glosses that complete a sentence in sign language. These glosses SHOULD be part of the sign language dictionary.

Example:

- `"EU" + "CASA" + "VOLTAR"` is the sequence of glosses in Brazilian Sign Language (Libras) that expresses the sentence “I come back home”.

The BVH and JSON files corresponding to each gloss are extracted from the sign language dictionary.

Example:

- Retrieve `EU.bvh`, `EU.json`, `CASA.bvh`, `CASA.json`, `VOLTAR.bvh`, `VOLTAR.json` from Libras dictionary.

The BVH and JSON files are concatenated to construct the movement needed to express the sentence in sign language.

Example:
- `EU.bvh + CASA.bvh + VOLTAR.bvh` => **`EU_CASA_VOLTAR.bvh`**
- `EU.json + CASA.json + VOLTAR.json` => **`EU_CASA_VOLTAR.json`**

In concatenation process, it is recommended to treat the transition between signs, so animation during this transition is as fast and smooth as possible.

D.4.3 Sign language motion bundle encoding

BVH and JSON files can be converted to a SLMB file using the process described in Figure D.3.

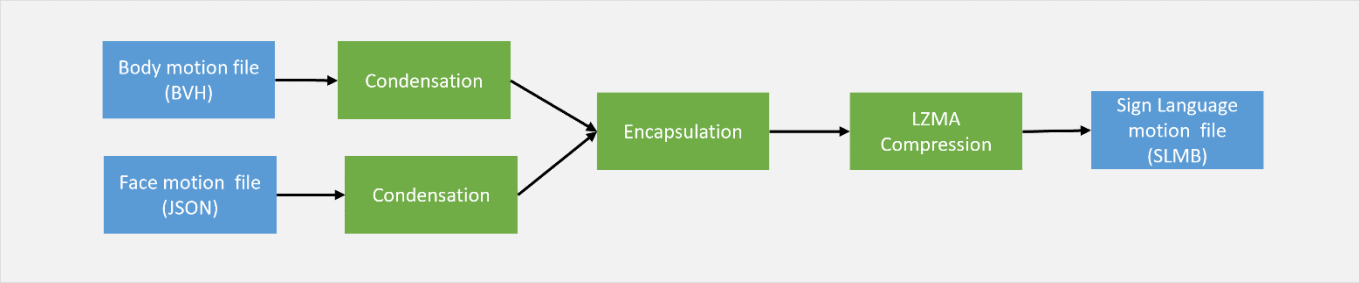

Figure D.3: Sign Language motion data encoding process.

Initially, BVH file is condensed and presented in the `BodyMotionBlock()` format, according to ABNT 25606 , Table D.8. Likewise, JSON file is condensed and presented in the `FaceMotionBlock()` format, according to ABNT NBR 25606 , table D.10.

Both body and face motion data are concatenated and encapsulated in `MotionBundle()` format, according to ABNT NBR 25606 , Table D.13. This bundle will be compressed with LZMA compression algorithm, resulting in the SLMB file.

D.4.3.1 Body and face motion data condensation

The information need to fill `BodyMotionBlock()` can be gotten from the BVH movement components, using the algorithm described in D.2.4.3.

The information needed to fill `FaceMotionBlock()` can be gotten from the JSON file, using the algorithm described in D.2.5.3.

D.4.3.2 Encapsulation

Body and face motion blocks should be encapsulated into one block according to ABNT NBR 25606 , Table D.13. The blocks may be ordered according to Figure D.3.

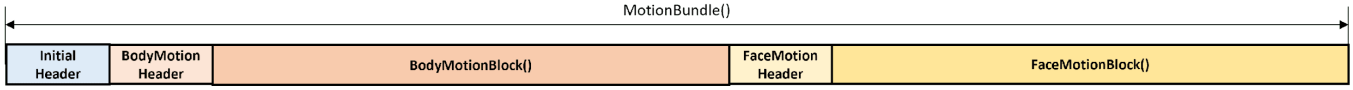

Figure D.3: Encapsulated motion data

Considering that `BodyMotionBlock()` contains the body motion data for geometry ID 1 and that `FaceMotionBlock()` contains the face motion data for geometry ID 1, the `MotionBundle()` data can be structured as in Table D.5.

Table D.5 – Example of `MotionBundle()` data

Number of bytes	Element	Parameter	Value	Comment
1	Title	`header`	0x60	= (4-1) << 5 + 0 (key size=4, payload size=0)
4		`key`	0x53 0x4c 0x4d 0x42	= “SLMB”
1	Body motion	`header`	0x3F	= (2-1) << 5 + 0x1F (key size=2, payload size>31)
2		`key`	0x01 0x01	= body motion, geometry ID 1
4		`payload_size`	`bSize`	= size of `BodyMotionBlock()` in bytes
`bSize`		`payload`	`BodyMotionBlock()`	
1	Face Motion	`header`	0x3F	= (2-1) << 5 + 0x1F (key size=2, payload size>31)
2		`key`	0x02 0x01	= face motion, geometry ID 1
4		`payload_size`	`fSize`	= size of `FaceMotionBlock()` in bytes
`fSize`		`payload`	`FaceMotionBlock()`	

D.4.3.3 Compression

Compression of `MotionBundle()` block to SLMB file may be done by the XZ Utils tool [8].

`MotionBundle()` block is saved in a file with extension `.slmb`.

- Example: save block in `EU_CASA_VOLTAR.slmb` file.

File is compressed with `xz` command.

- Example:

  `# xz EU_CASA_VOLTAR.slmb`

Result is the SLMB file with `.slmb.xz` extension.

- Example: result is `EU_CASA_VOLTAR.slmb.xz` file.

D.4.4 Sign language motion file transport

D.4.2.1 IMSC1 coding

The sign language motion file transport process starts with the construction of an intermediate synchronic document, as defined in ABNT NBR 25606 , D.6. The TTML document can be coded in IMSC1 sign language motion profile to reference the SLMB files with timestamps for synchronization with audio/video content.

Figure D.4 exemplifies the intermediate synchronic document exemplified in sign language motion profile.

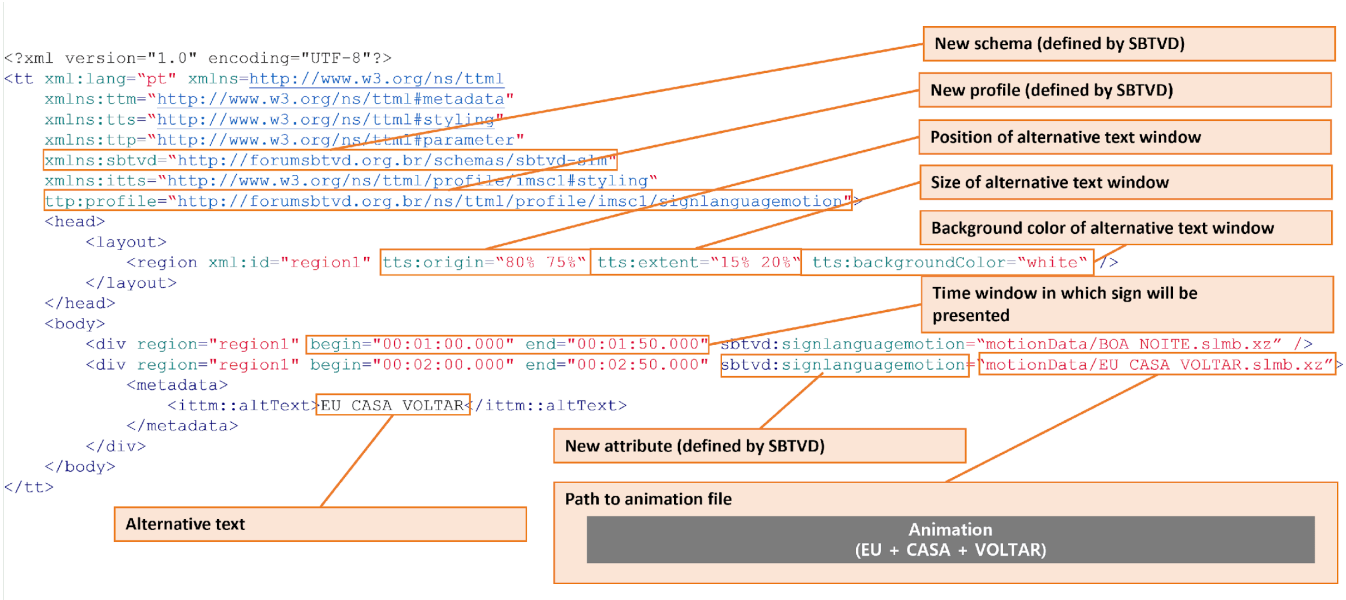

Figure D.4: Example of a document in sign language motion profile

The attributes related to region (`tts:origin`, `tts:extent`, `tts:backgroundColor`) refer to the window in which alternate text will be presented.

The attributes related to timing (`begin`, `end`) refer to the beginning and end times of the animation. It is recommended that the difference between `end` and `begin` attribute values matches with the duration of the sign language motion.

D.4.2.2 MP4 segmentation and encapsulation

The MP4 segmentation and encapsulation of a IMSC1 document in sign language motion profile must follow the specification in ABNT NBR 25606 , D.7 and D.8.

In segmentation process, the intermediate synchronic document is the base to produce the sign language motion segments. Each segment is related to a defined period. The `div` elements with `sbtvd:signlanguagemotion` attribute are filtered, so that only the elements that are presented within that period are included.

For example, based on the intermediate synchronic document shown in Figure D.4, if 50-second segments are used:

- Segment 1 (0s ~ 50s) – no sign language motion data
- Segment 2 (50s ~ 100s) – sign language motion data for “BOA NOITE”
- Segment 3 (100s ~ 150s) – sign language motion data for “BOA NOITE” and “EU CASA VOLTAR”
- Segment 4 (150s ~ 200s) – sign language motion data for “EU CASA VOLTAR”
- Segment 5 (200s ~ 250s) – no sign language motion data

The `sbtvd:signlanguagemotion` attribute refers to the path to the corresponding SLMB file. So, it is necessary to include this file in the segment, along with TTML file.

To fit more than one file in a single segment, the subsample resource in MP4 encapsulation should be used.

- Files are loaded according to the paths defined in `sbtvd:signlanguagemotion` attribute of each `div` element.
- TTML and binary files are concatenated in segment after MP4 header.
- TTML file is numbered with subsample=0. The subsequent binary files are numbered with subsample=1, 2, etc.
- The `subs` folder in MP4 header contains information about the number of subsamples, where each subsample begins and where it ends.
- The value of `sbtvd:signlanguagemotion` attribute is changed to `urn:mpeg:14496-30:subs:`<subsample number>

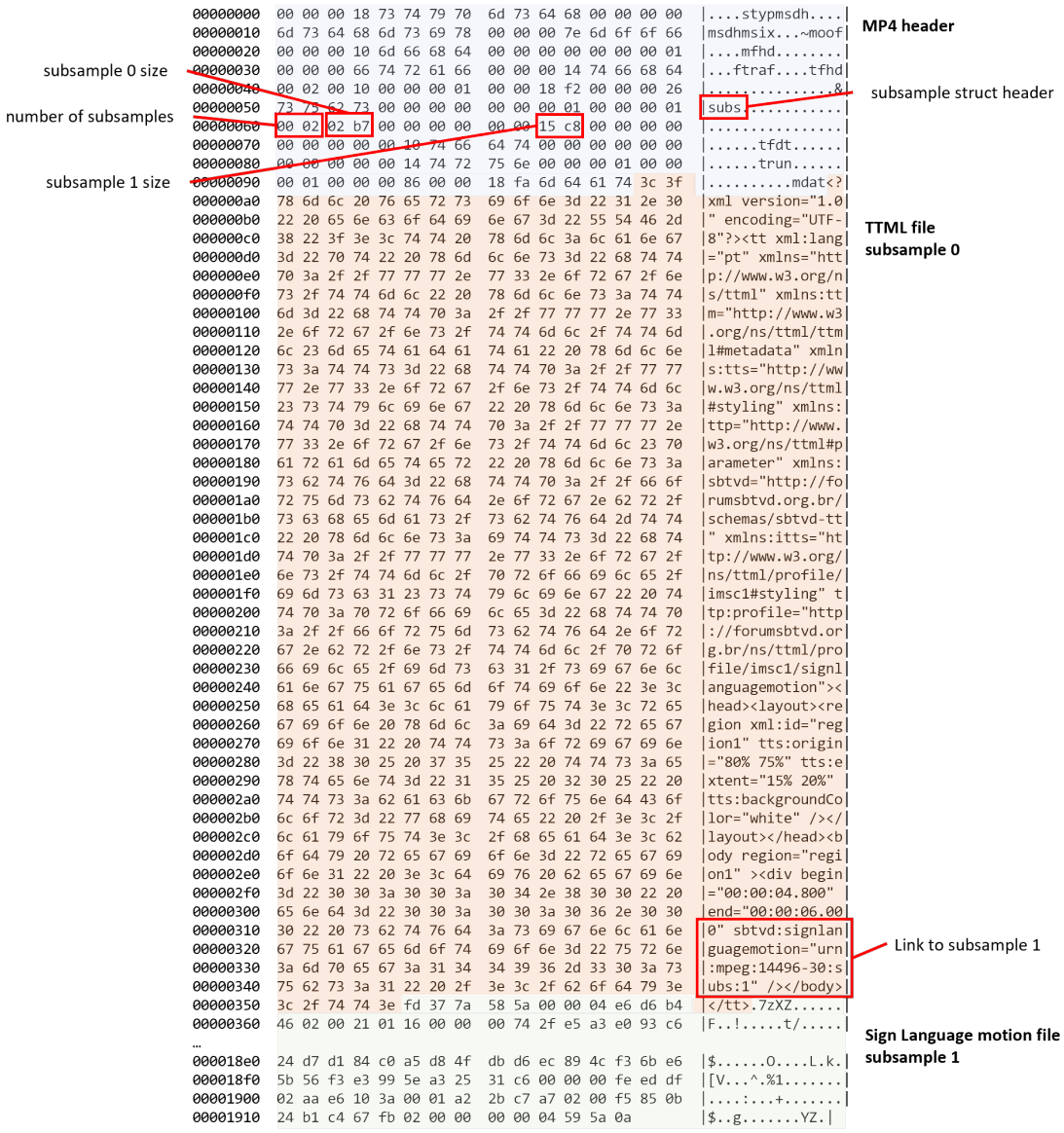

Figure D.5: Example of an IMSC1 segment with sign language motion profile

D.4.2.3 Media presentation description

Declaration of the TTML segments with IMSC1 sign language motion profile in MPD file should follow the same pattern used in other IMSC1 profiles, with the following adjustments:

- `SupplementalProperty` entry with `schemeIdUri` attribute set to `http://dashif.org/guidelines/dash-atsc-closedcaption` and `value` attribute with `profile` set to 2
- `codecs` attribute set to `stpp.ttml.im1m`

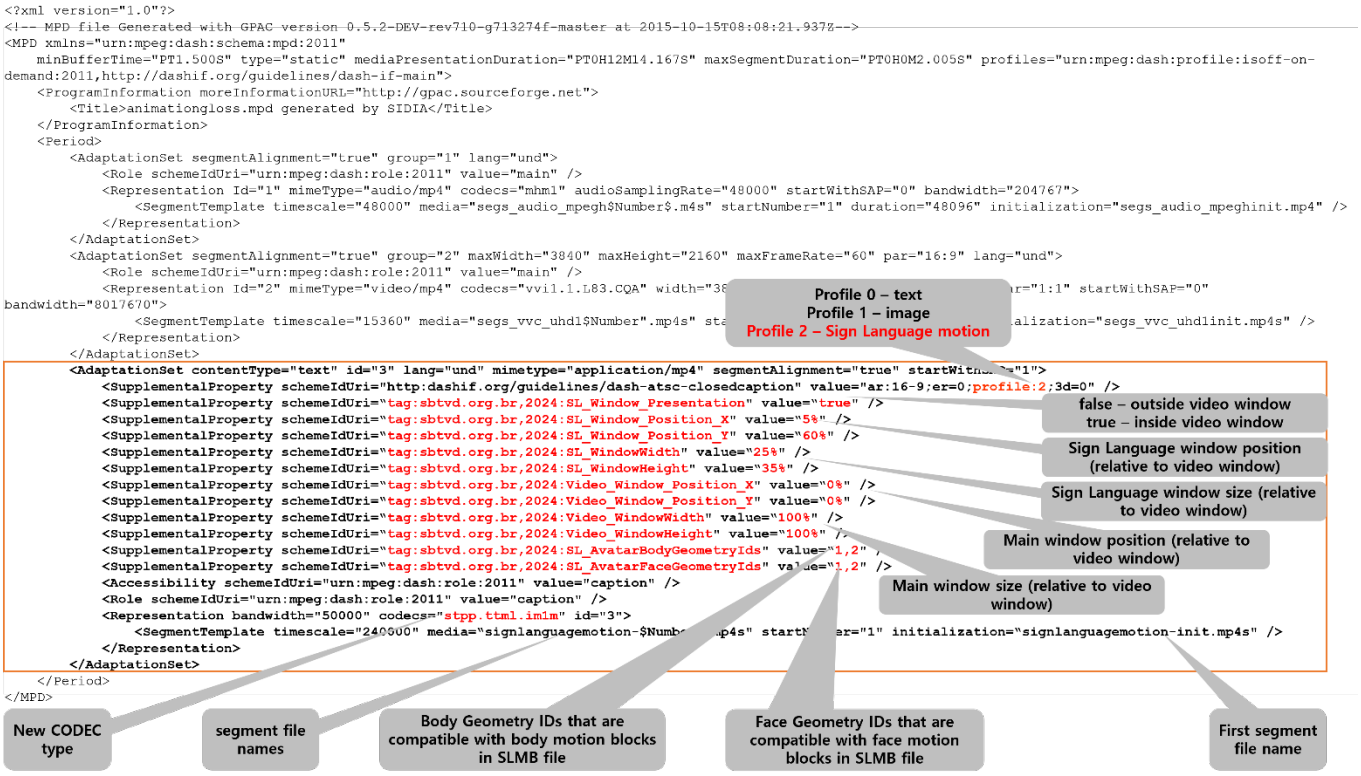

Figure D.6: Example of a IMSC1 Sign Language Motion profile stream declaration in a .MPD file

In case sign language window should be presented inside video window:

- `SL_Window_Presentation` attribute should be set to `true`
- `SL_Window_Position_X`, `SL_Window_Position_Y`, `SL_Window_Width` and `SL_Window_Position_Height` attributes should define the position and the size of the sign language window.
- `Video_Window_Position_X`, `Video_Window_Position_Y`, `Video_Window_Width` and `Video_Window_Position_Height` attributes should define the position and the size of the main language window.

The body geometry IDs defined in the body motion blocks that compose the SLMB files (refer to ABNT NBR 25606 , Table D.13) may be included in `SL_AvatarBodyGeometryIds`. Likewise, the face geometry IDs defined in the face motion blocks that compose the SLMB files may be included in `SL_AvatarFaceGeometryIds`.

D.5 Reception

D.5.1 Sign language motion file extraction

D.5.1.1 MPEG-DASH player

A MPEG-DASH player which can interpret the `AdaptationSet` entries for TTML segments in MPD file can also interpret `AdaptationSet` entries for segments in IMSC1 Sign Language Motion profile, as far as it can interpret correctly the new settings related to this profile.

- In `SupplementalProperty` entry in which `schemaIdUri` is set to `http://dashif.org/guidelines/dash-atsc-closedcaption`
  - `value` with `profile:2`
- `codecs` attribute set to `stpp.ttml.im1m`

In addition, the following information may be collected from `SupplementalProperty` entries:

- Sign language window presentation:
  - If value of `SL_WindowPresentation` is true:
    - Sign language window should be presented according to the values of the `SL_Window_Position_X`, `SL_Window_Position_Y`, `SL_Window_Width` and `SL_Window_Position_Height` attributes.
    - Main video window should be presented according to the values of the `Video_Window_Position_X`, `Video_Window_Position_Y`, `Video_Window_Width` and `Video_Window_Position_Height` attributes.
  - If value of `SL_WindowPresentation` is false or not defined:
    - Main video window and Sign Language window should be presented according to ABNT 25606 , 8.1.
- Body and face geometry IDs that are compatible with SLMB file, according to the values of `SL_AvatarBodyGeometryIds` and `SL_AvatarFaceGeometryIds`, respectively. This information may be used to select the avatars that can use the motion data provided by SLMB file.

D.5.1.2 MP4 parser

A MP4 parser that is compliant with ISO/IEC 14496-30 [12], Section 5, should be able to extract all its sub-samples that are embedded in ISOBMFF-encapsulated segment. For IMSC1 sign language motion profile, data extracted from the sub-sample 0 should be saved to a TTML file (text), and data extracted from the subsequent sub-samples should be saved to SLMB files (binary).

D.5.1.3 TTML parser

A TTML parser can parse a segment in IMSC1 sign language motion profile, as far as it interprets correctly the new attributes from this profile:

- `ttp:profile` attribute (value `http://forumsbtvd.org.br/ns/ttml/profile/imsc1/signlanguagemotion` identifies the sign language motion profile)
- `xmlns:sbtvd` attribute (value `http://forumsbtvd.org.br/schemas/sbtvd-slm` to include schema for sbtvd namespace)
- `sbtvd:signlanguagemotion` attribute: (`urn:mpeg:14496-30:subs:<subsample number>`).
  - `subsample number` refers to the index of the subsample where SLMB file is located.
  - For example, `subsample number` = *1* refers to the first SLMB file, `subsample number` = *2* refers to second SLMB file, and so on.

TTML parser may extract timed animation data from TTML segment, which may be used to configure the avatar presentation engine:

- SLMB file - animation data
- Start timestamp - define when to start animation
- End timestamp (optional) - define when to stop animation
- Alternate text (optional) – define a text to be presented in screen in case sign language avatar window

cannot be presented.
- Presentation window information (optional) – define the window in which alternate text should be presented.

D.5.2 Sign language motion bundle decoding

SLMB files can be decoded using the process described in Figure D.7:

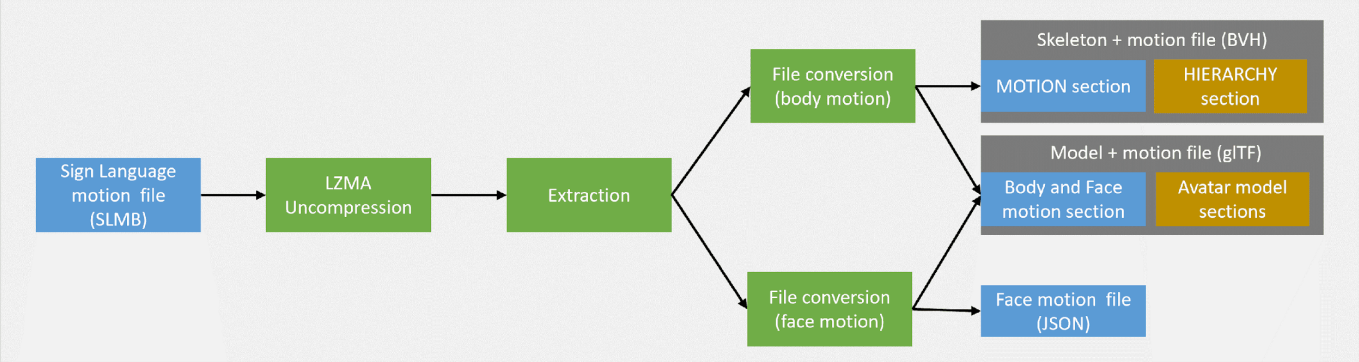

Figure D.7: Sign language motion data decoding process.

Initially, SLMB file is uncompressed using LZMA compression algorithm, resulting in a block in `MotionBundle()` format. Then, body and face animation data from blocks in `BodyMotionBlock()` and `FaceMotionBlock()` formats are extracted from uncompressed block.

The body animation data extracted from `BodyMotionBlock()` may be converted to data in a known animation file format. This operational guideline describes conversion to BVH and glTF file formats.

Likewise, the face animation data extracted from `FaceMotionBlock()` may be converted to data in a known animation file format. This operational guideline describes conversion to JSON and glTF file formats.

The resulting files can feed the animation engine to animate the avatar and produce the sentence in sign language.

D.5.2.1 Uncompression

Uncompression of SLMB file to `MotionBundle()` block may be done by the XZ Utils tool [8].

- Check that SLMB file has extension `.slmb.xz.` (ex. `EU_CASA_VOLTAR.slmb.xz`)
- Uncompress file using `unxz` command. (ex. **`unxz EU_CASA_VOLTAR.slmb.xz`**).
- Resulting file with `slmb` extension. (ex. `EU_CASA_VOLTAR.slmb`) contains the `MotionBundle()` block).

D.5.2.2 Motion block extraction

Extraction of face and body motion data from `MotionBundle()` block from ABNT NBR 25606 , Table D.13, may be done by parsing the header of each block. The `key` field in header indicates the block type – described in ABNT NBR 25606, Table D.12, whereas `payload_size` field indicates begin and end of the block. The next block header starts right after the end of the previous block.

```
pos = 0

element_size = 0

while (pos < <size of MotionBundle >) {

    element_size = element_size + 1

    index = element_size - 1

    key_size = (MotionBundle[pos] >> 5) + 1

    payload_size = MotionBundle[pos] & 0x1F

    pos = pos + 1

    element[index].key = MotionBundle[pos to pos + key_size - 1]

    pos = pos + element[index].key_size

    if (payload_size == 0x1F) {

        payload_size = MotionBundle[pos to pos + 3]

        pos = pos + 4

    }

    element[index].payload = MotionBundle[pos to pos + payload_size - 1]

    pos = pos + payload_size

}



for (index = 0; index < element_size; index++) {

    if (element[index].key == {0x01, 0x01}) {

        BodyMotionBlock = element[index].payload

    }

    if (element[index].key == {0x02, 0x01}) {

        FaceMotionBlock = element[index].payload

    }

}
```

It is important to select the body and face motion elements that are compatible with the body and face geometry IDs of the avatar that will be used for animation.

D.5.2.3 File conversion

The blocks extracted from SLMB file can be converted to some file formats that can be imported to some of the known animation software.

A BVH file may be created from the `BodyMotionBlock()` data. The information needed to fill data in the BVH file can be gotten with the algorithm described in D.2.4.4.

A JSON file may be created from the `FaceMotionBlock()` data. The information needed to fill data in the JSON file can be gotten with the algorithm described in D.2.5.4.

A glTF file may be created from both `BodyMotionBlock()` and `FaceMotionBlock()` data. The information needed to fill data in the JSON file can be gotten with the algorithm described in D.2.6.3.

D.5.3 3D Avatar Engines

Outcome from conversion done in D.4.2 may be used by some animation engines.

BVH format files in the structure defined in D.2.4.1 can be imported by almost all major motion capture software, such as Blender [9], Maya [10] and AR Emoji SDK [6]

Figure D.8 shows the BVH file opened by Blender.

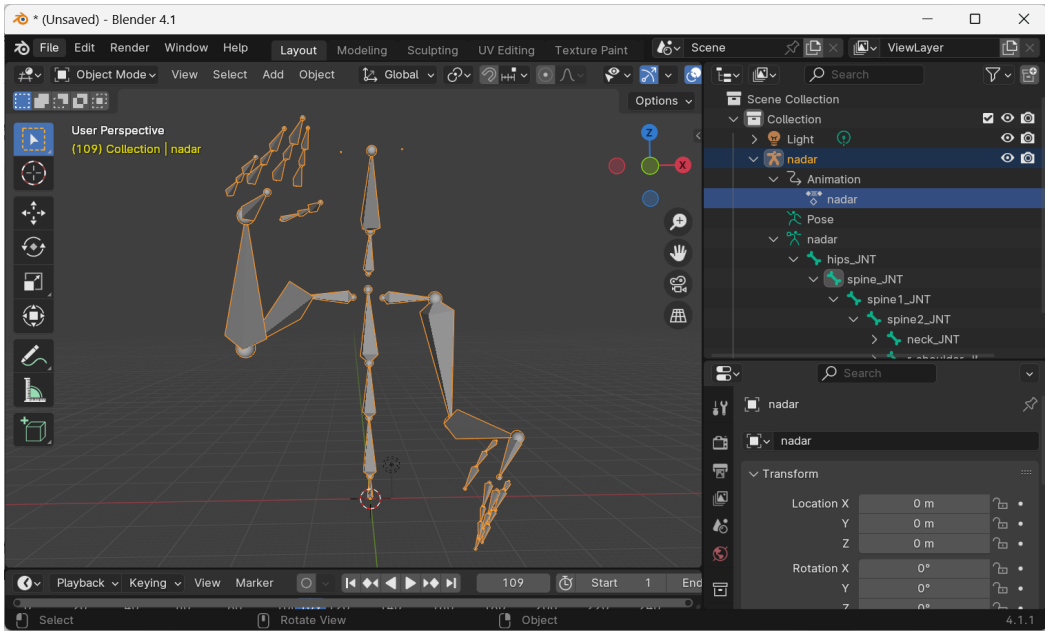

Figure D.8: BVH file imported to Blender software

JSON format files in the structure defined in section D.2.5.2 are recognized by AR Emoji SDK [6].

glTF format files can be imported in almost all major motion capture software, such as Blender [9], Maya [10] and AR Emoji SDK [6]. It can also be viewed by glTF Viewer [11]. Figure D.9 shows a glTF file opened by glTF Viewer.

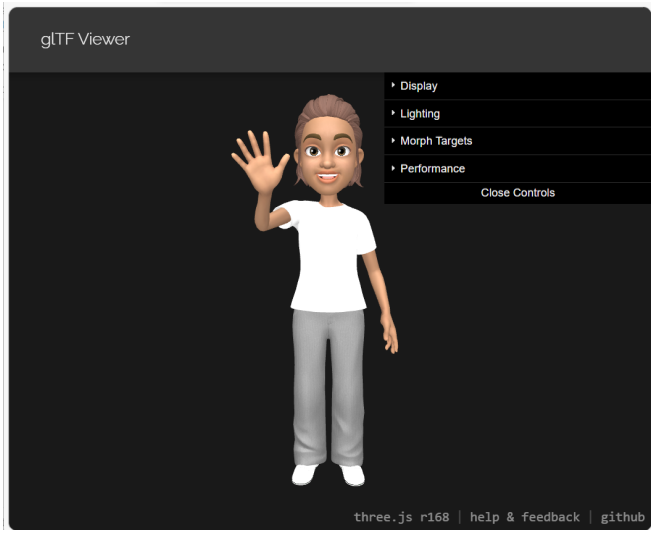

Figure D.9: glTF file imported to glTF Viewer software

A skin may be drawn around the skeleton defined in ABNT NBR 25606 , D.3, so that the visual appearance of avatar is customized in receiver. The skin should be designed to deform according to the skeleton movement.

ABNT NBR 25606 , D.4.4, defines objectively the deformations provided by the blend shapes but is based on a pre-defined face geometry. A different face geometry may be used for animation by receiver, as far as deformations done by the blend shapes in this face geometry are equivalent to the effect seen in reference file. Figure D.10 is an example of the `EyeBlink_Left` blend shape applied to different face geometries.

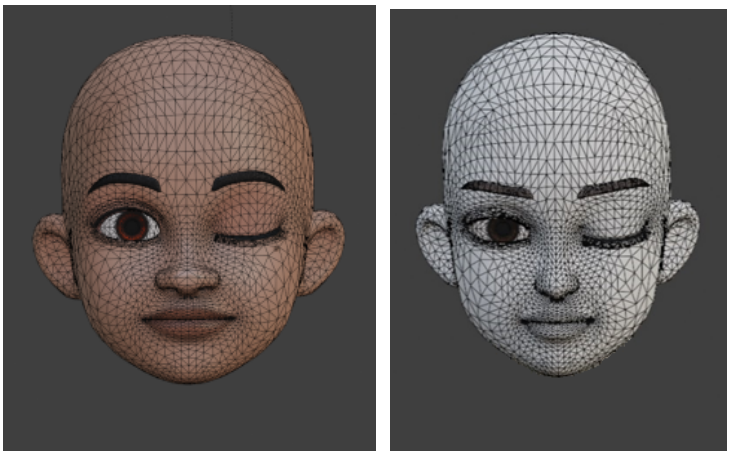

Figure D.10: Example of equivalent blend shape applied in different face geometries

It is recommended to treat the beginning and the end of the body/face movements, so that the transition between the sentences and transition from/to idle state are as smooth as possible.

Bibliography

[1] ABNT NBR 25606 , *TV 3.0 – Closed signing*

[2] **Quaternion to Euler angles conversion: A direct, general and computationally efficient method**, Evandro Bernardes and Stéphane Viollet. Available at <https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0276302>.

[3] **Motion Capture File Formats explained**, M. Meredith, S. Maddock. Available at <https://staffwww.dcs.shef.ac.uk/people/S.Maddock/publications/Motion%20Capture%20File%20Formats%20Explained.pdf>

[4] **BVH Motion Capture Data Animated**, CHAN Ka Chun et al. Available at <https://www.cs.cityu.edu.hk/~howard/Teaching/CS4185-5185-2007-SemA/Group12/BVH.html>

[5] **Biovision BVH**. Available at <https://research.cs.wisc.edu/graphics/Courses/cs-838-1999/Jeff/BVH.html>

[6] **Galaxy AR Emoji SDK for Unity**. Available at <https://developer.samsung.com/galaxy-ar-emoji/overview.html>

[7] **glTF™ 2.0 Specification**. The Khronos® 3D Formats Working Group Version 2.0.1, 2021-10-11 23:01:57Z. Available at <https://registry.khronos.org/glTF/specs/2.0/glTF-2.0.html>

[8] **XZ Utils**. Available at <https://tukaani.org/xz>

[9] **Blender**. Available at <https://www.blender.org>

[10] **Autodesk Maya**. Available at <https://www.autodesk.com/products/maya/overview?term=1-YEAR&tab=subscription>

[11] **glTF Viewer**. Available at < https://gltf-viewer.donmccurdy.com/>

[12] ISO/IEC-14496-30:2018, Information technology – Coding of audio-visual objects – Part 30: Timed text and other visual overlays in ISO base media file format