A video conference system under MPEG-4; Overview of Face Animation in MPEG-4; and Study of the compliance level of Eurecom's Face Animation-Teleconferencing System

Ana C. Andrés del Valle, Jean-Luc Dugelay, Danielle Pélé

March 2001

Joint Research Report
Eurecom: RR-2001-053
FTR&D: FT/BD/DIH/HDM/11/DP

TABLE OF CONTENTS

MPEG-4 GENERAL SYSTEM OVERVIEW

MPEG-4 standardizes the transport of coded audio, video and user-defined data. It adds to MPEG-1 and MPEG-2 the concepts of *audio-visual object* and *scene description*. In fact, MPEG-4 does not think about the data to transport in terms if its nature (video, audio,...) but in terms of what it represents. This new approach allows the mixing of different media in the same environment. For example, synthetic 3D and 2D objects can be correctly blended with natural backgrounds because MPEG-4 provides the means to precisely define the objects and their behavior in the scene. A video stream is not longer a sequence of frames to be coded and transported but a number of elements to wisely code and to concretely situate in a specific scene.

Several standard languages already integrate the concept of *scene description*; that is the case of VRML. MPEG-4 improves their performance by adding the ability to stream the data (we do not need to download the whole scene to see some action), providing ways of synchronization (important for the audio-video coherence) and allowing the setting of different timings for each element (therefore permitting the downloading of an object before it is used).

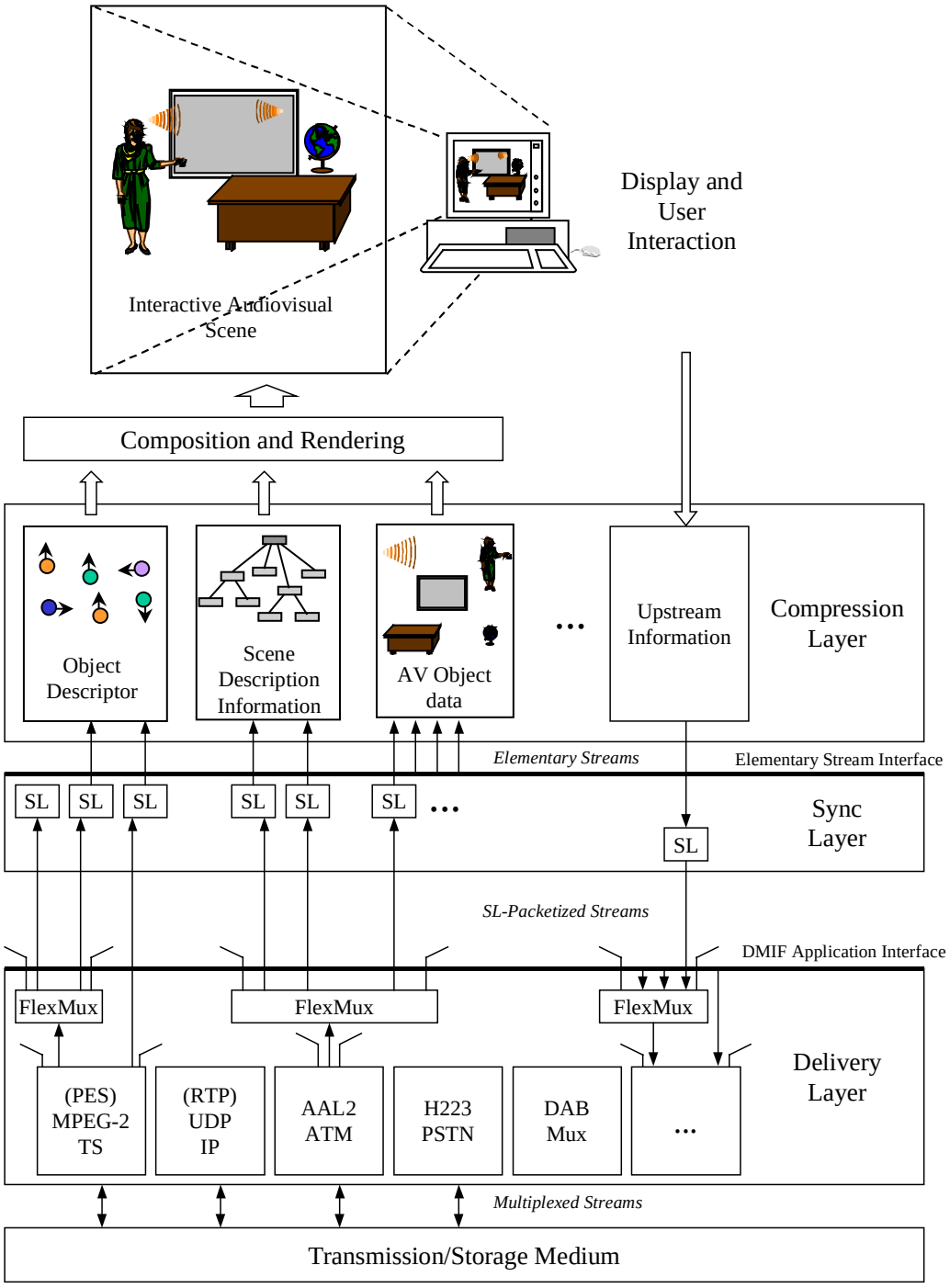
The concept of audio-visual object implies:

1. interaction with the content;
2. reusability of the content and
3. content-based scalability.

For many of the new applications merging from the multimedia technology, these three concepts are important. Taking as an example a virtual teleconference we can estate:

- The user should be able to interact with the conference scene to arrange it as his/her wish (1);
- he should also have the means to store what he has seen (2) and we want to be able to set the system depending on the available resources (3).
- This is done by allowing client interaction with the scene and a client-server feedback interaction.

Architecture: Scene description and object description



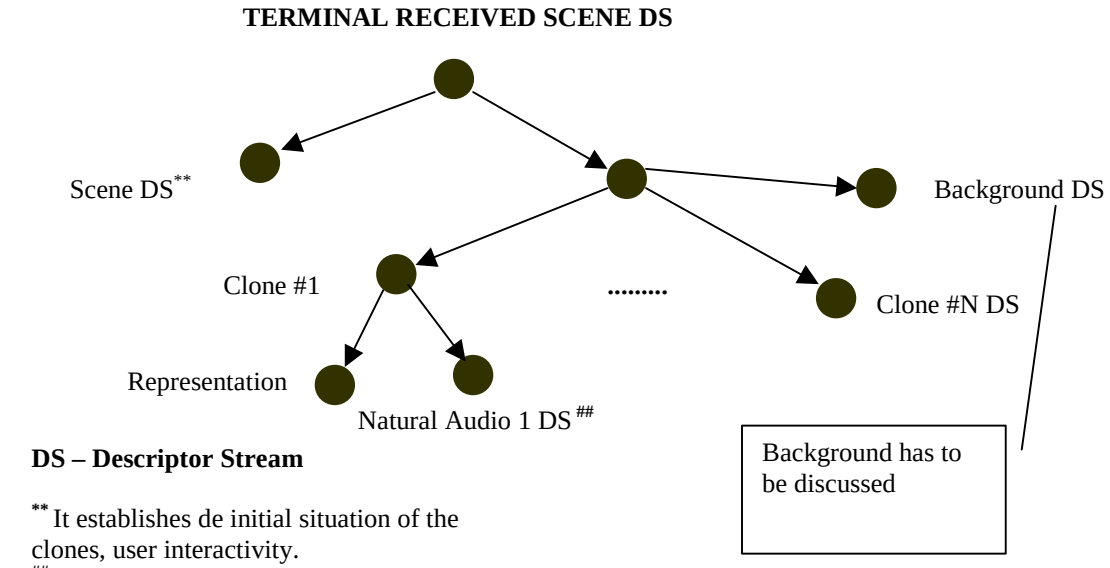
An Interactive Audiovisual Scene is created through the composition and rendition of different objets that we extract from the *Compression Layer*. In this layer we find the Object Descriptors, the Scene Description Information, Audio-Visual Objects Data and other Upchannel Information (feedback from the user). For each one of these components an elementary stream is needed. To create a scene and start working with

it, we need at least the Scene Description Information stream. Underneath the streams we find the *Sync Layer*, which is the sole mechanism of implementing timing and synchronization in MPEG-4. Then the Delivery Layer ensures MPEG-4 capability of using a wide range of delivery systems such as MPEG-2 Transport Streams, UDP over IP, ATM AAL2, etc.

The *initial object descriptor*, a derivative of the object descriptor, is crucial for accessing the MPEG-4 content. In the initial object descriptor we usually find a pointer on to the scene description stream and another on to an object descriptor stream. This object descriptor stream usually transports the object descriptors for the elementary streams that are referred to by some of the components in the scene description. We can "reuse" an initial object descriptor by transporting it as an object descriptor stream from another scene. By nesting initial object descriptors we create indexing points of the complete scene.

The ***Scene description*** is the coding of information that describes the spatio-temporal relationships between the various audio-visual objects present in the complete content. This coded information defines the spatial and temporal position of the objects, their behavior and the interactivity features available to the user. The scene description contains *pointers* to object descriptors when it refers to a particular audio-visual object.

The teleconference framework that we have set in section *Applications* could be designed following the MPEG-4 standard in the following way:



DS – Descriptor Stream

** It establishes de initial situation of the clones, user interactivity.
The amount of audio objects will be eventually determined. A priori, we could assign an audio stream per clone.

BIFS: coding audio-visual objects and scenes

MPEG-4 specifies a binary format for scenes (BIFS) that is used to describe scene composition information. Elements of the scene and the relationships among them form the scene graph that must be coded for transmission. The nuclear graph elements are the *nodes* that describe audio-visual primitives and their attribute. It can be thought as a superset of VRML. Although it does not provide some of VRML's capabilities, MPEG-4 provides better means for animation than VRML. As a main difference we point out that BIFS is a binary format whereas VRML is a textual format.

BIFS encodes scene components, it is used as a scene updating mechanism and it allows scene components to be animated.

BIFS and VRML scenes are both composed of a collection of nodes arranged in hierarchical tree. Each node represents a group or transforms an object in the scene and consists of a list of fields that define the node's behavior. All nodes have labels to be recognized. An important feature of the node is that node fields have default values; this way we can avoid the sending of non-useful information. We can also find the ROUTEs, which are connections that assign the value of one field to another field.

Some nodes can contain other nodes as fields. MPEG-4 has a rigidly typed collection of nodes that specifies exactly which ither nodes can be contained inside other ones. Nodes can be reused without having to be redefined.

1. BIFS-Command:

To load action information over the time MPEG-4 uses the BIFS-Command protocol and the elementary stream that carries it is called BIFS-Command stream. BIFS-Commands can have four main functionalities: scene replacement, node/filed/route insertion, node/value/route deletion, and node/field/value/route displacement.

2. BIFS-Anim: Streaming animators

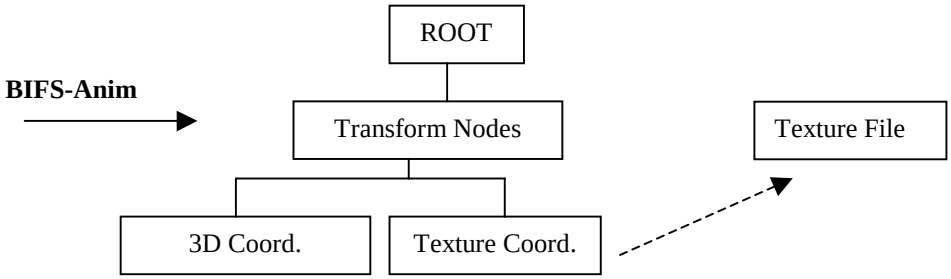
When heavy animation in needed or we seek better compression rates using BIFS-Command is not suitable. In those cases MPEG-4 provides the BIFS-Anim tool as an alternative for streaming the animation. The BIFS-Anim framework works as follows:

- The BIFS scene is loaded with objects that have been defined (DEF) and they have an unique node ID
- The animation mask is loaded, containing the list of nodes and fields to be animated
- The animation stream itself is streamed, containing animation frames in time-stamped access units.

The values for the BIFS-Anim can be initial values (I) or predicted values (P). These values are further arithmetically coded to give high compression.

BIFS provides support at the scene level for MPEG-4 facial animation. A special set of BIFS nodes expose the properties of the animated face. Nevertheless, even if these

special nodes are not used a clone can always be taken as a 3D mesh which is being animated through general BIFS-Anim.



This is an example of a very simplified BIFS 3D object scene description. The *Transform Nodes* describe what kind of interaction and movement the 3D object will have. The *3D coord.* node includes the list of 3D coordinates that build the 3D object. The *Texture coord.* node has the list of the related 2D coords. of the mesh nodes onto a texture file. This texture could also be a 2D object. BIFS structures can be nested in order to build more complex 3D objects. BIFS-Anim streams can be used to animate the object. For example, it could point to a *Transform Node* and by changing one of its values produce a translation.

MPEG-4 defines the syntax to be used to describe the scene. The nodes and the fields inside the nodes have a unique identification name that ultimately maps onto a binary representation. For instance, all BIFS scenes shall begin with a node of type SFTopNode; **Layer2D**, **OrderedGroup**, **Group** or **Layer3D** are just some examples of this kind of node. MPEG-4 also provides some specific nodes and fields for face animation related to the Face Object, for example: **FaceDefTransform** node defines fields **rotation**, **scale** or **translation** of a **Transform** node of **faceSceneGraph**. Its binary representation is:

FaceDefTransform

FaceDefTransform	SFWorldNode SFFaceDefTransformNode					0100111 1		
Field name	***Field type***	***DEF id***	***IN id***	***OUT id***	***DYN id***	***[m, M]***	***Q***	***A***
FaceSceneGraphNode	SF3DNode	000						
FieldId	SFInt32	001						
RotationDef	SFRotation	010				[-I, +I]	10	
ScaleDef	SFVec3f	011					7	
TranslationDef	SFVec3f	100					1	

Our clone animation system has been built taking into account the following constraints:

1. The face analysis and synthesis frame-rates, and the image processing delays, should be as low as possible;
2. The synthetic clones can be rendered from any point of view, and a full 3D model is needed (not only the frontal part of the face);
3. The face cloning system should operate without colored marks taped on the performer's face;
4. It should deal with unknown lighting conditions and background;
5. The motions and rotations of the user should not be restricted in front of the camera;
6. Finally, the clones should be visually realistic.

The clone animation has been developed in two parts: the *global motion estimation* and the *analysis-synthesis of local expressions*. Both parts rely on the high realism of the 3D head models or clones. Our goal is to reach near-real time performance. The use of a realistic model lets graphics hardware translate the information of 3D shape into the 2D image plane, and we only work at the image level, using little 3D information from the model and therefore decreasing the amount of processing related to geometry.

Next sections describe the relationship of our clone animation system to the MPEG-4 standard in terms of current compliance and future adaptation. This study covers the 3D model geometry used and each of the animation parts.

Geometric modeling:

We are currently using range data obtained from cylindrical geometry Cyberware(TM) finders to build person dependent realistic face models. This method is expensive and needs specialized hardware (laser scanners, advanced computers, etc.)

A complete clone is generated from the Cyberware(TM) data set by deforming an initial sphere mesh to approximate the face geometry. The model is refined by selecting those areas where more modeling precision is desired (eyes, mouth,...).

Our animation system is model independent in the sense that the nature of the clone does not influence the results of the animation. There is only one requirement: having enough number of primitives to provide enough lighting normals, which are crucial for the illumination compensation, discussed in section *«Global head motion estimation»*. It should also look as realistic as possible because our animation techniques count on the clone's resemblance to the user.

Currently we are studying 3D data acquisition using inexpensive techniques to substitute the Cyberware(TM) scanners. We believe that the adaptation of a generic face model to given pictures could not give suitable results with our animation algorithms. Present work involves developing a system to obtain 3D shape data using inexpensive and easy to use equipment. This process provides the 3D coordinates from several images of a light grid projected onto a head.

Our models versus MPEG-4 definition of a face model

The MPEG-4 standard includes the Face Object and the Body Object as specific well-defined 3D objects, and adds for them possible animation techniques, although it leaves the designer a great degree of freedom to build the animation system.

MPEG-4 does not provide a specific 3D model to be used. It only specifies a face model in its neutral state, a number of feature points or face description parameters (FDP) on it as reference points, and a set of face animation parameters (FAP), each corresponding to a particular action deforming the face from its neutral position. It also provides the means to "teach" a decoder how a face object should be animated with the use of Face Animation Tables (FAT) and Face Interpolation Table (FIT).

MPEG-4 specifies 84 FDP and they should be located in determined positions because FAPs animate taking as a reference those positions. FAP Units (FAPU) are also given as fixed distances of specific points of our model in neutral state. FAPU allow animation to be relative and therefore model independent.

Our head clones are not MPEG-4 compliant a priori, since they do not meet any of the requirements: there are no concrete location of FDPs and no definition of FAPU. Despite this, the great number of primitives they have makes relatively easy to arrange the nodes so they comply with the specific FDPs; from the FDPs the FAPU can be directly computed. The minimum amount of nodes that a face model should have are 50 (number of animated FDPs), although there should be al least 500 to give pleasant results. Our models currently have much more nodes; it is a requirement to obtain good realistic clones.

3D models generated from scanned data do not usually represent the inner part of the mouth therefore just after being designed our models do not have either teeth or tongue. To implement these parts, crucial for animating the clone while speaking, we have to manually 'cut' the head mesh along the lips middle line and introduce a cylinder shaped plane onto which we have textured the teeth and the tongue. This solution does not really match MPEG-4's description of teeth and tongue. Regarding the teeth, there is no much trouble since the standard only defines 4 FDPs that are not affected by FAPs. On the other hand MPEG-4 has thought about the tongue as a 3D active object therefore it has given it 4 FDPs that are situated in 3D space and affected by FAPs. The standard provides means of defining actions like showing out the tongue or stressing a very concrete phoneme that uses the tongue (i.e. /l/). All the FAPs related to the tongue movements are inside **group 6**, ANNEX B contains more details. In section *Analysis-synthesis of local expressions* we discuss this point more in depth arguing the reasons to have such a model and exposing possible solutions to fit into MPEG-4.

Usually, an MPEG-4 terminal that is able to decode FAP streams has its own proprietary model to animate. It is foreseen the use of other models and therefore the possibility of downloading customized 3D head models using an FDP node. MPEG-4 uses a way to describe the 3D/2D scenes through scene graphs (based on VRML syntax). A node is a part of this graph where some object, characteristic of an object or action is described. In this scene graph we find the static geometry of the face model in its neutral state. The surface properties and the animation rules have also to be provided. The animation rules are given using FAT. Our models are not defined in a scene graph and do not follow a

VRML-like structure. Nevertheless converting the data of our clones into VRML has already been performed; therefore we consider that it should not be a problem to build an MPEG-4 scene graph out of them. FAT will be more difficult to determined since our animation techniques (synthesis of expressions) are not entirely based on the physical movement of mesh nodes or morphing.

FATs define how a model is spatially deformed as a function of the amplitud of the FAPs. Three BIFS nodes provide this functionality: *FaceDefTable*, *FaceDefTransform* and *FaceDefMesh*. These nodes are considered to be part of the face model. Using faceDefTransform nodes and faceDefMesh nodes, the FaceDefTable specifies, for a FAP, which nodes of the scenegraph are animated by it and how.

MPEG-4 also outlines the possibility of only providing a limited number of FDPs and 'adapt' the proprietary model to them. Our system does not include any model adaptation since our first priority is high realism and adapting could generate undesirable results. MPEG-4 also points out that in case of not being able of 'adapting', 'calibrating' or 'generating' a model from the FDPs with a minimum quality guarantied the decoder should ignore those specifications and run its proprietary model with its animating rules. This last one is the ideal situation of our head animation system.

At the moment, our models are just a head-neck entity. Future plans include joining them with the upper part of the body so they create a pleasant and 'real' representation of a person. There is no intention yet of animating the upper part of the body. At this point we will have the option of taking it as a *body object* or just a *3D mesh.* It seems quite clear that only for the upper part of the body and if not complicated interaction is expected (for instance, it only follows head translations and rotations) taking it as a 3D mesh should be enough. Determining this point is out of the scope of this report.

In compliance with MPEG-4 we could also look at our model from another completely different perspective. We could think of it as just a 3D general mesh to be animated. In such a case, none of the requirements to be considered a *face object* would be necessary, but we would not be able to take advantage of the special treatment 3D head models have in the standard. It is a matter of research and deeper study to decide weather the first approach, which seems the most direct, is better or worse than the second one, which is more general. The use of one approach or the other will finally be decided on the animation techniques we use and their behavior under an MPEG-4 system.

By applying techniques that do not rely on the 3D nature of the model we keep MPEG-4 philosophy of model independence. Several tests have successfully been carried out to see the performance of the global head motion tracking algorithms with models constructed with low cost realistic modeling techniques.

Global head motion estimation

The global head motion estimation uses an analysis-synthesis cooperation. The designed system proceeds as follows:

1. ***Initialization***:

During this step the user aligns its head with her/his model and the program runs a 3D illumination compensation algorithm to estimate the lighting parameters that will reduce the photometric differences between the synthetic clone and the real head in the user environment.

2. ***Main loop***:

- A Kalman filter predicts the head's 3D position and orientation for time *t*.
- The synthetic model generates an approximation of the way the real face will appear in the video at time *t*.
- Patterns of contrasted facial features (eyes, eyebrows, nostrils,...) are extracted and matched with the users facial features in the real video frame.
- The system passes the 2D coordinates to the Kalman filter, which estimates the current head's 3D position and orientation. From here we go back to the starting point.

With this analysis-synthesis cooperation we are able to obtain six parameters, the 3 that represent the translation and the 3 that represent the rotation, following the X, Y, Z axes of the model synthetic world.

These parameters have a partial equivalence as FAPs in MPEG-4.

In the *face object* there is an FAP group called **head rotation (7)** that contains the 3 FAPs that define the rotation of the head. These rotations are relative to the perspective of the head model (having as point (0,0,0) the center of the model), and they are described as follows:

Head_pitch (48) [unit: AU] [Bidireccional] [start: down]
Head_yaw (49) [unit: AU] [Bidireccional] [start: left]
Head_roll (50) [unit: AU] [Bidireccional] [start: right]

An AU (Angle Unit) is equivalent to 10^{-5} radians.

For the 3 translation parameters there is no equivalence. In fact, the translation is applied to the entire head and therefore could be treated as a *Transform Node* situated above the *head object* in the complete scene graph. This translation is seen as a transform on the object and not as a specific movement of some nodes. For MPEG-4 the rotation of the head does not include the neck (as it happens in real life) whereas in our system the rotation is also applied to the neck. The exact animation of the joint head-neck will be considered when adding the upper part of the body.

If our clone is considered just a 3D object or a composition of 3D objects no special treatment will be needed. The use of Transform Nodes on top of the parts that we want to animate (in our system the global motion means the animation of the entire head as a unit) should ensure a correct behavior.

Analysis-synthesis of local expressions

We use view-based analysis techniques to estimate facial expressions. Their limitation mainly stays in the low number and the quality of training keyframes obtained by real users. Our approach is to replace the real user by the highly realistic clone during the training process so we obtain better training conditions for the system. The current implementation can only distinguish expressions from the front view of the user; nevertheless, to extend this technique to other views is more a matter of computer and programming skills than theoretical development.

The complete system follows these steps:

1. ***Training***:

Using a person-dependent clone, we optimally sample the visual space of facial expressions, via an animation database (defined by our own FAPs), to produce a synthetic image database. We reduce this image database to a set of vectors that characterize the facial expression via a simple correlation mechanism, which gives a compact parameterization vector ($\boldsymbol{\lambda}$). At last, a clone-dependent estimator learns the relationship between the FAPs and the λs.

2. ***Analysis of an unknown expression***:

After the synthetic clone trains the system, the analysis procedure extracts the corresponding features, parameterizes them with their eigenfigures (λ) and interprets them with the corresponding estimator (giving out FAPs)

3. ***Synthesis of facial expressions:***

For the FAP synthesis we use more than one technique:

a) Mesh morphing: Also called mesh morphing, it consists of interpolating the positions of the mesh vertices between extreme facial expressions. This technique is used to animate the eyelids and the mouth.
b) Animation of tecture coodinates: This technique consists in sliding the texture on top of the mesh without changing the model shape. It has been very convenient for the animation of the eyebrows.
c) Texture displacements: We alter the cylindrical texture mapped onto the mesh vertices at rendition time to produce further animations. We use this technique to control the eye movements. We are also using this technique to implement the model's teeth and tongue by overlapping several texture portions on a plane just behind the model's lips. As stated when discussing the compliance of our 3D clone with MPEG-4, this procedure doesn't provide means to articulate the tongue the way MPEG-4 permits. On the other hand, this solution has the advantage of being more realistic than using generic primitives accounting for the teeth and the tongue of an individual person.
d) exture blending It is possible to blend several textures together to produce a new one. For example, you can fade wrinkles into the model texture at a low cost in

terms of real-time animation, instead of hard-coding them in heavy spline-based meshes.

All these techniques are standard-independent which means that they only represent a way to synthesize FAPs as fast and realistically as possible.

Our current system was built not taking into account MPEG-4 definition of FAPs. These are the FAPs we have designed:

H_movement_LeftEye (1)
V_movement_LeftEye (2)
H_movement_RightEye (3)
V_movement_RightEye (4)
V_movement_LeftHalf_LeftEyebrow (5)
V_movement_RightHalf_LeftEyebrow (6)
V_movement_LeftHalf_RightEyebrow (7)
V_movement_RightHalf_RightEyebrow (8)
H_movement_RightHalf_Mouth(9)
H_movement_LeftHalf_Mouth(10)
V_movement_LowerTeeth(11)
V_movement_LeftHalf_Mouth(12)
V_movement_RightHalf_Mouth(13)
V_movement_LeftHalf_UpperLip(15)
V_movement_RightHalf_UpperLip(16)
V_movement_LowerLip(17)

Compared to the 63 (68 – Viseme – Expression – 3 rotations) FAPs that the MPEG-4 standard provides, our list has a fairly small number of defined actions. Some of our FAPs content more than just one MPEG-4 FAP action. Studying in depth the options we have and how to better relate our concept of FAP to the one MPEG-4 defines can give more versatility to the system. Look at ANNEX A for more details on how our FAPs are mapped on MPEG-4 FAPs.

MPEG-4 allows the modeling and animation of cartoon-like characters by introducing exaggerated values for the FAPs. Our system is conceived to generate realistic animation. Analysis and synthesis are thought in separate but intimately related parts and no experiment has been set to see how our system behaves when introducing unexpected λ values.

Taken as a 3D object, our system would possibly lose part of its strength because several of our synthesis techniques are not so easy to translate into MPEG-4 commands or actions. MPEG-4 tools for animating scene objects, whether they are BIFS-Command or BIFS-Anim, are meant to animate by changing the values of fields of concrete nodes (for instance, a Transform Node). Regarding 3D meshes those values could be the 3D coordinates of the mesh nodes, to indicate their new situation. Some of our synthesis techniques do not rely on mesh movements but on texture displacements. A priori, coding all the information that is related to such techniques would imply much more effort if implemented with BIFS-Anim than if we keep it transparent to the decoder by using FAPs. The animation of the clones will always be somewhat dependent on the techniques we use. It would be necessary a well-structured and correct definition of our Face Object and its characteristics if a 100% MPEG-4 compliance is sought.

All main image processing should be integrated into the encoder leaving the network little data to transport. The training process can be seen as the setting up of the program, it is only needed once in the system life and it is a completely isolated process. It only gives the reference database for analysis-synthesis of the expressions, and that is an initial part of the system database. This is someway related to the notions FAT and FIT under MPEG-4. For a concrete model concrete animation rules can be defined.

The difference between FAT and FIT is that an FAT defines the way some FAPs should act upon the face object (movements of the vertices, type of transform, etc.) and FIT define the way to interpolate the values of some FAPs out of the values of previously defined and animated FAPs. FIT are very helpful to decrease the number of FAP to send when specifying animation for a concrete head model. For instance, the top inner FAPs can be sent to determine the top outer lip FAPs. FAPs for one eye can be used to animate the other one in case of symmetry. We can assume symmetry in all those cases where only the FAPs of one of the eyes are sent (and no weird behavior is permitted).

Future work in the face expression animation part includes:

- improving the current face animation system (mostly in the database generation and retrieval) to get better results;
- defining and narrowing face animation parameters to gain control over the movements (taking advantage of the use of the clones in the training process);
- developing a more automatic way of training the system to ease the current one which requires a lot of manual work;
- looking into new ways of synthesized face expressions and introduce the ones that have already developed into the system[*];
- blending the face expressions with the global tracking in such a way that they do not interfere with each other.

[*] In autumn 2000 there will a student project for the MM department to build an MPEG-4 compliant animator. Firstly, it will be developed using the default morphing techniques. Later, we will study how to map our realistic techniques on the general MPEG-4 system.

APPLICATIONS

Being able to analyze the facial expressions of a human face in a video sequence and reproduce them on a synthetic head using a compact set of FAPs is of tremendous importance for many multimedia applications like model-based coding, virtual actors, human-machine communication, interactive environments, video-telephony and virtual teleconferencing.

The aim of our research is developing an analysis-synthesis system that will not be dependent on the architecture of the application it will be used for. Introducing the standard MPEG-4 implies independence at the decoder. Our next step is to build a video to FAP converter as general and platform independent as possible.

This section describes one specific application that involves the most general environment: a virtual teleconference. This application has been the first scenery onto which we have applied the techniques regarding face cloning developed at the Image Group laboratory.

POSSIBLE FRAMEWORK FOR A VIRTUAL TELECONFERENCE

Visual Part:

GRAPHICS

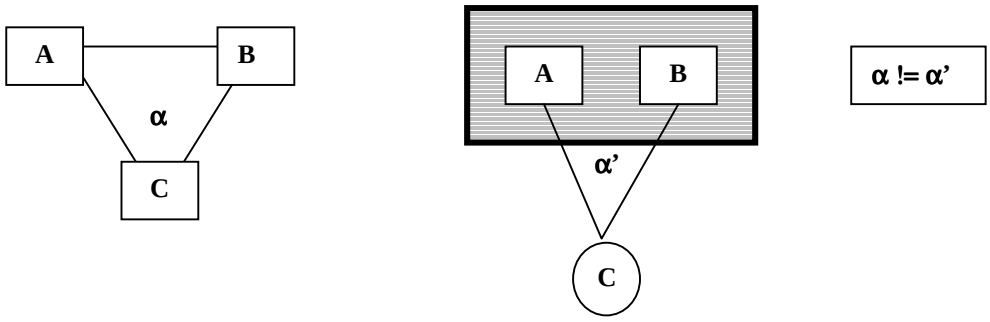
Current contents:

- BACKGROUND:
  3 meshes and a reference texture to be sent.
- CLONES:
  3D head model → cylindrical piece of mouth-teeth
  \+ *Texture*
  ↓
  Cylindrical pieces for eyes → *texture for teeth and eyes*

Setting up the session (3 simultaneous users as an example):

COMMANDS

1. Load the 3D head model from the other users (either locally stored or downloaded from a server)
2. Load the teleconference scene → background (meshes + texture)
3. Select the situation of the users → real life versus limited area representation



Physical theoretical situation Physical practical situation, user in front of the screen

The situation of the users must be coherent. Users should have the same reference and relative position among them. This way, global movements are obtained from the user reaction towards the synthesized models on the screen. For instance, if A is to be situated on the right of B, B should have A on the left in all representations. This way, if A looks at where B is situated in his/her screen, A's model should turn towards B in the screens of the rest of the users, and at the same B should become the main interest point for A's screen.

ACTION

- Render background depending on the perspective
- Render the neutral clones in selected situation

} Specific for each user

Procedure during the teleconference session:

COMMANDS

1. Different global scene movements that will influence the overall situation of the clones and the background will appear. As an example, we may choose just to show one of our session partners.
2. Depending on their face expression, what they say, how and where they look, the clones of the user should be rendered conveniently → Interaction between analysis/synthesis for all the users. (This has to be handled interactively among the users)

FROM THE ANALYSIS YOU GET **N** PARAMETERS → WHILE SYNTHESIZING YOU USE YOUR PARAMETERS FOR GENERAL PERSPERTIVE AND THE PARAMETERS FROM THE OTHERS FOR SPECIFIC MOVEMENTS

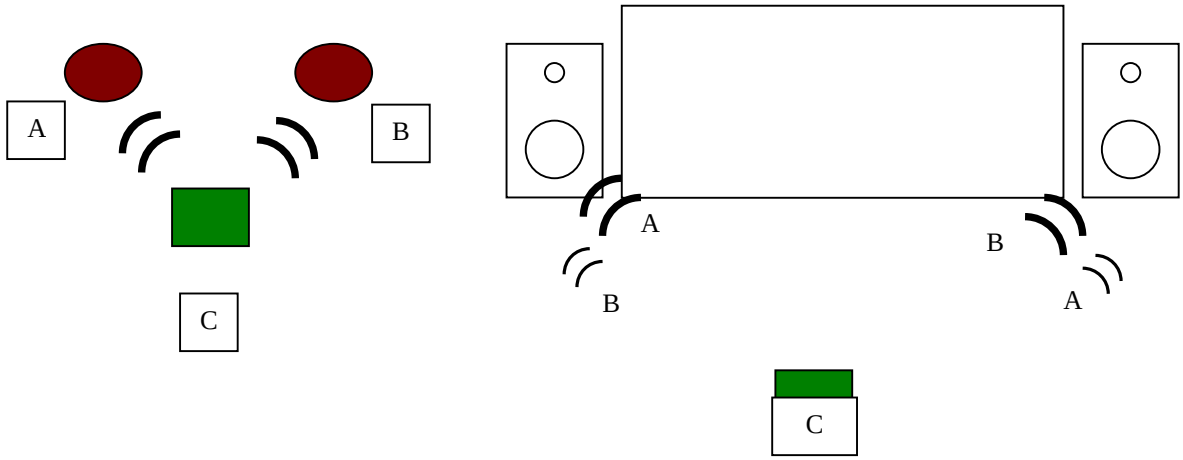
ACTION

The main action during teleconference will be to properly render the synthesized global displacements and concrete face expressions of each clone. On top of this, the teleconference background and the clones of the users will move depending on the situation of the complete scene.

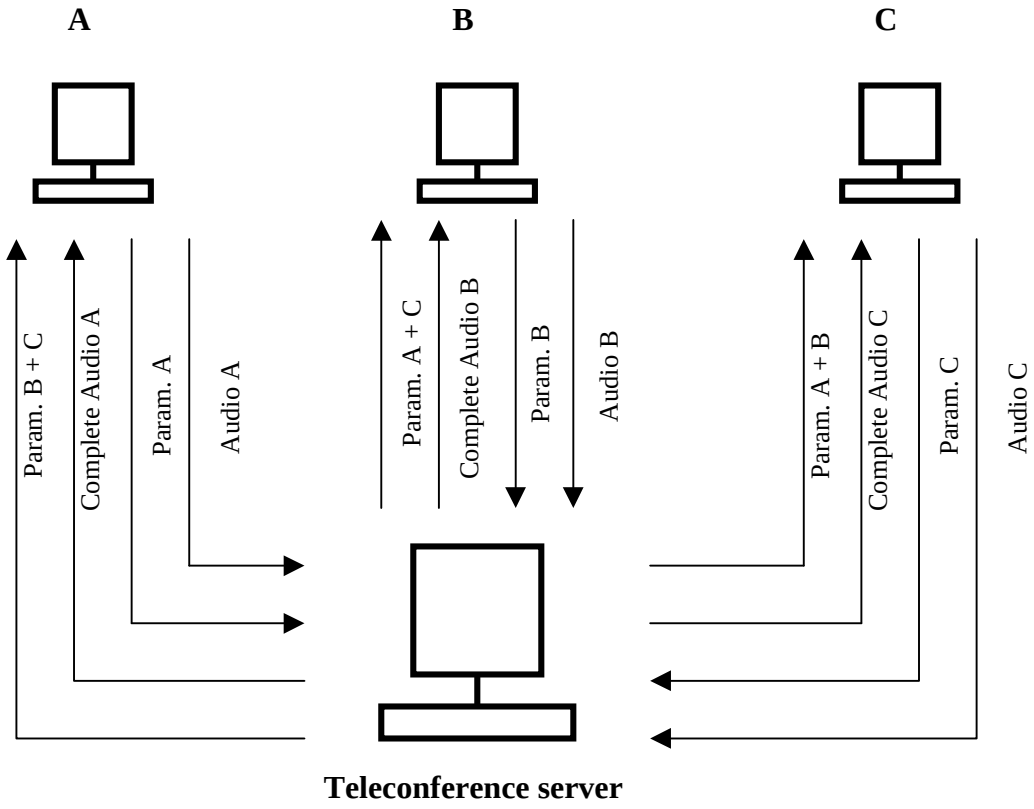
Audio Part:

To maintain the same spatialization concept → applying sound variations depending on the situation (stereo audio sensing).

1. In neutral state the sound should come from the situation of the user in reference with the initial situation of the other users (left, right, from the front and so on).
2. As soon as the users start moving, we should extract the sound orientation from their 3D situation.



Example of a possible implementation:



Terminals A, B, C should deal with the analysis of user movements (generation of parameters to be sent) and the synthesis of the Animation Parameters received from the others. No sound processing is required.

The **Teleconference server** should deal with those no direct animation tasks as setting up the session (connection with the users, handling of their initial position and so on), adding new users,... No direct analysis or synthesis is needed, good data streaming and possible processing of parameters may be required. Since the server is the one to control the situation of the users it could deal with the audio[1] (not to overload the conference terminals)

The terminals and the server do a lot of processing and synthesis therefore the amount of data to be sent (parameters + audio) can be very low. This system implementation can be applied low-rate networks.

[1]The procedure for the audio has not yet been studied, the approach taken so far is to work with it as an audio data stream, and let the server arrange all possible processing required for the sound spatialization. This way the separation between terminals and server can also be seen as separation between the Visual part and the Audio part. On top of this the proper communication system has to be defined.

Annex A

Current FAPs versus MPEG-4 FAPs

This is a compilation of all the MPEG-4 FAPs together with a brief note related to their implementation in our complete animation system.

MPEG-4 has been designed to allow exaggerated movements and animation that is not natural for human beings. In a teleconference environment such freedom is not necessary. MPEG-4 establishes that to be compliant with the standard all FAPS have to be understood, such an effort may be not convenient. A solution would be to accept those FAPs that are not implemented but ignore them since they cannot be animated. Would this system still be MPEG-4 compliant? Maybe not, but priority is to reduce complexity on the encoder and decoder side of our system, where the animation analysis and synthesis will take place.

#	FAP name	FAP description	units	Uni-orBi dir	Pos Motion	FAP in our system and relationship with MPEG-4
1	Viseme	Set of values determining the mixture of two visemes for this frame (e.g. pbm, fv, th)	na	na	Na	At present, it is not contemplated in the system. We do not study mouth animation from the phoneme perspective therefore it is not in our scope.
2	expression	A set of values determining the mixture of two facial expression	Na	na	Na	It is not contemplated in our current system. A priori, should not be very difficult to implement. It is a matter of designing a correct combination of movements to create common face expressions like : happiness, sadness, anger ....
3	open_jaw	Vertical jaw displacement (does not affect mouth opening)	MNS	U	Down	Not contemplated.
4	lower_t_midlip	Vertical top middle inner lip displacement	MNS	B	Down	Someway included in: H_movement_RightHalf_Mouth(9) H_movement_LeftHalf_Mouth(10) V_movement_LeftHalf_Mouth(12) V_movement_RightHalf_Mouth(13)

						V_movement_LeftHalf_UpperLip (15) V_movement_RightHalf_UpperLip(16) V_movement_LowerLip(17)
5	raise_b_midlip	Vertical bottom middle inner lip displacement	MNS	B	Up	Someway included in: H_movement_RightHalf_Mouth(9) H_movement_LeftHalf_Mouth(10) V_movement_LeftHalf_Mouth(12) V_movement_RightHalf_Mouth(13) V_movement_LeftHalf_UpperLip (15) V_movement_RightHalf_UpperLip(16) V_movement_LowerLip(17)
6	stretch_l_cornerlip	Horizontal displacement of left inner lip corner	MW	B	left	Someway included in: H_movement_RightHalf_Mouth(9) H_movement_LeftHalf_Mouth(10) V_movement_LeftHalf_Mouth(12) V_movement_RightHalf_Mouth(13) V_movement_LeftHalf_UpperLip (15) V_movement_RightHalf_UpperLip(16) V_movement_LowerLip(17)
7	stretch_r_cornerlip	Horizontal displacement of right inner lip corner	MW	B	right	Someway included in: H_movement_RightHalf_Mouth(9) H_movement_LeftHalf_Mouth(10) V_movement_LeftHalf_Mouth(12) V_movement_RightHalf_Mouth(13) V_movement_LeftHalf_UpperLip (15) V_movement_RightHalf_UpperLip(16) V_movement_LowerLip(17)

8	lower_t_lip_lm	Vertical displacement of midpoint between left corner and middle of top inner lip	MNS	B	down	Someway included in: H_movement_RightHalf_Mouth(9) H_movement_LeftHalf_Mouth(10) V_movement_LeftHalf_Mouth(12) V_movement_RightHalf_Mouth(13) V_movement_LeftHalf_UpperLip(15) V_movement_RightHalf_UpperLip(16) V_movement_LowerLip(17)
9	lower_t_lip_rm	Vertical displacement of midpoint between right corner and middle of top inner lip	MNS	B	down	Someway included in: H_movement_RightHalf_Mouth(9) H_movement_LeftHalf_Mouth(10) V_movement_LeftHalf_Mouth(12) V_movement_RightHalf_Mouth(13) V_movement_LeftHalf_UpperLip(15) V_movement_RightHalf_UpperLip(16) V_movement_LowerLip(17)
10	raise_b_lip_lm	Vertical displacement of midpoint between left corner and middle of bottom inner lip	MNS	B	up	Someway included in: H_movement_RightHalf_Mouth(9) H_movement_LeftHalf_Mouth(10) V_movement_LeftHalf_Mouth(12) V_movement_RightHalf_Mouth(13) V_movement_LeftHalf_UpperLip(15) V_movement_RightHalf_UpperLip(16) V_movement_LowerLip(17)
11	raise_b_lip_rm	Vertical displacement of midpoint between right corner and middle of bottom	MNS	B	up	Someway included in: H_movement_RightHalf_Mouth(9) H_movement_LeftHalf_Mouth(10)

		inner lip				V_movement_LeftHalf_Mouth(12) V_movement_RightHalf_Mouth(13) V_movement_LeftHalf_UpperLip(15) V_movement_RightHalf_UpperLip(16) V_movement_LowerLip(17)
12	raise_l_cornerlip	Vertical displacement of left inner lip corner	MNS	B	up	H_movement_RightHalf_Mouth(9) H_movement_LeftHalf_Mouth(10) V_movement_LeftHalf_Mouth(12) V_movement_RightHalf_Mouth(13) V_movement_LeftHalf_UpperLip(15) V_movement_RightHalf_UpperLip(16) V_movement_LowerLip(17)
13	raise_r_cornerlip	Vertical displacement of right inner lip corner	MNS	B	up	Someway included in: H_movement_RightHalf_Mouth(9) H_movement_LeftHalf_Mouth(10) V_movement_LeftHalf_Mouth(12) V_movement_RightHalf_Mouth(13) V_movement_LeftHalf_UpperLip(15) V_movement_RightHalf_UpperLip(16) V_movement_LowerLip(17)
14	thrust_jaw	Depth displacement of jaw	MNS	U	forward	Not contemplated
15	shift_jaw	Side to side displacement of jaw	MW	B	right	Not contemplated
16	push_b_lip	Depth displacement of bottom middle lip	MNS	B	forward	Someway included in: H_movement_RightHalf_Mouth(9) H_movement_LeftHalf_Mouth(10) V_movement_LeftHalf_Mouth(12)

						V_movement_RightHalf_Mouth(13) V_movement_LeftHalf_UpperLip(15) V_movement_RightHalf_UpperLip(16) V_movement_LowerLip(17)
17	push_t_lip	Depth displacement of top middle lip	MNS	B	forward	Someway included in: H_movement_RightHalf_Mouth(9) H_movement_LeftHalf_Mouth(10) V_movement_LeftHalf_Mouth(12) V_movement_RightHalf_Mouth(13) V_movement_LeftHalf_UpperLip(15) V_movement_RightHalf_UpperLip(16) V_movement_LowerLip(17)
18	depress_chin	Upward and compressing movement of the chin (like in sadness)	MNS	B	up	Not contemplated
19	close_t_l_eyelid	Vertical displacement of top left eyelid	IRISD	B	down	Not contemplated
20	close_t_r_eyelid	Vertical displacement of top right eyelid	IRISD	B	down	Not contemplated
21	close_b_l_eyelid	Vertical displacement of bottom left eyelid	IRISD	B	up	Not contemplated
22	close_b_r_eyelid	Vertical displacement of bottom right eyelid	IRISD	B	up	Not contemplated
23	yaw_l_eyeball	Horizontal orientation of left eyeball	AU	B	left	H_movement_LeftEye (1)
24	yaw_r_eyeball	Horizontal orientation of right eyeball	AU	B	left	H_movement_RightEye (3)
25	pitch_l_eyeball	Vertical orientation	AU	B	down	V_movement_LeftEye (2)

		of left eyeball				
26	pitch_r_eyeball	Vertical orientation of right eyeball	AU	B	down	V_movement_RightEye (4)
27	thrust_l_eyeball	Depth displacement of left eyeball	ES	B	forward	Not contemplated. This is an unrealistic movement.
28	thrust_r_eyeball	Depth displacement of right eyeball	ES	B	forward	Not contemplated. This is an unrealistic movement.
29	dilate_l_pupil	Dilation of left pupil	IRISD	B	growing	Not contemplated
30	dilate_r_pupil	Dilation of right pupil	IRISD	B	growing	Not contemplated
31	raise_l_i_eyebrow	Vertical displacement of left inner eyebrow	ENS	B	up	Someway included in: V_movement_RightHalf_LeftEyebrow (6)
32	raise_r_i_eyebrow	Vertical displacement of right inner eyebrow	ENS	B	up	Someway included in: V_movement_LeftHalf_RightEyebrow (7)
33	raise_l_m_eyebrow	Vertical displacement of left middle eyebrow	ENS	B	up	Someway included in: V_movement_LeftHalf_LeftEyebrow (5) V_movement_RightHalf_LeftEyebrow (6)
34	raise_r_m_eyebrow	Vertical displacement of right middle eyebrow	ENS	B	up	Someway included in: V_movement_LeftHalf_RightEyebrow (7) V_movement_RightHalf_RightEyebrow (8)
35	raise_l_o_eyebrow	Vertical displacement of left outer eyebrow	ENS	B	up	Someway included in: V_movement_LeftHalf_LeftEyebrow (5)
36	raise_r_o_eyebrow	Vertical displacement of right outer eyebrow	ENS	B	up	Someway included in: V_movement_RightHalf_RightEyebrow (8)
37	squeeze_l_eyebrow	Horizontal displacement of left eyebrow	ES	B	right	Someway included in: V_movement_LeftHalf_LeftEyebrow (5)

						V_movement_RightHalf_LeftEyebrow (6) V_movement_LeftHalf_RightEyebrow (7) V_movement_RightHalf_RightEyebrow (8)
38	squeeze_r_eyebrow	Horizontal displacement of right eyebrow	ES	B	left	Someway included in: V_movement_LeftHalf_LeftEyebrow (5) V_movement_RightHalf_LeftEyebrow (6) V_movement_LeftHalf_RightEyebrow (7) V_movement_RightHalf_RightEyebrow (8)
39	puff_l_cheek	Horizontal displacement of left cheeck	ES	B	left	Not contemplated
40	puff_r_cheek	Horizontal displacement of right cheeck	ES	B	right	Not contemplated
41	lift_l_cheek	Vertical displacement of left cheek	ENS	U	up	Not contemplated
42	lift_r_cheek	Vertical displacement of right cheek	ENS	U	up	Not contemplated
43	shift_tongue_tip	Horizontal displacement of tongue tip	MW	B	right	Not contemplated
44	raise_tongue_tip	Vertical displacement of tongue tip	MNS	B	up	Not contemplated
45	thrust_tongue_tip	Depth displacement of tongue tip	MW	B	forward	Not contemplated
46	raise_tongue	Vertical displacement of tongue	MNS	B	up	Not contemplated
47	tongue_roll	Rolling of the tongue into U shape	AU	U	concave upward	Not contemplated
48	head_pitch	Head pitch angle from top of spine	AU	B	down	Rotation parameter from the Head Tracking

49	head_yaw	Head yaw angle from top of spine	AU	B	left	Rotation parameter from the Head Tracking
50	head_roll	Head roll angle from top of spine	AU	B	right	Rotation parameter from the Head Tracking
51	lower_t_midlip _o	Vertical top middle outer lip displacement	MNS	B	down	Someway included in: H_movement_RightHalf_Mouth(9) H_movement_LeftHalf_Mouth(10) V_movement_LeftHalf_Mouth(12) V_movement_RightHalf_Mouth(13) V_movement_LeftHalf_UpperLip(15) V_movement_RightHalf_UpperLip(16) V_movement_LowerLip(17)
52	raise_b_midlip_o	Vertical bottom middle outer lip displacement	MNS	B	up	Someway included in: H_movement_RightHalf_Mouth(9) H_movement_LeftHalf_Mouth(10) V_movement_LeftHalf_Mouth(12) V_movement_RightHalf_Mouth(13) V_movement_LeftHalf_UpperLip(15) V_movement_RightHalf_UpperLip(16) V_movement_LowerLip(17)
53	stretch_l_cornerlip_o	Horizontal displacement of left outer lip corner	MW	B	left	Someway included in: H_movement_RightHalf_Mouth(9) H_movement_LeftHalf_Mouth(10) V_movement_LeftHalf_Mouth(12) V_movement_RightHalf_Mouth(13) V_movement_LeftHalf_UpperLip(15) V_movement_RightHalf_UpperLip(16) V_movement_LowerLip(17)

54	stretch_r_cornerlip_o	Horizontal displacement of right outer lip corner	MW	B	right	Someway included in: H_movement_RightHalf_Mouth(9) H_movement_LeftHalf_Mouth(10) V_movement_LeftHalf_Mouth(12) V_movement_RightHalf_Mouth(13) V_movement_LeftHalf_UpperLip(15) V_movement_RightHalf_UpperLip(16) V_movement_LowerLip(17)
55	lower_t_lip_lm _o	Vertical displacement of midpoint between left corner and middle of top outer lip	MNS	B	down	Someway included in: H_movement_RightHalf_Mouth(9) H_movement_LeftHalf_Mouth(10) V_movement_LeftHalf_Mouth(12) V_movement_RightHalf_Mouth(13) V_movement_LeftHalf_UpperLip(15) V_movement_RightHalf_UpperLip(16) V_movement_LowerLip(17)
56	lower_t_lip_rm _o	Vertical displacement of midpoint between right corner and middle of top outer lip	MNS	B	down	Someway included in: H_movement_RightHalf_Mouth(9) H_movement_LeftHalf_Mouth(10) V_movement_LeftHalf_Mouth(12) V_movement_RightHalf_Mouth(13) V_movement_LeftHalf_UpperLip(15) V_movement_RightHalf_UpperLip(16) V_movement_LowerLip(17)
57	raise_b_lip_lm_o	Vertical displacement of midpoint between left corner and middle of bottom	MNS	B	up	Someway included in: H_movement_RightHalf_Mouth(9) H_movement_LeftHalf_Mouth(10)

		outer lip				V_movement_LeftHalf_Mouth(12) V_movement_RightHalf_Mouth(13) V_movement_LeftHalf_UpperLip(15) V_movement_RightHalf_UpperLip(16) V_movement_LowerLip(17)
58	raise_b_lip_rm_o	Vertical displacement of midpoint between right corner and middle of bottom outer lip	MNS	B	up	Someway included in: H_movement_RightHalf_Mouth(9) H_movement_LeftHalf_Mouth(10) V_movement_LeftHalf_Mouth(12) V_movement_RightHalf_Mouth(13) V_movement_LeftHalf_UpperLip(15) V_movement_RightHalf_UpperLip(16) V_movement_LowerLip(17)
59	raise_l_cornerlip_o	Vertical displacement of left outer lip corner	MNS	B	up	Someway included in: H_movement_RightHalf_Mouth(9) H_movement_LeftHalf_Mouth(10) V_movement_LeftHalf_Mouth(12) V_movement_RightHalf_Mouth(13) V_movement_LeftHalf_UpperLip(15) V_movement_RightHalf_UpperLip(16) V_movement_LowerLip(17)
60	raise_r_cornerlip _o	Vertical displacement of right outer lip corner	MNS	B	up	Someway included in: H_movement_RightHalf_Mouth(9) H_movement_LeftHalf_Mouth(10) V_movement_LeftHalf_Mouth(12) V_movement_RightHalf_Mouth(13) V_movement_LeftHalf_UpperLip(15)

						V_movement_RightHalf_UpperLip(16) V_movement_LowerLip(17)
61	stretch_l_nose	Horizontal displacement of left side of nose	ENS	B	left	Not contemplated.
62	stretch_r_nose	Horizontal displacement of right side of nose	ENS	B	right	Not contemplated
63	raise_nose	Vertical displacement of nose tip	ENS	B	up	Not contemplated
64	bend_nose	Horizontal displacement of nose tip	ENS	B	right	Not contemplated
65	raise_l_ear	Vertical displacement of left ear	ENS	B	up	Not contemplated
66	raise_r_ear	Vertical displacement of right ear	ENS	B	up	Not contemplated
67	pull_l_ear	Horizontal displacement of left ear	ENS	B	left	Not contemplated
68	pull_r_ear	Horizontal displacement of right ear	ENS	B	right	Not contemplated

Annex B

MPEG-4 overview: FACE ANIMATION ON ISO/IEC 14496

Data about the standard

ISO/IEC JTC1/SC29/WG11
MPEG-4 SNHC Working group.
Group on Face/Body animation

Version 1 – Facial Animation definition – FINISHED
Body Animation definition – FINISHED

ISO/IEC 14496-1: Systems
ISO/IEC 14496-2: Visual
ISO/IEC 14496-3: Audio

MPEG-4 is an object-based multimedia compression standard, which allows for encoding of different audio-visual objects (AVO) in the scene independently. The visual objects may have natural or synthetic content, including arbitrary shape video objects, special synthetic objects such as human face and body, and generic 2-D/3-D objects composed of primitives like rectangles, spheres, or indexed face sets, which define an object surface by means of vertices and surface patches. The synthetic visual objects are animated by transforms and special purpose animation techniques, such as face/body animation and 2D-mesh animation. MPEG-4 also provides synthetic audio tools such as structured audio tools and a text-to-speech interface (TTSI).

It is important to note that MPEG-4 only specifies the decoding of compliant bit streams in an MPEG-4 terminal. The encoders do enjoy a large degree of freedom in how to generate MPEG-4 compliant bit streams. Decoded audio-visual objects can be composed into 2D and 3D scenes using the Binary Format for Scenes (BIFS), which also allows implementation of animation of objects and their properties using the BIFS-Anim node.

This annex aims to provide all specific information regarding Face Animation. It is remarkable how Face Animation is intimately related to Body Animation. Most all the information gathered in this report is based or partially taken from ISO/IEC 14496-2 (MPEG-4 Visual) and «*Face and 2-D Mesh Animation in MPEG-4*» [Murat Tekalp and Jörn Ostermann]

Main goals of the standard

Specifications for Synthetic Video Objects

Types of Synthetic Video Objects

a) The object data shall include the following appearance attributes: font style, texture, color, transparency, surface characteristics.
b) Scene data shall include viewing characteristics as applicable notably lighting, and viewpoint.
c) It shall be possible to download objects or components so that some objects may be added, removed or modified.
d) Geometry:
Both: indexed face and line sets with defining list of shared vertices.
2D: rectangle, circle, line, polygon, Bezier curve, 2D mesh with implicit structure.
3D: box, cone, cylinder, sphere, 3D mesh.
a) Material properties:
Both: transparency, color, normal, texture mapping texture translation.
2D: filled or empty shape, border/line width, dotted border/line, , shadow properties.
b) Surface appearance:
Both: material, image texture, video texture.
c) Text:
Simple text, formatted text, font styles
International language including direction of composition
Justification of text, direction of streaming
d) Animated streams:
Dynamic state information (position and attitude, FBA, 2D mesh)
e) Mixing of 2D and 3D objects
f) Face and Body objects

Note

For exact definitions of some of the above items, refer to VRML node definitions.
Behavior is also a composition issue.

2D/3D Mesh Compression

Tools shall be provided for incremental and error resilient download of models. Rendering of models shall be possible with incomplete models

- Error resilience: it is possible to cope with packet loss without having to retransmit the entire model.
- Incremental download: useful rendering can be done already before the complete model has been downloaded

Examples

- 3D elevation grid
- Face and body objects in the form of 3D polygon meshes.
- A 2D Delaunay mesh that can be represented by vertex positions.
- 3D Models can be very large with respect to the network capacity, and therefore it is useful if they can be rendered even before completely downloaded.

Definition & Animation Parameter Compression

MPEG-4 shall provide syntax and compression for Face Animation Parameters (FAP) and Face Definition Parameters (FDP), as well as Body Animation Parameters (BAP) and Body Definition Parameters (BDP).
It shall be possible to compress FAP with 2 kbit/s.

Example

A baseline face in a decoder shall be capable of immediately receiving FAPs from the bitstream, to produce facial animation: expressions, speech, etc. without downloading a specific face. If FDPs are received, they can be used to transform a generic face into a particular face determined

by its shape and (optional) texture. Such tailoring of the bitstream for terminal capability must recognize the performance capabilities and limitations of the terminal in set-up.

Note

Specification of body animation is under investigation.

Texture Mapping

MPEG-4 shall support texture mapping on 2D/3D mesh.
Texture size shall be an integer power of 2 (16x16, 1024x1024, ...)

Note

Real time texture mapping capabilities are expected once the texture is loaded into texture memory. The mesh onto which texture is mapped can be regular or have an arbitrary shape.

Examples

Mapping of a face image on a face mesh, mapping of an aerial image on a grid mesh.

- Alpha blending of a still or moving texture onto a video object.

Text Overlay

- MPEG-4 shall provide capability for text overlay.
- MPEG-4 shall allow standalone text overlay, in the absence of natural audio and video.
- Text overlay can be independent of underlying A/V representation, as well as MPEG-4 shall provide capability to compose overlay text into layered spatial hierarchies that can be arranged and synchronized with spatial and temporal events in associated audio and video.
- MPEG-4 shall provide capability to use bitmapped text.
- MPEG-4 shall provide capability to animate text at slow or real-time rates for ready interpretation and comprehension, controllable by user and/or provider.
- MPEG-4 shall provide capabilities for spatial-temporal location and manipulation of text overlay.
- MPEG-4 shall support international character sets, and text composition.

Note

The MPEG-4 text overlay standard shall be designed to accommodate its easy incorporation into systems utilizing other existing and developing standards, e.g. MPEG-1 and MPEG-2.

Examples

- News program similar to “PointCast Network”
- Program guides for broadcast television
- User-selected electronic ticker tape information
- Low-bandwidth news delivery (broadcast, Internet, etc.)
- Real-time “insertion” of advertisements (e.g. product background information, event announcements, local phone numbers, etc.)
- Hyperlinked text in video

Image and Graphics Overlay

- MPEG-4 shall provide capability for image and graphics overlay.
- MPEG-4 shall allow standalone image and graphics overlay, in the absence of natural audio and video.
- MPEG-4 shall provide image and graphics overlay that can be independent of underlying A/V representation, as well as MPEG-4 shall provide capability to compose overlay images and graphics into layered spatial hierarchies that can be arranged and synchronized with spatial and temporal events in associated audio and video.

- MPEG-4 shall provide capability to use coded images and graphics based on existing standards.
- MPEG-4 shall provide capability to animate overlaid images and graphics at slow or real-time rates for ready interpretation and comprehension, controllable by user and/or provider.
- MPEG-4 shall provide capabilities for spatial-temporal location and manipulation of image and graphics overlays.

Notes

The MPEG-4 image and graphics overlay standard shall be designed to accommodate its easy incorporation into systems utilizing other existing and developing standards, e.g. MPEG-1 and MPEG-2.

Examples

- Low-bandwidth news delivery (broadcast, Internet, etc.)
- Special effects for advertising
- Real-time "insertion" of advertisements (e.g. local company logos on network programming, etc.)
- Hyperlinked images and graphics in video

View-Dependent Texture Scalability

MPEG-4 shall provide means to change non-uniformly the spatial resolution of the texture data by taking into account viewing conditions (viewpoint, aimpoint, lighting,...) and 3D mesh on which texture is to be mapped.
Forward channel bandwidth shall be up to 1 Mbit/s.

Note

Use of back channel may be required to transmit viewing conditions.

Example

An aerial view is mapped on a 3D grid mesh, the most visible regions of this texture are transmitted with a high quality, the least visible with the lowest quality (may even not be transmitted).

Geometrical transformations

MPEG-4 shall provide cost-effective means to cope with a large number of geometrical transformations without significant effect on the quality of the final rendered data. Geometric transformations shall support relative positioning, scaling, and orientation of objects in scene composition.
2D and 3D transformations are to be supported:
a) linear affine;
b) non-linear or perspective affine;
c) bi-linear transformations.

Video Object Tracking

MPEG-4 shall support efficient coding of mesh-based video object tracking information. This includes coding of mesh geometry (once for each video object or a temporal segment of a video object) and one motion vector for each node point at each frame.
Applicable to all video objects considered for coding. It shall be possible to code Video tracking as side information at 4-5 kbits/s.

Note

The inclusion of tracking information with a video object is optional.

Example

- Animated texture or graphics overlay on a moving natural or synthetic video object.
- Synthetic transfiguration and augmented reality.

Face Animation

The 'Face Animation' part of the standard allows sending parameters that calibrate and animate synthetic faces. These models themselves are not standardized by MPEG-4, only the parameters are.

- Definition and coding of face animation parameters (model independent):
  - Feature point positions and orientations to animate the face definition meshes
  - Visemes, or visual lip configurations equivalent to speech phonemes
- Definition and coding of face definition parameters (for model calibration):
  - 3-D feature point positions
  - 3-D head calibration meshes for animation
  - Texture map of face
  - Personal characteristics
- Facial texture coding;

MPEG-4 enables integration of face animation with multimedia communications and presentations and allows face animation over low bit rate communication channels, for point to point as well as multi-point connections with low-delay.

MPEG-4 AND FACE ANIMATION

The representation of synthetic visual objects in MPEG-4 is based on the prior VRML standard using nodes such as Transform, which defines rotation, scale or translation of an object, and IndexedFaceSet describing 3-D shape of an object by an indexed face set. However, MPEG-4 is the first international standard that specifies a compressed binary representation of animated synthetic audio-visual objects.

Specification and Animation of Faces

MPEG-4 specifies a face model in its neutral state, a number of feature points on this neutral face as reference points, and a set of FAPs, each corresponding to a particular facial action deforming a face model in its neutral state. The FAP value for a particular FAP indicates the magnitude of the corresponding action, e.g., a big versus a small smile or deformation of a mouth corner. For an MPEG-4 terminal to interpret the FAP values using its face model, it has to have predefined model specific animation rules to produce the facial action corresponding to each FAP. The terminal can either use its own animation rules or download a face model and the associated face animation tables (FAT) to have a customized animation behavior. Since the FAPs are required to animate faces of different sizes and proportions, the FAP values are defined in face animation parameter units (FAPU). The FAPU are computed from spatial distances between major facial features on the model in its neutral state.

DESCRIPTION OF THE FACE OBJECT

Neutral face and Facial Animation Parameter Units

As the first step, MPEG-4 defines a generic face model in its neutral state by the following properties (see Figure 1):

- the coordinate system is right-handed; head axes are parallel to the world axes
- gaze is in direction of Z axis,
- all face muscles are relaxed,
- eyelids are tangent to the iris, the pupil is one third of the diameter of the iris,
- lips are in contact; the line of the lips is horizontal and at the same height of lip corners,
- the mouth is closed and the upper teeth touch the lower ones,
- the tongue is flat, horizontal with the tip of tongue touching the boundary between upper and lower teeth.

A FAPU and the feature points used to derive the FAPU are defined next with respect to the face in its neutral state.

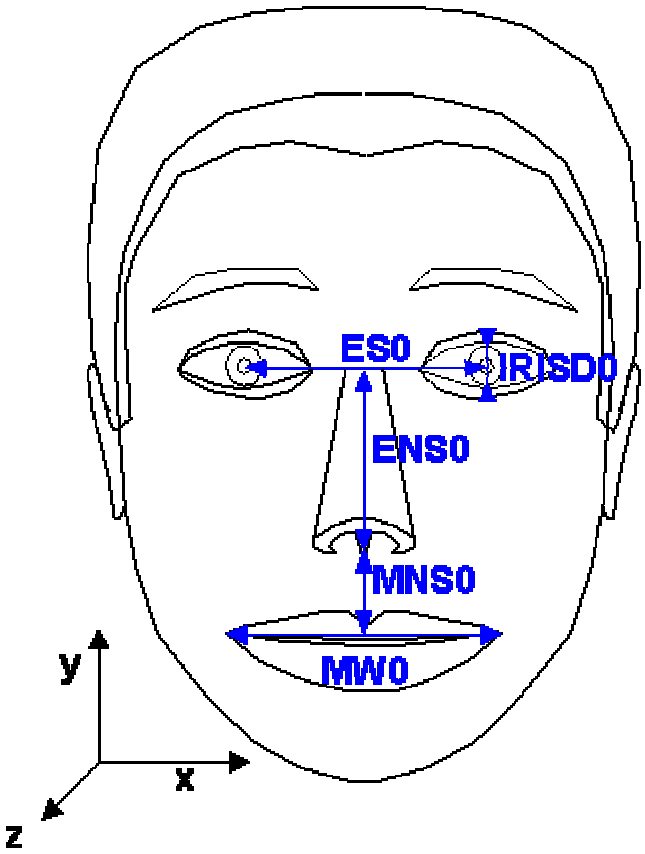


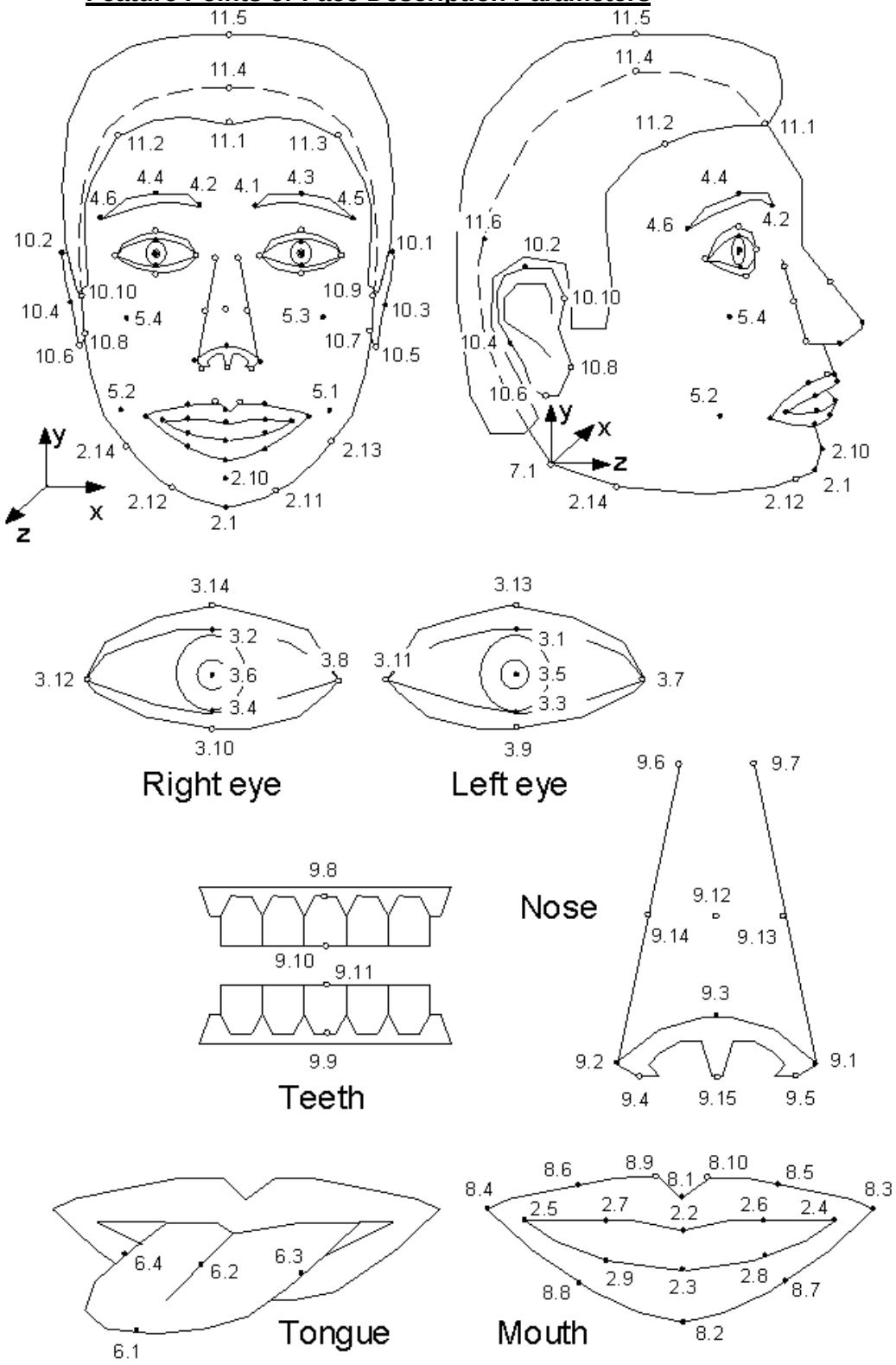
Figure 1: A face model in its neutral state and the feature points used to define FAP units (FAPU). Fractions of distances between the marked key features are used to define FAPU

The FAPU allow interpretation of the FAPs on any facial model in a consistent way, producing reasonable results in terms of expression and speech pronunciation. The measurement units are shown in Table 1.

Table 1: Facial Animation Parameter Units and their definitions.

Description		FAPU Value
IRISD0 = 3.1.y – 3.3.y = 3.2.y – 3.4.y	Iris diameter (by definition it is equal to the distance between upper ad lower eyelid) in neutral face	IRISD = IRISD0 / 1024
ES0 = 3.5.x – 3.6.x	Eye separation	ES = ES0 / 1024
ENS0 = 3.5.y – 9.15.y	Eye - nose separation	ENS = ENS0 / 1024
MNS0 = 9.15.y – 2.2.y	Mouth - nose separation	MNS = MNS0 / 1024
MW0 = 8.3.x – 8.4.x	Mouth width	MW = MW0 / 1024
AU	Angle Unit	10^{-5} rad

Feature Points or Face Description Parameters



• Feature points affected by FAPs

▫ Other feature points

MPEG-4 specifies 84 feature points on the neutral face. Feature points are arranged in groups like cheeks, eyes, and mouth. The location of these feature points has to be known for any MPEG-4 compliant face model. The feature points on the model should be located according to the Figure of the previous page.

The FDPs are normally transmitted once per session, followed by a stream of compressed FAPs. If the decoder does not receive the FDPs, the use of FAPUs ensures that it can still interpret the FAP stream. This insures minimal operation in broadcast or teleconferencing applications. The FDP set is specified in BIFS syntax (see ISO/IEC 14496-1). The FDP node defines the face model to be used at the receiver. Two options are supported:

- calibration information is downloaded so that the proprietary face of the receiver can be configured using facial feature points and optionally a 3D mesh or texture.
- a face model is downloaded with the animation definition of the Facial Animation Parameters. This face model replace the proprietary face model in the receiver.

In the following, the notation 2.1.x indicates the x coordinate of feature point 2.1.

Feature points		**Recommended location constraints**		
#	Text description	x	y	z
2.1	Bottom of the chin	7.1.x		
2.2	Middle point of inner upper lip contour	7.1.x		
2.3	Middle point of inner lower lip contour	7.1.x		
2.4	Left corner of inner lip contour			
2.5	Right corner of inner lip contour			
2.6	Midpoint between f.p. 2.2 and 2.4 in the inner upper lip contour	(2.2.x+2.4.x)/2		
2.7	Midpoint between f.p. 2.2 and 2.5 in the inner upper lip contour	(2.2.x+2.5.x)/2		
2.8	Midpoint between f.p. 2.3 and 2.4 in the inner lower lip contour	(2.3.x+2.4.x)/2		
2.9	Midpoint between f.p. 2.3 and 2.5 in the inner lower lip contour	(2.3.x+2.5.x)/2		
2.10	Chin boss	7.1.x		
2.11	Chin left corner	> 8.7.x and < 8.3.x		
2.12	Chin right corner	> 8.4.x and < 8.8.x		
2.13	Left corner of jaw bone			
2.14	Right corner of jaw bone			
3.1	Center of upper inner left eyelid	(3.7.x+3.11.x)/2		

3.2	Center of upper inner right eyelid	(3.8.x+3.12.x)/2		
3.3	Center of lower inner left eyelid	(3.7.x+3.11.x)/2		
3.4	Center of lower inner right eyelid	(3.8.x+3.12.x)/2		
3.5	Center of the pupil of left eye			
3.6	Center of the pupil of right eye			
3.7	Left corner of left eye			
3.8	Left corner of right eye			
3.9	Center of lower outer left eyelid	(3.7.x+3.11.x)/2		
3.10	Center of lower outer right eyelid	(3.7.x+3.11.x)/2		
3.11	Right corner of left eye			
3.12	Right corner of right eye			
3.13	Center of upper outer left eyelid	(3.8.x+3.12.x)/2		
3.14	Center of upper outer right eyelid	(3.8.x+3.12.x)/2		
4.1	Right corner of left eyebrow			
4.2	Left corner of right eyebrow			
4.3	Uppermost point of the left eyebrow	(4.1.x+4.5.x)/2 or x coord of the uppermost point of the contour		
4.4	Uppermost point of the right eyebrow	(4.2.x+4.6.x)/2 or x coord of the uppermost point of the contour		
4.5	Left corner of left eyebrow			
4.6	Right corner of right eyebrow			
5.1	Center of the left cheek		8.3.y	
5.2	Center of the right cheek		8.4.y	
5.3	Left cheek bone	> 3.5.x and < 3.7.x	> 9.15.y and < 9.12.y	
5.4	Right cheek bone	> 3.6.x and < 3.12.x	> 9.15.y and < 9.12.y	
6.1	Tip of the tongue	7.1.x		
6.2	Center of the tongue body	7.1.x		
6.3	Left border of the tongue			6.2.z
6.4	Right border of the tongue			6.2.z
7.1	top of spine (center of head rotation)			

8.1	Middle point of outer upper lip contour	7.1.x		
8.2	Middle point of outer lower lip contour	7.1.x		
8.3	Left corner of outer lip contour			
8.4	Right corner of outer lip contour			
8.5	Midpoint between f.p. 8.3 and 8.1 in outer upper lip contour	(8.3.x+8.1.x)/2		
8.6	Midpoint between f.p. 8.4 and 8.1 in outer upper lip contour	(8.4.x+8.1.x)/2		
8.7	Midpoint between f.p. 8.3 and 8.2 in outer lower lip contour	(8.3.x+8.2.x)/2		
8.8	Midpoint between f.p. 8.4 and 8.2 in outer lower lip contour	(8.4.x+8.2.x)/2		
8.9	Right hiph point of Cupid's bow			
8.10	Left hiph point of Cupid's bow			
9.1	Left nostril border			
9.2	Right nostril border			
9.3	Nose tip	7.1.x		
9.4	Bottom right edge of nose			
9.5	Bottom left edge of nose			
9.6	Right upper edge of nose bone			
9.7	Left upper edge of nose bone			
9.8	Top of the upper teeth	7.1.x		
9.9	Bottom of the lower teeth	7.1.x		
9.10	Bottom of the upper teeth	7.1.x		
9.11	Top of the lower teeth	7.1.x		
9.12	Middle lower edge of nose bone (or nose bump)	7.1.x	(9.6.y + 9.3.y)/2 or nose bump	
9.13	Left lower edge of nose bone		(9.6.y +9.3.y)/2	
9.14	Right lower edge of nose bone		(9.6.y +9.3.y)/2	
9.15	Bottom middle edge of nose	7.1.x		
10.1	Top of left ear			
10.2	Top of right ear			
10.3	Back of left ear		(10.1.y+10.5.y)/ 2	
10.4	Back of right ear		(10.2.y+10.6.y)/ 2	

10.5	Bottom of left ear lobe			
10.6	Bottom of right ear lobe			
10.7	Lower contact point between left lobe and face			
10.8	Lower contact point between right lobe and face			
10.9	Upper contact point between left ear and face			
10.10	Upper contact point between right ear and face			
11.1	Middle border between hair and forehead	7.1.x		
11.2	Right border between hair and forehead	< 4.4.x		
11.3	Left border between hair and forehead	> 4.3.x		
11.4	Top of skull	7.1.x		> 10.4.z and < 10.2.z
11.5	Hair thickness over f.p. 11.4	11.4.x		11.4.z
11.6	Back of skull	7.1.x	3.5.y	

FDP field	Description
FeaturePointsCoord	contains a **Coordinate** node. Specifies feature points for the calibration of the proprietary face. The coordinates are listed in the 'point' field in the **Coordinate** node in the prescribed order, that a feature point with a lower label is listed before a feature point with a higher label (e.g. feature point 3.14 before feature point 4.1).
TextureCoords	contains a **Coordinate** node. Specifies texture coordinates for the feature points. The coordinates are listed in the **point** field in the **Coordinate** node in the prescribed order, that a feature point with a lower label is listed before a feature point with a higher label (e.g. feature point 3.14 before feature point 4.1).
TextureType	contains a hint to the decoder on the type of texture image, in order to allow better interpolation of texture coordinates for the vertices that are not feature points. If **textureType** is 0, the decoder should assume that the texture image is obtained by cylindrical projection of the face. If **textureType** is 1, the decoder should assume that the texture image is obtained by orthographic projection of the face.
FaceDefTables	contains **faceDefTables** nodes. The behavior of FAPs is defined in this field for the face in **faceSceneGraph**.
FaceSceneGraph	contains a **Group** node. In case of option 1, this can be used to contain a texture image as explained above. In case of option 2, this is the grouping node for face model rendered in the compositor and has to contain the face model. In this case, the

	effect of Facial Animation Parameters is defined in the **faceDefTablesfield**.

Face Animation Parameters

The FAPs are based on the study of minimal perceptible actions and are closely related to muscle actions. The 68 parameters are categorized into 10 groups related to parts of the face. FAPs can also be used to define facial action units. Exaggerated amplitudes permit the definition of actions that are normally not possible for humans, but are desirable for cartoon-like characters.

The FAP set contains two high level parameters visemes and expressions. A viseme is a visual correlate to a phoneme. The viseme parameter allows viseme rendering (without having to express them in terms of other parameters) and enhances the result of other parameters, insuring the correct rendering of visemes. Only static visemes which are clearly distinguished are included in the standard set. Additional visemes may be added in future extensions of the standard. Similarly, the expression parameter allows definition of high level facial expressions. The facial expression parameter values are defined by textual descriptions. To facilitate facial animation, FAPs that can be used together to represent natural expression are grouped together in FAP groups, and can be indirectly addressed by using an expression parameter. The expression parameter allows for a very efficient means of animating faces.

FAP 1 (visemes) and FAP 2 (expressions) are high-level animation parameters. A face model designer creates them for each face model. Using FAP 1 and FAP 2 together with low-level FAPs 3-68 that affect the same areas as FAP 1 and 2, may result in unexpected visual representations of the face. Generally, the lower level FAPs have priority over deformations caused by FAP 1 or 2. When specifying an expression with FAP 2, the encoder may sent an init_face bit that deforms the neutral face of the model with the expression prior to superimposing FAPs 3-68. This deformation is applied with the neutral face constraints of mouth closure, eye opening, gaze direction and head orientation. Since the encoder does not know how FAP 1 and 2 are implemented, we recommend using only those low-level FAPs that will not interfere with FAP 1 and 2.

FAPs names may contain letters with the following meaning: l = left, r = right, t = top, b = bottom, i = inner, o = outer, m = middle. The sum of two corresponding top and bottom eyelid FAPs must equal 1024 when the eyelids are closed. Inner lips are closed when the sum of two corresponding top and bottom lip FAPs equals zero. For example: (lower_t_midlip + raise_b_midlip) = 0 when the lips are closed. All directions are defined with respect to the face and not the image of the face.

Table C-1 -- FAP definitions, group assignments and step sizes

#	FAP name	FAP description	units	Uni-orBidir	Pos motion	Grp	FDP subgrp num	Quant step size	Min/Max I-Frame quantized values	Min/Max P-Frame quantized values
1	Viseme	Set of values determining the mixture of two visemes for this frame (e.g. pbm, fv,	na	na	na	1	na	1	viseme_blend: +63	viseme_blend: +-63

		th)								
2	expression	A set of values determining the mixture of two facial expression	Na	na	na	1	na	1	expression_intensity1, expression_intensity2: +63	expression_intensity1, expression_intensity2: +-63
3	open_jaw	Vertical jaw displacement (does not affect mouth opening)	MNS	U	down	2	1	4	+1080	+360
4	lower_t_midlip	Vertical top middle inner lip displacement	MNS	B	down	2	2	2	+-600	+-180
5	raise_b_midlip	Vertical bottom middle inner lip displacement	MNS	B	up	2	3	2	+-1860	+-600
6	stretch_l_cornerlip	Horizontal displacement of left inner lip corner	MW	B	left	2	4	2	+-600	+-180
7	stretch_r_cornerlip	Horizontal displacement of right inner lip corner	MW	B	right	2	5	2	+-600	+-180
8	lower_t_lip_lm	Vertical displacement of midpoint between left corner and middle of top inner lip	MNS	B	down	2	6	2	+-600	+-180
9	lower_t_lip_rm	Vertical displacement of midpoint between right corner and middle of top inner lip	MNS	B	down	2	7	2	+-600	+-180
10	raise_b_lip_lm	Vertical displacement of midpoint between left corner and middle of bottom inner lip	MNS	B	up	2	8	2	+-1860	+-600
11	raise_b_lip_rm	Vertical displacement of midpoint between right corner and middle of bottom	MNS	B	up	2	9	2	+-1860	+-600

		inner lip								
12	raise_l_cornerlip	Vertical displacement of left inner lip corner	MNS	B	up	2	4	2	+-600	+-180
13	raise_r_cornerlip	Vertical displacement of right inner lip corner	MNS	B	up	2	5	2	+-600	+-180
14	thrust_jaw	Depth displacement of jaw	MNS	U	forward	2	1	1	+600	+180
15	shift_jaw	Side to side displacement of jaw	MW	B	right	2	1	1	+-1080	+-360
16	push_b_lip	Depth displacement of bottom middle lip	MNS	B	forward	2	3	1	+-1080	+-360
17	push_t_lip	Depth displacement of top middle lip	MNS	B	forward	2	2	1	+-1080	+-360
18	depress_chin	Upward and compressing movement of the chin (like in sadness)	MNS	B	up	2	10	1	+-420	+-180
19	close_t_l_eyelid	Vertical displacement of top left eyelid	IRISD	B	down	3	1	1	+-1080	+-600
20	close_t_r_eyelid	Vertical displacement of top right eyelid	IRISD	B	down	3	2	1	+-1080	+-600
21	close_b_l_eyelid	Vertical displacement of bottom left eyelid	IRISD	B	up	3	3	1	+-600	+-240
22	close_b_r_eyelid	Vertical displacement of bottom right eyelid	IRISD	B	up	3	4	1	+-600	+-240
23	yaw_l_eyeball	Horizontal orientation of left eyeball	AU	B	left	3	na	128	+-1200	+-420
24	yaw_r_eyeball	Horizontal orientation of right eyeball	AU	B	left	3	na	128	+-1200	+-420
25	pitch_l_eyeball	Vertical orientation of left eyeball	AU	B	down	3	na	128	+-900	+-300
26	pitch_r_eyeball	Vertical orientation	AU	B	down	3	na	128	+-900	+-300

		of right eyeball								
27	thrust_l_eyeball	Depth displacement of left eyeball	ES	B	forward	3	na	1	+-600	+-180
28	thrust_r_eyeball	Depth displacement of right eyeball	ES	B	forward	3	na	1	+-600	+-180
29	dilate_l_pupil	Dilation of left pupil	IRISD	B	growing	3	5	1	+-420	+-120
30	dilate_r_pupil	Dilation of right pupil	IRISD	B	growing	3	6	1	+-420	+-120
31	raise_l_i_eyebrow	Vertical displacement of left inner eyebrow	ENS	B	up	4	1	2	+-900	+-360
32	raise_r_i_eyebrow	Vertical displacement of right inner eyebrow	ENS	B	up	4	2	2	+-900	+-360
33	raise_l_m_eyebrow	Vertical displacement of left middle eyebrow	ENS	B	up	4	3	2	+-900	+-360
34	raise_r_m_eyebrow	Vertical displacement of right middle eyebrow	ENS	B	up	4	4	2	+-900	+-360
35	raise_l_o_eyebrow	Vertical displacement of left outer eyebrow	ENS	B	up	4	5	2	+-900	+-360
36	raise_r_o_eyebrow	Vertical displacement of right outer eyebrow	ENS	B	up	4	6	2	+-900	+-360
37	squeeze_l_eyebrow	Horizontal displacement of left eyebrow	ES	B	right	4	1	1	+-900	+-300
38	squeeze_r_eyebrow	Horizontal displacement of right eyebrow	ES	B	left	4	2	1	+-900	+-300
39	puff_l_cheek	Horizontal displacement of left cheeck	ES	B	left	5	1	2	+-900	+-300
40	puff_r_cheek	Horizontal displacement of right cheeck	ES	B	right	5	2	2	+-900	+-300
41	lift_l_cheek	Vertical displacement of left cheek	ENS	U	up	5	3	2	+-600	+-180

42	lift_r_cheek	Vertical displacement of right cheek	ENS	U	up	5	4	2	+-600	+-180
43	shift_tongue_tip	Horizontal displacement of tongue tip	MW	B	right	6	1	1	+-1080	+-420
44	raise_tongue_tip	Vertical displacement of tongue tip	MNS	B	up	6	1	1	+-1080	+-420
45	thrust_tongue_tip	Depth displacement of tongue tip	MW	B	forward	6	1	1	+-1080	+-420
46	raise_tongue	Vertical displacement of tongue	MNS	B	up	6	2	1	+-1080	+-420
47	tongue_roll	Rolling of the tongue into U shape	AU	U	concave upward	6	3, 4	512	+300	+60
48	head_pitch	Head pitch angle from top of spine	AU	B	down	7	na	170	+-1860	+-600
49	head_yaw	Head yaw angle from top of spine	AU	B	left	7	na	170	+-1860	+-600
50	head_roll	Head roll angle from top of spine	AU	B	right	7	na	170	+-1860	+-600
51	lower_t_midlip _o	Vertical top middle outer lip displacement	MNS	B	down	8	1	2	+-600	+-180
52	raise_b_midlip_o	Vertical bottom middle outer lip displacement	MNS	B	up	8	2	2	+-1860	+-600
53	stretch_l_cornerlip_o	Horizontal displacement of left outer lip corner	MW	B	left	8	3	2	+-600	+-180
54	stretch_r_cornerlip_o	Horizontal displacement of right outer lip corner	MW	B	right	8	4	2	+-600	+-180
55	lower_t_lip_lm _o	Vertical displacement of midpoint between left corner and middle of top outer lip	MNS	B	down	8	5	2	+-600	+-180
56	lower_t_lip_rm _o	Vertical displacement of midpoint between	MNS	B	down	8	6	2	+-600	+-180

		right corner and middle of top outer lip								
57	raise_b_lip_lm_o	Vertical displacement of midpoint between left corner and middle of bottom outer lip	MNS	B	up	8	7	2	+-1860	+-600
58	raise_b_lip_rm_o	Vertical displacement of midpoint between right corner and middle of bottom outer lip	MNS	B	up	8	8	2	+-1860	+-600
59	raise_l_cornerlip_o	Vertical displacement of left outer lip corner	MNS	B	up	8	3	2	+-600	+-180
60	raise_r_cornerlip _o	Vertical displacement of right outer lip corner	MNS	B	up	8	4	2	+-600	+-180
61	stretch_l_nose	Horizontal displacement of left side of nose	ENS	B	left	9	1	1	+-540	+-120
62	stretch_r_nose	Horizontal displacement of right side of nose	ENS	B	right	9	2	1	+-540	+-120
63	raise_nose	Vertical displacement of nose tip	ENS	B	up	9	3	1	+-680	+-180
64	bend_nose	Horizontal displacement of nose tip	ENS	B	right	9	3	1	+-900	+-180
65	raise_l_ear	Vertical displacement of left ear	ENS	B	up	10	1	1	+-900	+-240
66	raise_r_ear	Vertical displacement of right ear	ENS	B	up	10	2	1	+-900	+-240
67	pull_l_ear	Horizontal displacement of left ear	ENS	B	left	10	3	1	+-900	+-300
68	pull_r_ear	Horizontal displacement of right ear	ENS	B	right	10	4	1	+-900	+-300

Table C-2 -- FAP grouping

Group	Number of FAPs
1: visemes and expressions	2
2: jaw, chin, inner lowerlip, cornerlips, midlip	16
3: eyeballs, pupils, eyelids	12
4: eyebrow	8
5: cheeks	4
6: tongue	5
7: head rotation	3
8: outer lip positions	10
9: nose	4
10: ears	4

In the following, each facial expression is defined by a textual description and a pictorial example. (reference [10], page 114.) This reference was also used for the characteristics of the described expressions.

Table C-3 -- Values for expression_select

Expression_select	expression name	Textual description
0	na	Na
1	joy	The eyebrows are relaxed. The mouth is open and the mouth corners pulled back toward the ears.
2	sadness	The inner eyebrows are bent upward. The eyes are slightly closed. The mouth is relaxed.
3	anger	The inner eyebrows are pulled downward and together. The eyes are wide open. The lips are pressed against each other or opened to expose the teeth.
4	fear	The eyebrows are raised and pulled together. The inner eyebrows are bent upward. The eyes are tense and alert.
5	disgust	The eyebrows and eyelids are relaxed. The upper lip is raised and curled, often asymmetrically.
6	surprise	The eyebrows are raised. The upper eyelids are wide open, the lower relaxed. The jaw is opened.

Table C-5 -- Values for viseme_select

viseme_select	phonemes	example
0	none	na
1	p, b, m	put, bed, mill

2	f, v	far, voice
3	T,D	think, that
4	t, d	tip, doll
5	k, g	call, gas
6	tS, dZ, S	chair, join, she
7	s, z	sir, zeal
8	n, l	lot, not
9	r	red
10	A:	car
11	e	bed
12	I	tip
13	Q	top
14	U	book

Face Model Specification versus a proprietary Model

Every MPEG-4 terminal that is able to decode FAP streams has to provide an MPEG-4 compliant face model that it animates. Usually, this is a model proprietary to the decoder. The encoder does not know about the look of the face model. Using an FDP (Face Definition Parameter) node, MPEG-4 allows the encoder to completely specify the face model to animate. This involves defining the static geometry of the face model in its neutral state using a scene graph, defining the surface properties and defining the animation rules using Face Animation Tables (FAT) that specify how this model gets deformed by the facial animation parameters. Alternatively, the FDP node can be used to 'calibrate' the proprietary face model of the decoder. However, MPEG-4 does not specify how to 'calibrate' or adapt a proprietary face model.

In order for a face model to be MPEG-4 compliant, it has to be able to execute all FAPs. Therefore, the face model has to have at least as many vertices as there are feature points that can be animated. Thus, an MPEG-4 compliant face model may have as little as 50 vertices. Such a model would not generate a pleasing impression. MPEG-4 expects to require at least 500 vertices for pleasant and reasonable face models.

Structure of the coded data

Upon construction, the Face object contains a generic face with a neutral expression. This face can already be rendered. It is also immediately capable of receiving the FAPs. If FDPs are received, they transform the generic face into a particular face determined by its shape and texture. Optionally, a complete face model can be downloaded via the FDP set as a scene graph

for insertion in the face node. The translation from phonemes to FAPs is not standardized. It is assumed that every decoder has a default face model with default parameters. The setup stage is used to customize the face at the decoder.

A face object is formed by a temporal sequence of face object planes. This is depicted as follows in Figure 6-9.

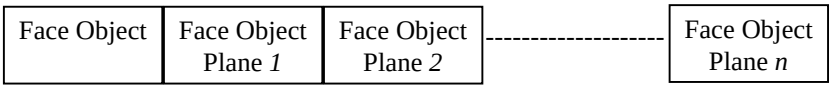
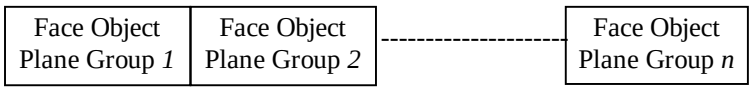


Figure 6-9 -- Structure of the face object bitstream

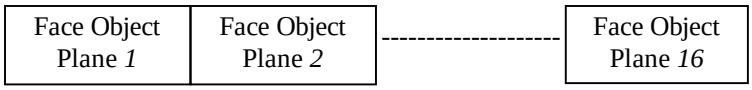
A face object represents a node in an ISO/IEC 14496 scene graph. The scene graph is the hierarchical representation of the ISO/IEC 14496 scene structure (see ISO/IEC 14496-1).

Alternatively, a face object can be formed by a temporal sequence of face object plane groups (called segments for simplicity), where each face object plane group itself is composed of a temporal sequence of 16 face object planes, as depicted in the following:

face object:



face object plane group:



When the alternative face object bitstream structure is employed, the bitstream is decoded by DCT-based face object decoding as described in *DCT Based Object Decoding* part. Otherwise, the bitstream is decoded by the frame-based face object decoding

FACE OBJECT DECODING AND OUTPUT

Frame based face object decoding

The coded data is decoded by an arithmetic decoding process, described in annex B of ISO/IEC 14496-2. Following the arithmetic decoding, the data is de-quantized by an inverse quantization process. The FAPs are obtained by a predictive decoding scheme as shown in Figure 7-47.

The base quantization step size QP for each FAP is listed in Table C-1. The quantization parameter fap_quant is applied uniformly to all FAPs. The magnitude of the quantization scaling factor ranges from 1 to 8. The value of fap_quant == 0 has a special meaning, it is used to indicate lossless coding mode, so no dequantization is applied. The quantization stepsize is obtained as follows:

```
if (fap_quant)
    qstep = QP * fap_quant
else
    qstep = 1
```

The dequantized FAP'(t) is obtained from the decoded coefficient FAP''(t) as follows:
FAP'(t) = qstep * FAP''(t)

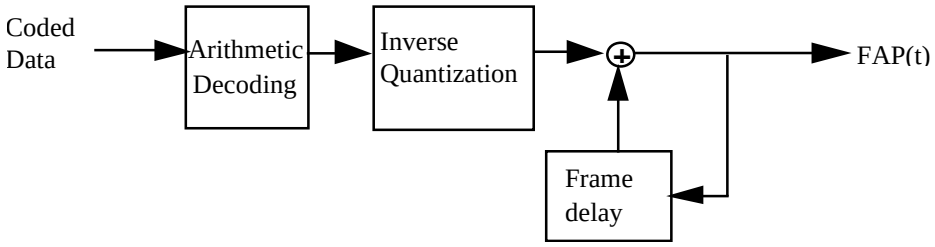


Figure 7-47 -- FAP decoding

Decoding of faps

For a given frame FAPs in the decoder assume one of three of the following states:
1. set by a value transmitted by the encoder
2. retain a value previously sent by the encoder
3. interpolated by the decoder

FAP values which have been initialized in an intra coded FAP set are assumed to retain those values if subsequently masked out unless a special mask mode is used to indicate interpolation by the decoder. FAP values which have never been initialized must be estimated by the decoder. For example, if only FAP group 2 (inner lip) is used and FAP group 8 (outer lip) is never used, the outer lip points must be estimated by the decoder. In a second example the FAP decoder is also expected to enforce symmetry when only the left or right portion of a symmetric FAP set is received (e.g. if the left eye is moved and the right eye is subject to interpolation, it is to be moved in the same way as the left eye).

DCT based face object decoding

The bitstream is decoded into segments of FAPs, where each segment is composed of a temporal sequence of 16 FAP object planes. The block diagram of the decoder is shown in Figure 7-48.

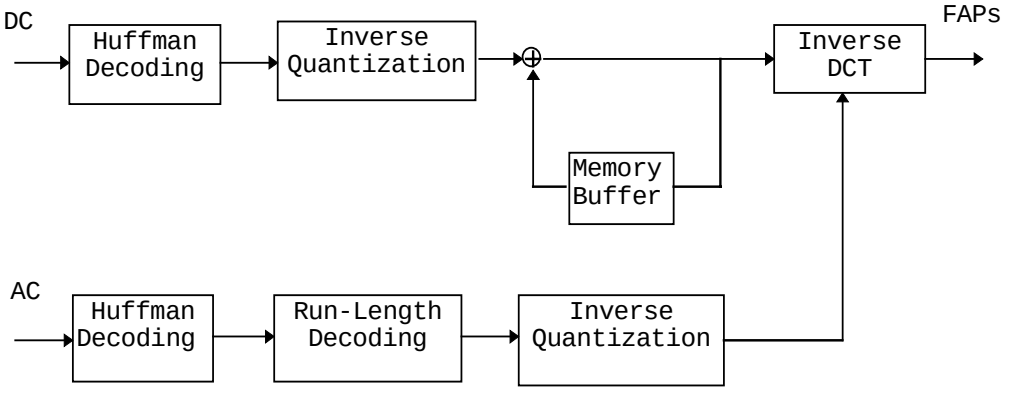


Figure 7-48 -- Block diagram of the DCT-based decoding process

The DCT-based decoding process consists of the following three basic steps:
1. Differential decoding the DC coefficient of a segment.
2. Decoding the AC coefficients of the segment
3. Determining the 16 FAP values of the segment using inverse discrete cosine transform (IDCT).

A uniform quantization step size is used for all AC coefficients. The quantization step size for AC coefficients is obtained as follows:

qstep[i] = fap_scale[fap_quant_inex] * DCTQP[i]

where DCTQP[i] is the base quantization step size and its value is defined in a subclause of the ISO/IEC 14496-2 (Maui version). The quantization step size of the DC coefficient is one-third of the AC coefficients. Different quantization step sizes are used for different FAPs.

The DCT-based decoding process is applied to all FAP segments except the viseme (FAP #1) and expression (FAP #2) parameters. The latter two parameters are differential decoded without transform. The decoding of viseme and expression segments are described at the end of this subclause.

For FAP #3 to FAP #68, the DC coefficient of an intra coded segment is stored as a 16-bit signed integer if its value is within the 16-bit range. Otherwise, it is stored as a 31-bit signed integer. For an inter coded segment, the DC coefficient of the previous segment is used as a prediction of the current DC coefficient. The prediction error is decoded using a Huffman table of 512 symbols. . An "ESC" symbol, if obtained, indicates that the prediction error is out of the range [-255, 255]. In this case, the next 16 bits extracted from the bitstream are represented as a signed 16-bit integer for the prediction error. If the value of the integer is equal to -256*128, it means that the value of the prediction error is over the 16-bit range. Then the following 32 bits from the bitstream are extracted as a signed 32-bit integer, in twos complement format and the most significant bit first

The AC coefficients, for both inter and intra coded segments, are decoded using Huffman tables. The run-length code indicates the number of leading zeros before each non-zero AC coefficient. The run-length ranges from 0 to 14 and proceeds the code for the AC coefficient. The symbol 15 in the run length table indicates the end of non-zero symbols in a segment. Therefore, the Huffman table of the run-length codes contains 16 symbols. The values of non-zero AC coefficients are decoded in a way similar to the decoding of DC prediction errors but with a different Huffman table.

The bitstreams corresponding to viseme and expression segments are basically differential decoded without IDCT. For an intra coded segment, the quantized values of the first viseme_select1, viseme_select2, viseme_blend, expression_select1, expression_select2, expression_intensity1, and expression_intensity2 within the segment are decoded using fixed length code. These first values are used as the prediction for the second viseme_select1, viseme_select2, ... etc of the segment and the prediction error are differential decoded using Huffman tables. For an inter coded segment, the last viseme_select1, for example, of the previous decoded segment is used to predict the first viseme_select1 of the current segment. In general, the decoded values (before inverse quantization) of differential coded viseme and expression parameter fields are obtained

byviseme_segment_select1q[k] = viseme_segment_select1q[k-1] +
viseme_segment_select1q_diff[k] - 14
viseme_segment_select2q[k] = viseme_segment_select2q[k-1] +
viseme_segment_select2q_diff[k] - 14
viseme_segment_blendq[k] = viseme_segment_blendq[k-1] +
viseme_segment_blendq_diff[k] - 63
expression_segment_select1q[k] = expression_segment_select1q[k-1] +
expression_segment_select1q_diff[k] - 6
expression_segment_select2q[k] = expression_segment_select2q[k-1] +
expression_segment_select2q_diff[k] - 6
expression_segment_intensity1q[k] = expression_segment_intensity1q[k-1] +
expression_segment_intensity1q_diff[k] - 63
expression_segment_intensity2q[k] = expression_segment_intensity2q[k-1] +
expression_segment_intensity2q_diff[k] - 63

Decoding of the viseme parameter fap 1

Fourteen visemes have been defined for selection by the Viseme Parameter FAP 1. The viseme parameter allows two visemes from a standard set to be blended together. The viseme parameter is composed of a set of values as follows.

Table 7-17 -- Viseme parameter range

viseme () {	**Range**
viseme_select1	0-14
viseme_select2	0-14
viseme_blend	0-63
viseme_def	0-1
}	

Viseme_blend is quantized (step size = 1) and defines the blending of viseme1 and viseme2 in the decoder by the following symbolic expression where viseme1 and 2 are graphical interpretations of the given visemes as suggested in the non-normative annex.

final viseme = (viseme 1) * (viseme_blend / 63) + (viseme 2) * (1 - viseme_blend / 63)

The viseme can only have impact on FAPs that are currently allowed to be interpolated.
If the viseme_def bit is set, the current mouth FAPs can be used by the decoder to define the selected viseme in terms of a table of FAPs. This FAP table can be used when the same viseme is invoked again later for FAPs which must be interpolated.

Decoding of the viseme parameter fap 2

The expression parameter allows two expressions from a standard set to be blended together. The expression parameter is composed of a set of values as follows.

Table 7-18 -- Expression parameter range

expression () {	**Range**
expression_select1	0-6

expression_intensity1	0-63
expression_select2	0-6
expression_intensity2	0-63
init_face	0-1
expression_def	0-1
}	

Expression_intensity1 and expression_intensity2 are quantized (step size = 1) and define excitation of expressions 1 and 2 in the decoder by the following equations where expressions 1 and 2 are graphical interpretations of the given expression as suggested by the non-normative reference:

final expression = expression1 * (expression_intensity1 / 63)+ expression2 * (expression_intensity2 / 63)

The decoder displays the expressions according to the above fomula as a superposition of the 2 expressions.

The expression can only have impact on FAPs that are currently allowed to be interpolated. If the init_face bit is set, the neutral face may be modified within the neutral face constraints of mouth closure, eye opening, gaze direction, and head orientation before FAPs 3-68 are applied. If the expression_def bit is set, the current FAPs can be used to define the selected expression in terms of a table of FAPs. This FAP table can then be used when the same expression is invoked again later.

Fap masking

The face is animated by sending a stream of facial animation parameters. FAP masking, as indicated in the bitstream, is used to select FAPs. FAPs are selected by using a two level mask hierarchy. The first level contains two bit code for each group indicating the following options:

1. no FAPs are sent in the group.
2. a mask is sent indicating which FAPs in the group are sent. FAPs not selected by the group mask retain their previous value if any previously set value (not interpolated by decoder if previously set)
3. a mask is sent indicating which FAPs in the group are sent. FAPs not selected by the group mask retain must be interpolated by the decoder.
4. all FAPs in the group are sent.

Output of the decoding process for SNHC

Here we describe the output of the theoretical model of the decoding process that decodes bitstreams conforming to the Visual part of ISO/IEC 14496.

The visual decoding process input is one or more coded visual bitstreams (one for each of the layers). The visual layers are generally multiplexed by the means of a system stream that also contains timing information.

Video data

The output of the video decoding process is a series of VOPs that are normally the input of a display process. The order in which fields or VOPs are output by the decoding process is called the display order, and may be different from the coded order (when B-VOPs are used).

2D Mesh data
The output of the decoding process is a series of one or more mesh object planes. The mesh object planes are normally input to a compositor that maps the texture of a related video object or still texture object onto each mesh. The coded order and the composited order of the mesh object planes are identical.

Face animation parameter data
The output of the decoding process is a sequence of facial animation parameters. They are input to a display process that uses the parameters to animate a face object.

INTEGRATION INTO MPEG-4 SYSTEMS

In order to use face animation in the context of MPEG-4 systems, a BIFS scene graph has to be transmitted to the decoder. The minimum scene graph contains a Face node and a FAP node. The FAP decoder writes the amplitude of the FAPs into fields of the FAP node. The FAP node might have the children Viseme and Expression which are FAPs requiring a special syntax. This scene graph would enable an encoder to animate the proprietary face model of the decoder. If a face model is to be controlled from a TTS system, an AudioSource node is to be attached to the face node.

In order to download a face model to the decoder, the face node requires an FDP node as one of its children. This FDP node contains the position of the feature points in the downloaded model, the scene graph of the model and the FaceDefTable, FaceDefMesh and FaceDefTransform nodes required to define the action caused by FAPs.

BIFS Syntax for Face Animation

Node interface

Face {

exposedField	SFNode	**fit**	NULL
exposedField	SFNode	**fdp**	NULL
exposedField	SFNode	**fap**	NULL
exposedField	SFNode	**ttsSource**	NULL
exposedField	MFNode	**renderedFace**	NULL

}

NOTE — For the binary encoding of this node see Document MPEG-4 NODES A.1.36.

Functionality and semantics

The **Face** node is used to define and animate a face in the scene. In order to animate the face with a facial animation stream, it is necessary to link the **Face** node to a BIFS-Anim stream. The node shall be assigned a `nodeID`, through the DEF mechanism. Then, as for any BIFS-Anim stream, an animation mask is sent in the object descriptor of the BIFS-Anim stream (`specificInfo` field). The animation mask points to the

Face node using its `nodeID`. The terminal shall then connect the facial animation decoder to the appropriate **Face** node.

The **FAP** field shall contain a **FAP** node, describing the facial animation parameters (FAPs). Each **Face** node shall contain a non-NULL **FAP** field.

The **FDP** field, which defines the particular look of a face by means of downloading the position of face definition points or an entire model, is optional. If the **FDP** field is not specified, the default face model of the terminal shall be used.

The **FIT** field, when specified, allows a set of FAPs to be defined in terms of another set of FAPs. When this field is non-NULL, the terminal shall use **FIT** to compute the maximal set of FAPs before using the FAPs to compute the mesh.

The **ttsSource** field shall only be non-NULL if the facial animation is to determine the facial animation parameters from an audio TTS source (see ISO/IEC 14496-3, section 6). In this case the **ttsSource** field shall contain an **AudioSource** node and the face shall be animated using the phonemes and bookmarks received from the TTS.

renderedFace is the scene graph of the face after it is rendered (all FAP's applied).

FaceDefMesh

Node interface

FaceDefMesh {			
field	SFNode	**faceSceneGraphNode**	NULL
field	MFInt32	**intervalBorders**	[]
field	MFInt32	**coordIndex**	[]
field	MFVec3f	**displacements**	[]
}			

NOTE — For the binary encoding of this node see Document MPEG-4 NODES A.1.37.

Functionality and semantics

The **FaceDefMesh** node allows for the deformation of an **IndexedFaceSet** as a function of the amplitude of a FAP as specified in the related **FaceDefTable** node. The **FaceDefMesh** node defines the piece-wise linear motion trajectories for vertices of the **faceSceneGraphNode** field, which shall contain an **IndexedFaceSet** node. This **IndexedFaceSet** node belongs to the scenegraph of the **faceSceneGraph** field of the **FDP** node.

The **intervalBorders** field specifies interval borders for the piece-wise linear approximation in increasing order. Exactly one interval border shall have the value 0.

The **coordIndex** field shall contain a list of indices into the **Coordinate** node of the **IndexedFaceSet** node specified by the **faceSceneGraphNode** field.

For each vertex indexed in the **coordIndex** field, displacement vectors are given in the **displacements** field for the intervals defined in the **intervalBorders** field. There must be exactly (num(**intervalBorders**)-1)*num(**coordIndex**) values in this field.

In most cases, the animation generated by a FAP cannot be specified by updating a **Transform** node. Thus, a deformation of an **IndexedFaceSet** node needs to be performed. In this case, the **FaceDefTables** shall define which **IndexedFaceSets** are affected by a given FAP and how the **coord** fields of these nodes are updated. This is done by means of tables.

If a FAP affects an **IndexedFaceSet**, the **FaceDefMesh** shall specify a table of the following format for this **IndexedFaceSet**:

Table 7 - Vertex displacements

Vertex no.	**1st Interval [I1, I2]**	**2nd Interval [I2, I3]**	**…**
Index 1	Displacement D11	Displacement D12	…
Index 2	Displacement D21	Displacement D22	…
…	…	…	…

Exactly one interval border I_k must have the value 0:

$$[I_1, I_2], [I_2, I_3], \ldots[I_{k-1}, 0], [0, I_{k+1}], [I_{k+1}, I_{k+2}], \ldots[I_{max-1}, I_{max}]$$

During animation, when the terminal receives a FAP, which affects one or more **IndexedFaceSets** of the face model, it shall piece-wise linearly approximate the motion trajectory of each vertex of the affected **IndexedFaceSets** by using the appropriate table.

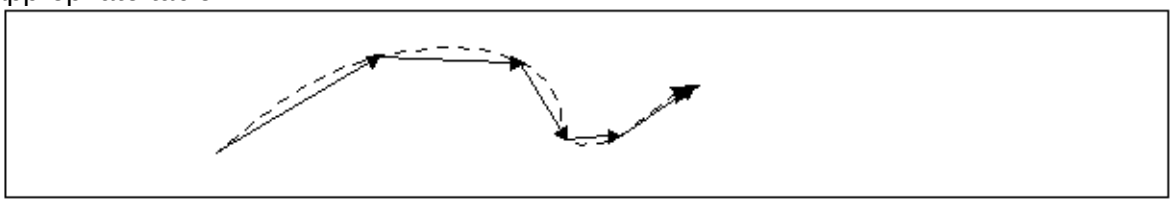


Figure 4 - An arbitrary motion trajectory is approximated as a piece-wise linear one.

If $\boldsymbol{P}_m$ is the position of the m^{th} vertex in the **IndexedFaceSet** in neutral state (FAP = 0), $\boldsymbol{P'}_m$ the position of the same vertex after animation with the given FAP and $\boldsymbol{D_{mk}}$ the 3D displacement in the k^{th} interval, the following algorithm shall be applied to determine the new position $\boldsymbol{P'}_m$.
Determine, in which of the intervals listed in the table the received FAP is lying.

If the received FAP is lying in the j^{th} interval [Ij, Ij+1] and 0=Ik ≤ Ij, the new vertex position P'm of the mth vertex of the **IndexedFaceSet** is given by:

$$P'_m = FAPU * ((I_{k+1}-0) * D_{m,k} + (I_{k+2}-I_{k+1}) * D_{m,k+1} + \ldots + (I_j - I_{j-1}) * D_{m,j-1} + (FAP-I_j) * D_{m,j}) + P_m. \quad \text{(Eq. 1)}$$

If FAP > I_{max}, then P'_m is calculated by using equation Eq. 1 and setting the index j = max.
If the received FAP is lying in the jth interval [I_j, I_{j+1}] and $I_{j+1} \leq I_k$=0, the new vertex position P'_m is given by:

$P'_m = FAPU * ((I_{j+1} - FAP) * D_{m,j} + (I_{j+2} - I_{j+1}) * D_{m,j+1} + ... + (I_{k-1} - I_{k-2}) * D_{m,k-2} + (0 - I_{k-1}) * D_{m,k-1}) + P_m$ (Eq. 2)

If $FAP < I_1$, then P'_m is calculated by using equation Eq. 1 and setting the index $j+1 = 1$. If for a given FAP and **IndexedFaceSet** the table contains only one interval, the motion is strictly linear:
P'm = FAPU * FAP * Dm1 + Pm.

EXAMPLE —

```
FaceDefMesh {
   objectDescriptorID UpperLip
   intervalBorders [ -1000, 0, 500, 1000 ]
   coordIndex [ 50, 51]
   displacements [1 0 0,  0.9 0 0, 1.5 0 4, 0.8 0 0, 0.7 0 0, 2 0 0 ]
}
```

This **FaceDefMesh** defines the animation of the mesh "UpperLip". For the piecewise-linear motion function three intervals are defined: [-1000, 0], [0, 500] and [500, 1000]. Displacements are given for the vertices with the indices 50 and 51. The displacements for the vertex 50 are: (1 0 0), (0.9 0 0) and (1.5 0 4), the displacements for vertex 51 are (0.8 0 0), (0.7 0 0) and (2 0 0). Given a FAPValue of 600, the resulting displacement for vertex 50 would be:

displacement(vertex 50) = $500*(0.9\ 0\ 0)^T + 100 * (1.5\ 0\ 4)^T = (600\ 0\ 400)^T$.

If the FAPValue is outside the given intervals, the boundary intervals are extended to +I or -I, as appropriate.

FaceDefTables

Node interface

FaceDefTables {

field	SFInt32	**fapID**	0
field	SFInt32	**highLevelSelect**	0
exposedField	MFNode	**faceDefMesh**	[]
exposedField	MFNode	**faceDefTransform**	[]

}

NOTE — For the binary encoding of this node see Document MPEG-4 NODES A.1.38.

Functionality and semantics

The **FaceDefTables** node defines the behavior of a facial animation parameter FAP on a downloaded face model in **faceSceneGraph** by specifying the displacement vectors for moved vertices inside **IndexedFaceSet** objects as a function of the FAP **fapID** and/or specifying the value of a field of a **Transform** node as a function of FAP **fapID**.

The **FaceDefTables** node is transmitted directly after the BIFS bitstream of the **FDP** node. The **FaceDefTables** lists all FAPs that animate the face model. The FAPs animate the downloaded face model by updating the **Transform** or **IndexedFaceSet** nodes of the scene graph in **faceSceneGraph**. For each listed FAP, the **FaceDefTables** node describes which nodes are animated by this FAP and how they are animated. All FAPs that occur in the bitstream have to be specified in the

FaceDefTables node. The animation generated by a FAP can be specified either by updating a **Transform** node (using a **FaceDefTransform**), or as a deformation of an **IndexedFaceSet** (using a **FaceDefMesh**).

The FAPUs shall be calculated by the terminal using the feature points that shall be specified in the FDP. The FAPUs are needed in order to animate the downloaded face model.

Semantics

The **fapID** field specifies the FAP, for which the animation behavior is defined in the **faceDefMesh** and **faceDefTransform** fields.

If **fapID** has value 1 or 2, the **highLevelSelect** field specifies the type of viseme or expression. In other cases this field has no meaning and shall be ignored.

The **faceDefMesh** field shall contain a **FaceDefMesh** node.

The **faceDefTransform** field shall contain a **FaceDefTransform** node.

FaceDefTransform

Node interface

FaceDefTransform {

field	SFNode	**faceSceneGraphNode**	NULL
field	SFInt32	**fieldId**	1
field	SFRotation	**rotationDef**	0, 0, 1, 0
field	SFVec3f	**scaleDef**	1, 1, 1
field	SFVec3f	**translationDef**	0, 0, 0

}

NOTE — For the binary encoding of this node see Document MPEG-4 NODES A.1.39.

Functionality and semantics

The **FaceDefTransform** node defines which field (**rotation**, **scale** or **translation**) of a **Transform** node (**faceSceneGraphNode)** of **faceSceneGraph** (defined in an **FDP** node) is updated by a facial animation parameter, and how the field is updated. If the face is in its neutral position, the **faceSceneGraphNode** has its **translation**, **scale**, and **rotation** fields set to the neutral values $(0,0,0)^T$, $(1,1,1)^T$, (0,0,1,0), respectively.

The **faceSceneGraphNode** field specifies the **Transform** node for which the animation is defined. The node shall be part of **faceScenegraph** as defined in the **FDP** node.

The **fieldId** field specifies which field in the **Transform** node, specified by the **faceSceneGraphNode** field, is updated by the FAP during animation. Possible fields are **translation**, **rotation**, **scale**.

If **fieldID**==1, **rotation** shall be updated using **rotationDef** and FAPValue.
If **fieldID**==2, **scale** shall be updated using **scaleDef** and FAPValue.
If **fieldID**==3, **translation** shall be updated using **translationDef** and FAPValue.

The **rotationDef** field is of type SFRotation. With **rotationDef**=(r_x,r_y,r_z,θ), the new value of the **rotation** field of the **Transform** node **faceSceneGraphNode** is:

rotation: **=**(r_x,r_y,r_z,θ*FAPValue*AU) [AU is defined in ISO/IEC FCD 14496-2]

The **scaleDef** field is of type SFVec3f. The new value of the **scale** field of the **Transform** node **faceSceneGraphNode** is:

scale:= FAPValue***scaleDef**

The **translationDef** field is of type SFVec3f. The new value of the **translation** field of the **Transform** node **faceSceneGraphNode** is:

translation:= FAPValue***translationDef**

FAP

Node interface

FAP {

ExposedField	SFNode	**viseme**	NULL
ExposedField	SFNode	**expression**	NULL
exposedField	SFInt32	**open_jaw**	+I
exposedField	SFInt32	**lower_t_midlip**	+I
exposedField	SFInt32	**raise_b_midlip**	+I
exposedField	SFInt32	**stretch_l_corner**	+I
exposedField	SFInt32	**stretch_r_corner**	+I
exposedField	SFInt32	**lower_t_lip_lm**	+I
exposedField	SFInt32	**lower_t_lip_rm**	+I
exposedField	SFInt32	**lower_b_lip_lm**	+I
exposedField	SFInt32	**lower_b_lip_rm**	+I
exposedField	SFInt32	**raise_l_cornerlip**	+I
exposedField	SFInt32	**raise_r_cornerlip**	+I
exposedField	SFInt32	**thrust_jaw**	+I
exposedField	SFInt32	**shift_jaw**	+I
exposedField	SFInt32	**push_b_lip**	+I
exposedField	SFInt32	**push_t_lip**	+I
exposedField	SFInt32	**depress_chin**	+I
exposedField	SFInt32	**close_t_l_eyelid**	+I
exposedField	SFInt32	**close_t_r_eyelid**	+I
exposedField	SFInt32	**close_b_l_eyelid**	+I
exposedField	SFInt32	**close_b_r_eyelid**	+I
exposedField	SFInt32	**yaw_l_eyeball**	+I
exposedField	SFInt32	**yaw_r_eyeball**	+I
exposedField	SFInt32	**pitch_l_eyeball**	+I
exposedField	SFInt32	**pitch_r_eyeball**	+I
exposedField	SFInt32	**thrust_l_eyeball**	+I
exposedField	SFInt32	**thrust_r_eyeball**	+I
exposedField	SFInt32	**dilate_l_pupil**	+I
exposedField	SFInt32	**dilate_r_pupil**	+I
exposedField	SFInt32	**raise_l_i_eyebrow**	+I
exposedField	SFInt32	**raise_r_i_eyebrow**	+I
exposedField	SFInt32	**raise_l_m_eyebrow**	+I

exposedField	SFInt32	**raise_r_m_eyebrow**	+I
exposedField	SFInt32	**raise_l_o_eyebrow**	+I
exposedField	SFInt32	**raise_r_o_eyebrow**	+I
exposedField	SFInt32	**squeeze_l_eyebrow**	+I
exposedField	SFInt32	**squeeze_r_eyebrow**	+I
exposedField	SFInt32	**puff_l_cheek**	+I
exposedField	SFInt32	**puff_r_cheek**	+I
exposedField	SFInt32	**lift_l_cheek**	+I
exposedField	SFInt32	**lift_r_cheek**	+I
exposedField	SFInt32	**shift_tongue_tip**	+I
exposedField	SFInt32	**raise_tongue_tip**	+I
exposedField	SFInt32	**thrust_tongue_tip**	+I
exposedField	SFInt32	**raise_tongue**	+I
exposedField	SFInt32	**tongue_roll**	+I
exposedField	SFInt32	**head_pitch**	+I
exposedField	SFInt32	**head_yaw**	+I
exposedField	SFInt32	**head_roll**	+I
exposedField	SFInt32	**lower_t_midlip_o**	+I
exposedField	SFInt32	**raise_b_midlip_o**	+I
exposedField	SFInt32	**stretch_l_cornerlip**	+I
exposedField	SFInt32	**stretch_r_cornerlip_o**	+I
exposedField	SFInt32	**lower_t_lip_lm_o**	+I
exposedField	SFInt32	**lower_t_lip_rm_o**	+I
exposedField	SFInt32	**raise_b_lip_lm_o**	+I
exposedField	SFInt32	**raise_b_lip_rm_o**	+I
exposedField	SFInt32	**raise_l_cornerlip_o**	+I
exposedField	SFInt32	**raise_r_cornerlip_o**	+I
exposedField	SFInt32	**stretch_l_nose**	+I
exposedField	SFInt32	**stretch_r_nose**	+I
exposedField	SFInt32	**raise_nose**	+I
exposedField	SFInt32	**bend_nose**	+I
exposedField	SFInt32	**raise_l_ear**	+I
exposedField	SFInt32	**raise_r_ear**	+I
exposedField	SFInt32	**pull_l_ear**	+I
exposedField	SFInt32	**pull_r_ear**	+I

}

NOTE — For the binary encoding of this node see Document MPEG-4 NODES A.1.40.

Functionality and semantics

This node defines the current look of the face by means of expressions and FAPs and gives a hint to TTS controlled systems on which viseme to use. For a definition of the facial animation parameters see ISO/IEC 14496-2, Annex C.

The **viseme** field shall contain a **Viseme** node.

The **expression** field shall contain an **Expression** node.

The semantics for the remaining fields are described in the ISO/IEC 14496-2, Annex C and in particular in Table C-1.

A FAP of value +I shall be interpreted as indicating that the particular FAP is uninitialized.

FDP

Node interface

FDP {

exposedField	SFNode	**featurePointsCoord**	NULL
exposedField	SFNode	**textureCoords**	NULL
exposedField	SFBool	**useOrthoTexture**	FALSE
ExposedField	MFNode	**faceDefTables**	[]
ExposedField	MFNode	**faceSceneGraph**	[]

}

NOTE — For the binary encoding of this node see Document MPEG-4 NODES A.1.41.

Functionality and semantics

The **FDP** node defines the face model to be used at the terminal. Two options are supported:

1. If **faceDefTables** is NULL, calibration information is downloaded, so that the proprietary face of the terminal can be calibrated using facial feature points and, optionally, the texture information. In this case, the **featurePointsCoord** field shall be set. **featurePointsCoord** contains the coordinates of facial feature points, as defined in ISO/IEC 14496-2, Annex C, Figure C-1, corresponding to a neutral face. If a coordinate of a feature point is set to +I, the coordinates of this feature point shall be ignored. The **textureCoord** field, if set, is used to map a texture on the model calibrated by the feature points. The **textureCoord** points correspond to the feature points. That is, each defined feature point shall have corresponding texture coordinates. In this case, the **faceSceneGraph** shall contain exactly one texture image, and any geometry it might contain shall be ignored. The terminal shall interpret the feature points, texture coordinates, and the **faceSceneGraph** in the following way:

Feature points of the terminal's face model shall be moved to the coordinates of the feature points supplied in **featurePointsCoord**, unless a feature point is to be ignored, as explained above.
If **textureCoord** is set, the texture supplied in the **faceSceneGraph** shall be mapped onto the terminal's default face model. The texture coordinates are derived from the texture coordinates of the feature points supplied in **textureCoords**. The **useOrthoTexture** field provides a hint to the decoding terminal that, when FALSE, indicates that the texture image is best obtained by cylindrical projection of the face. If **useOrthoTexture** is TRUE, the texture image is best obtained by orthographic projection of the face.

2. A face model as described in the **faceSceneGraph** is decoded. This face model replaces the terminal's default face model in the terminal. The **faceSceneGraph** shall contain the face in its neutral position (all FAPs = 0). If desired, the **faceSceneGraph** shall contain the texture maps of the face. The functions defining the way in which the **faceSceneGraph** shall be modified, as a function of the FAPs, shall also be decoded. This information is described by **faceDefTables** that define how the **faceSceneGraph** is to be modified as a function of each FAP. By means of **faceDefTables**, **IndexedFaceSets** and **Transform** nodes of the **faceSceneGraph** can be animated. Since the amplitude of FAPs is defined in units that are dependent on the size of the face model, the

featurePointsCoord field defines the position of facial features on the surface of the face described by **faceSceneGraph**. From the location of these feature points, the terminal computes the units of the FAPs. Generally, only two node types in the scene graph of a decoded face model are affected by FAPs: **IndexedFaceSet** and **Transform** nodes. If a FAP causes a deformation of an object (e.g. lip stretching), then the coordinate positions in the affected **IndexedFaceSets** shall be updated. If a FAP causes a movement which can be described with a **Transform** node (e.g. FAP 23, yaw_l_eyeball), then the appropriate fields in this **Transform** node shall be updated. It shall be assumed that this **Transform** node has its **rotation**, **scale**, and **translation** fields set to neutral values if the face is in its neutral position. A unique nodeId shall be assigned via the DEF statement to all **IndexedFaceSet** and **Transform** nodes which are affected by FAPs so that they can be accessed unambiguously during animation.

The **featurePointsCoord** field shall contain a **Coordinate** node that specifies feature points for the calibration of the terminal's default face. The coordinates are specified in the **point** field of the **Coordinate** node in the prescribed order, that a feature point with a lower label number is listed before a feature point with a higher label number.

EXAMPLE — Feature point 3.14 before feature point 4.1

The **textureCoords** field shall contain a **Coordinate** node that specifies texture coordinates for the feature points. The coordinates are listed in the **point** field in the **Coordinate** node in the prescribed order, that a feature point with a lower label is listed before a feature point with a higher label.

The **useOrthoTexture** field may contain a hint to the terminal as to the type of texture image, in order to allow better interpolation of texture coordinates for the vertices that are not feature points. If **useOrthoTexture** is FALSE, the terminal may assume that the texture image was obtained by cylindrical projection of the face. If **useOrthoTexture** is 1, the terminal may assume that the texture image was obtained by orthographic projection of the face.

The **faceDefTables** field shall contain **FaceDefTables** nodes. The behavior of FAPs is defined in this field for the face in **faceSceneGraph**.

The **faceSceneGraph** field shall contain a **Group** node. In the case of option 1 (above), this may be used to contain a texture image as described above. In the case of option 2, this shall be the grouping node for the face model rendered in the compositor and shall contain the face model. In this case, the effect of facial animation parameters is defined in the **faceDefTables** field.

FIT

Node interface

FIT {

exposedField	MFInt32	**FAPs**	[]
exposedField	MFInt32	**graph**	[]

```
    exposedField    MFInt32     numeratorTerms          []
    exposedField    MFInt32     denominatorTerms        []
    exposedField    MFInt32     numeratorExp            []
    exposedField    MFInt32     denominatorExp          []
    exposedField    MFInt32     numeratorImpulse        []
    exposedField    MFFloat     numeratorCoefs          []
    exposedField    MFFloat     denominatorCoefs        []
}
```

NOTE — For the binary encoding of this node see Document MPEG-4 NODES A.1.42.

Functionality and semantics

The **FIT** node allows a smaller set of FAPs to be sent during a facial animation. This small set can then be used to determine the values of other FAPs, using a rational polynomial mapping between parameters. In a **FIT** node, rational polynomials are used to specify interpolation functions.

EXAMPLE — The top inner lip FAPs can be sent and then used to determine the top outer lip FAPs. Another example is that only viseme and/or expression FAPs are sent to drive the face. In this case, low-level FAPs are interpolated from these two high-level FAPs.

To make the scheme general, sets of FAPs are specified, along with a FAP interpolation graph (FIG) between the sets that specifies which sets are used to determine which other sets. The FIG is a graph with directed links. Each node contains a set of FAPs. Each link from a parent node to a child node indicates that the FAPs in the child node can be interpolated from the parent node. **Expression** (FAP#1) or **Viseme** (FAP #2) and their fields shall not be interpolated from other FAPs.

In a FIG, a FAP may appear in several nodes, and a node may have multiple parents. For a node that has multiple parent nodes, the parent nodes are ordered as 1st parent node, 2nd parent node, etc. During the interpolation process, if this child node needs to be interpolated, it is first interpolated from 1st parent node if all FAPs in that parent node are available. Otherwise, it is interpolated from 2nd parent node, and so on.
An example of FIG is shown in Figure 5. Each node has a `nodeID`. The numerical label on each incoming link indicates the order of these links.

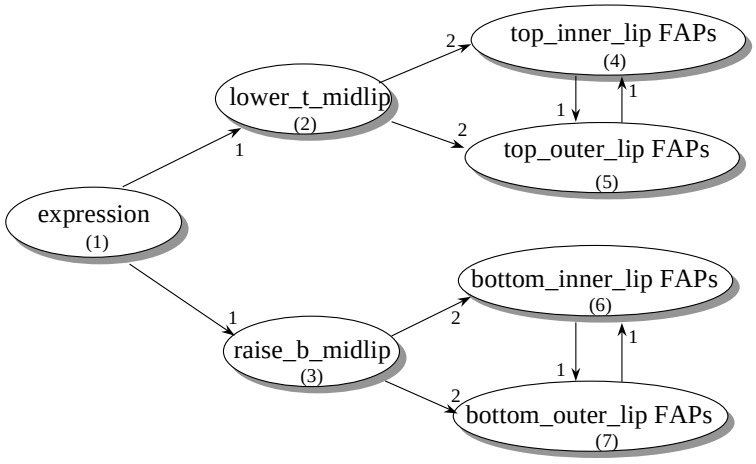


Figure 5 - A FIG example

The interpolation process based on the FAP interpolation graph is described using pseudo-C code as follows:

```
do {
   interpolation_count = 0;
   for (all Node_i) {   // from Node_1 to Node_N
      for (ordered Node_i's parent Node_k) {
         if (FAPs in Node_i need interpolation and
                FAPs in Node_k have been interpolated or are available) {
            interpolate Node_i from Node_k;  //using interpolation function
                                             // table here
            interpolation_count ++;
            break;
         }
      }
   }
} while (interpolation_count != 0);
```

Each directed link in a FIG is a set of interpolation functions. Suppose F_1, F_2, ..., F_n are the FAPs in a parent set and f_1, f_2, ..., f_m are the FAPs in a child set.
Then, there are m interpolation functions denoted as:
$f_1 = I_1(F_1, F_2, ..., F_n)$
$f_2 = I_2(F_1, F_2, ..., F_n)$
...
$f_m = I_m(F_1, F_2, ..., F_n)$
Each interpolation function I_k () is in a rational polynomial form if the parent node does not contain viseme FAP or expression FAP.

$$I(F_1, F_2, ..., F_n) = \sum_{i=0}^{K-1} (c_i \prod_{j=1}^{n} F_j^{l_{ij}}) \Big/ \sum_{i=0}^{P-1} (b_i \prod_{j=1}^{n} F_j^{m_{ij}})$$

Otherwise, an impulse function is added to each numerator polynomial term to allow selection of expression or viseme.

$$I(F_1, F_2, ..., F_n) = \sum_{i=0}^{K-1} \delta(F_{s_i} - a_i)(c_i \prod_{j=1}^{n} F_j^{l_{ij}}) \Big/ \sum_{i=0}^{P-1} (b_i \prod_{j=1}^{n} F_j^{m_{ij}})$$

In both equations, K and P are the numbers of polynomial products, c_i and b_i are the coefficient of the ith product. l_{ij} and m_{ij} are the power of F_j in the ith product. An impulse function equals 1 when $F_{s_i} = a_i$, otherwise, equals 0. F_{s_i} can only be viseme_select1, viseme_select2, expression_select1, and expression_select2. a_i is an integer that ranges from 0 to 6 when F_{s_i} is expression_select1 or expression_select2, ranges 0 to 14 when F_{s_i} is viseme_select1 or viseme_select2. The encoder shall send an interpolation function table which contains K, P, a_i, s_i, c_i, b_i, l_{ij}, m_{ij} to the terminal.

To aid in the explanation below, it is assumed that there are N different sets of FAPs with index 1 to N, and that each set has n_i, i=1,..,N parameters. It is also assumed that

there are L directed links in the FIG and that each link points from the FAP set with index P_i to the FAP set with index C_i, for i = 1, .. , L

The FAPs field shall contain a list of FAP-indices specifying which animation parameters form sets of FAPs. Each set of FAP indices is terminated by –1. There shall be a total of $N + n_1 + n_2 + ... + n_N$ numbers in this field, with N of them being –1. FAP#1 to FAP#68 are of indices 1 to 68. Fields of the **Viseme** FAP (FAP#1), namely, **viseme_select1**, **viseme_select2**, **viseme_blend**, are of indices from 69 to 71. Fields of the **Expression** FAP (FAP#2), namely, **expression_select1**, **expression_select2**, **expression_intensity1**, **expression_intensity2** are of indices from 72 to 75. When the parent node contains a **Viseme** FAP, three indices, 69, 70, 71, shall be included in the node (but not index 1). When a parent node contains an **Expression** FAP, four indices, 72,73,74,75, shall be included in the node (but not index 2).

The **graph** field shall contain a list of pairs of integers, specifying a directed links between sets of FAPs. The integers refer to the indices of the sets specified in the **FAPs** field, and thus range from 1 to N. When more than one direct link terminates at the same set, that is, when the second value in the pair is repeated, the links have precedence determined by their order in this field. This field shall have a total of 2L numbers, corresponding to the directed links between the parents and children in the FIG.

The **numeratorTerms** field shall be a list containing the number of terms in the polynomials of the numerators of the rational functions used to interpolate parameter values. Each element in the list corresponds to *K* in equation 1 above). Each link i (that is, the *i*th integer pair) in the **graph** field must have n_{Ci} values specified, one for each child FAP. The order in the **numeratorTerms** list shall correspond to the order of the links in the **graph** field and the order that the child FAP appears in the **FAPs** field. There shall be $n_{C1} + n_{C2} + ... + n_{CL}$ numbers in this field.

The **denominatorTerms** field shall contain a list of the number of terms in the polynomials of the denominator of the rational functions controlling the parameter value. Each element in the list corresponds to *P* in equation 1. Each link i (that is, the *i*th integer pair) in the **graph** field must have n_{Ci} values specified, one for each child FAP. The order in the **denominatorTerms** list corresponds to the order of the links in the **graph** field and the order that the child FAP appears in the **FAPs** field. There shall be $n_{C1} + n_{C2} + ... + n_{CL}$ numbers in this field.

The **numeratorImpulse** field shall contain a list of impulse functions in the numerator of the rational function for links with the **Viseme** or **Expression** FAP in parent node. This list corresponds to the $\delta(F_{s_i} - a_i)$. Each entry in the list is (s_i, a_i).

The **numeratorExp** field shall contain a list of exponents of the polynomial terms in the numerator of the rational function controlling the parameter value. This list corresponds to l_{ij}. For each child FAP in each link i, n_{Pi}*K values need to be specified.

The order in the **numeratorExp** list shall correspond to the order of the links in the **graph** field and the order that the child FAP appears in the **FAPs** field.

NOTE — K may be different for each child FAP.

The **denominatorExp** field shall contain a list of exponents of the polynomial terms of the denominator of the rational function controlling the parameter value. This list corresponds to m_{ij}. For each child FAP in each link i, n_{Pi}*P values need to be specified.

The order in the **denominatorExp** list shall correspond to the order of the links in the **graph** field and the order that the child FAP appears in the **FAPs** field.

NOTE — P may be different for each child FAP.

The **numeratorCoefs** field shall contain a list of coefficients of the polynomial terms of the numerator of the rational function controlling the parameter value. This list corresponds to c_i. The list shall have *K* terms for each child parameter that appears in a link in the FIG, with the order in **numeratorCoefs** corresponding to the order in **graph** and **FAPs**.

NOTE — K is dependent on the polynomial, and is not a fixed constant.

The **denominatorCoefs** field shall contain a list of coefficients of the polynomial terms in the numerator of the rational function controlling the parameter value. This list corresponds to b_i. The list shall have *P* terms for each child parameter that appears in a link in the FIG, with the order in **denominatorCoefs** corresponding to the order in **graph** and **FAPs**.

NOTE — P is dependent on the polynomial, and is not a fixed constant.

EXAMPLE — Suppose a FIG contains four nodes and 2 links. Node 1 contains FAP#3, FAP#3, FAP#5. Node 2 contains FAP#6, FAP#7. Node 3 contains an expression FAP, which means contains FAP#72, FAP#73, FAP#74, and FAP#75. Node 4 contains FAP#12 and FAP#17. Two links are from node 1 to node 2, and from node 3 to node 4. For the first link, the interpolation functions are

$$F_6 = (F_3 + 2F_4 + 3F_5 + 4F_3F_4^2)/(5F_5 + 6F_3F_4F_5)$$

$$F_7 = F_4 .$$

For the second link, the interpolation functions are

$$F_{12} = \delta(F_{72} - 6)(0.6F_{74}) + \delta(F_{73} - 6)(0.6F_{75})$$

$$F_{17} = \delta(F_{72} - 6)(-1.5F_{74}) + \delta(F_{73} - 6)(-1.5F_{75}) .$$

The second link simply says that when the expression is surprise (FAP#72=6 or FAP#73=6), for FAP#12, the value is 0.6 times of expression intensity FAP#74 or FAP#75; for FAP#17, the value is –1.5 tims of FAP#74 or FAP#75.

After the FIT node given below, we explain each field separately.

```
FIT {
   FAPs            [ 3 4 5 -1 6 7 –1 72 73 74 75 –1 12 17 -1]
   graph           [ 1 2 3 4]
   numeratorTerms  [ 4 1 2 2 ]
   denominatorTerms  [2 1 1 1]
   numeratorExp    [1 0 0   0 1 0   0 0 1  1 2 0  0 1 0
                        0 0 1 0  0 0 0 1  0 0 1 0  0 0 0 1 ]
   denominatorExp  [ 0 0 1  1 1 1  0 0 0
```

```
                    0 0 0 0  0 0 0 0 ]
  numeratorImpulse  [ 72 6  73 6  72 6  73 6 ]
  numeratorCoefs  [1 2 3 4   1   0.6 0.6  -1.5 –1.5 ]
  denominatorCoefs  [5 6 1 1 1 ]
}
```

`FAPs [ 3 4 5 -1 6 7 –1 72 73 74 75 –1 12 17 -1]`
Four sets of FAPs are defined, the first with FAPs number 3, 4, and 5, the second with FAPs number 6 and 7, the third with FAPs number 72, 73, 74, 75, and the fourth with FAPs number 12, 17.

`graph  [ 1 2 3 4]`
The first set is made to be the parent of the second set, so that FAPs number 6 and 7 will be determined by FAPs 3, 4, and 5. Also, the third set is made to be the parent of the fourth set, so that FAPs number 12 and 17 will be determined by FAPs 72, 73, 74, and 75.

`numeratorTerms [ 4 1 2 2]`
The rational functions that define F6 and F7 are selected to have 4 and 1 terms in their numerator, respectively. Also, the rational functions that define F12 and F17 are selected to have 2 and 2 terms in their numerator, respectively.

`denominatorTerms [ 2 1 1 1]`
The rational functions that define F6 and F7 are selected to have 2 and 1 terms in their denominator, respectively. Also, the rational functions that define F12 and F17 are selected to both have 1 term in their denominator.

`numeratorExp [ 1 0 0  0 1 0  0 0 1  1 2 0  0 1 0  0 0 1 0  0 0 0 1  0 0 1 0  0 0 0 1]`
The numerator selected for the rational function defining F6 is F3 + 2F4 + 3 F5 + 4F3F42. There are 3 parent FAPs, and 4 terms, leading to 12 exponents for this rational function. For F7, the numerator is just F4, so there are three exponents only (one for each FAP). Values for F12 and F17 are derived in the same way.

`denominatorExp [ 0 0 1  1 1 1  0 0 0  0 0 0 0  0 0 0 0]`
The denominator selected for the rational function defining F6 is 5F5+ 6F3F4F5 , so there are 3 parent FAPs and 2 terms and hence, 6 exponents for this rational function. For F7, the denominator is just 1, so there are three exponents only (one for each FAP). Values for F12 and F17 are derived in the same way.

`numeratorImpulse [72 6  73 6  72 6  73 6]`
For the second link, all four numerator polynomial terms contain impulse function $\delta(F_{72}-6)$ or $\delta(F_{73}-6)$.

`numeratorCoefs [ 1 2 3 4  1  0.6 0.6  -1.5 –1.5]`
There is one coefficient for each term in the numerator of each rational function.

`denominatorCoefs [ 5 6  1  1 1]`
There is one coefficient for each term in the denominator of each rational function.

MPEG-4 PROFILES AND LEVELS FOR FACE ANIMATION

MPEG-4 defines profiles to which decoders have to conform. A profile consists of objects defining the tools of the profile. Levels of a profile and object put performance and parameter limits on the tools. MPEG-4 Audio, Visual and Systems define parts of face animation.

A profile is a defined sub-set of the entire bitstream syntax that is defined by this part of ISO/IEC 14496. A level is a defined set of constraints imposed on parameters in the bitstream. All syntactic elements and parameter values which are not explicitly constrained may take any of the possible values that are allowed by this part of ISO/IEC 14496. A decoder shall be deemed to be conformant to a given profile at a given level if it is able to properly decode all allowed values of all syntactic elements as specified by that profile at that level.

Related to SNHC:

The profiles for synthetic and synthetic/natural hybrid visual content are:

- The Simple Facial Animation Visual Profile provides a simple means to animate a face model, suitable for applications such as audio/video presentation for the hearing impaired.
- The Scalable Texture Visual Profile provides spatial scalable coding of still image (texture) objects useful for applications needing multiple scalability levels, such as mapping texture onto objects in games, and high-resolution digital still cameras.
- The Basic Animated 2-D Texture Visual Profile provides spatial scalability, SNR scalability, and mesh-based animation for still image (textures) objects and also simple face object animation.
- The Hybrid Visual Profile combines the ability to decode arbitrary-shaped and temporally scalable natural video objects (as in the Core Visual Profile) with the ability to decode several synthetic and hybrid objects, including simple face and animated still image objects. It is suitable for various content-rich multimedia applications.

The Face Animation part is related to the Synthetic Visual Category of the Visual Profiles (composed by the Scaleable Texture prof. Ad the Simple Face Animation prof.)

The Simple Face Animation Profile:

All ISO/IEC 14496-2 facial animation decoders (for all object types) are required to generate at their output a facial model including all the feature points defined in this part of ISO/IEC 14496, even if some of the features points will not be affected by any information received from the encoder.
The Simple Face object is not required to implement the viseme_def/expression_def functionality.

Level 1:

number of objects: 1,
The total FAP decode frame-rate in the bitstream shall not exceed 72 Hz ,
The decoder shall be capable of a face model rendering update of at least 15 Hz, and
Maximum bitrate 16 kbit/s.

Level 2:

maximum number of objects: 4,
The FAP decode frame-rate in the bitstream shall not exceed 72 Hz (this means that the FAP decode framerate is to be shared among the objects),
The decoder shall be capable of rendering the face models with the update rate of at least 60Hz, sharable between faces, with the constraint that the update rate for each individual face is not required to exceed 30Hz, and
Maximum bitrate 32 kbit/s.

From audio profiles and from systems profiles:

In MPEG-4 Audio, the TTSI with the bookmark identifiers for face animation as well as the interface to the Phoneme/Bookmark-to-FAP-converter is defined. It is part of all Audio profiles. Using a TTS, any Audio profile and a Visual profile containing the Face object allows to define

interactive services with face animation at extremely low data rates. Without using a TTS, any Audio profile and a Visual profile containing the Face object allows to play speech and animate the proprietary face model.

In order to enable the specification of the face, the BIFS node FDP and its children have to be transmitted. This is possible for terminals that support the Complete Scene Graph profile and the Complete Graphics profile.

Annex C

MPEG-4 FACE OBJECT DESCRIPTION

SF – marks that is a single item
MF – marks that is a vector item

SFNode Face(#36)
{

- SFNode FIT (#42) [NULL]
  {
  - MFInt32 FAPs [] – enumeration of the FAP used for the interpolation each group separated by –1.
  - MFInt32 graph
  - MFInt32 numeratorTerms
  - MFInt32 denominatorTerms
  - MFInt32 numeratorExp
  - MFInt32 denominatorExp
  - MFInt32 numeratorImpulse
  - MFFloat numeratorCoefs
  - MFFloat denominatorCoefs

  }
- SFNode FDP(#41)[NULL]
  {
  - SFNode featurePointCoordinate(#24) – Coordinate – MFVec3f point – NULL: It shows the FPD 3D coord. in the specific order. [NULL]
  - SFNode textureCoords(#25) – Coordinate2D – MFVec2f – It shoe the texture coordinates in the specific order. [NULL]
  - SFBool useOrthoTexture – Tells the kind of projection to be used: true = orthogonal, false = cylindrical. [FALSE]
  - MFNode faceDefTables (#38) []
    {
    - SFInt32 fapID – is the ID number of the described FAP [0]
    - SFInt32 highLevelSelect – 1 → we are dealing with a viseme; 2 → we are dealing with an expression; other → ignore it. [0]
    - MFNode faceDefMesh(#37)[]
      {
      - SFNode faceSceneGraphNode – IndexedFaceSet(#48) – It contains the coordinates belonging to the part we want to move. It should be a subset of the sceneGraph field of the FPD node [NULL] LOOK: ISO/IEC … subclause 6.23 [10]
        {
        - MFInt32 set_colorIndex
        - MFInt32 set_coordIndex
        - MFInt32 normalIndex
        - MFInt32 set_texCoordIndex
        - SFNode color [NULL]
        - SFNode coord [NULL]
        - SFNode normal [NULL]
        - SFNode texCoord [NULL]
        - SFBool ccw [TRUE]
        - MFInt32 colorIndex []
        - SFBool colorPerVertex [TRUE]
        - SFBool convex [TRUE]
        - MFInt32 coordIndex []
        - SFFloat creaseAngle 0.0
        - MFInt32 normalIndex[]
        - SFBool normalPervertex[TRUE]
        - SFBool solid [TRUE]
        - MFInt32 texCoordIndex []

        }
      - MFInt32 intervalBorders – specifies interval borders for the piece-wise linear approximation in increasing order. One interval border shall have the value 0. []
      - MFInt32 coordIndex – the number of the coordinate of the previous IndexedFaceSet whose movement we want to define []
      - MFVec3f displacement – for each Index it should display the displacement vector for each interval. []

      }
    - MFNode faceDefTransform (#39)[]
      {
      - SFNode faceSceneGraphNode – a subset already defined in the sceneGraph of the FDP. [NULL]
      - SFInt32 fieldId – 1 → rotation; 2 → scale; 3 → translation [1]
      - SFRotation rotationDef [0, 0, 1, 0]
      - SFVec3f scaleDef [1, 1, 1]
      - SFVec3f translationDef [0, 0, 0]

      }

  }

- MFNode faceSceneGraph [] – ALREADY DESCRIBED

}

- SFNode FAP (#40) – Description of Facial Animation Parameters. [NULL]
  {
  - SFNode viseme – VISEME node (#97). [NULL]
  - SFNode expression – EXPRESSION node (#34). [NULL]
  - SFInt32 open_jaw [+I]
  - … all FAPs, +I means the FAP has not been initialized.
  - SFInt32 pull_r_ear [+I]

  }
- SFNode ttsSource - realted to AudioSource if used a TTS if not NULL. [NULL]
- MFNode renderedFace [NULL] It is the face scene graph after all FAPs are applied.

}

BIBLIOGRAPHY

1. J.-L. Dugelay, Katia Fintzel, S. Valente,. *Synthesitc Natural Hybrid Coding for Virtual Teleconferencing Systems.* IEEE Picture Coding Symposium; April 21-23 1999; Portland Oregon USA

2. S. Valente, J.-L. Dugelay. *Face Tracking and Realistic Animations for Telecommunicant Clones.* IEEE Multimedia Jan-Mar 2000

3. S. Valente, J.-L. Dugelay. *A Visual Analysis/Synthesis Feedback Loop for Accurate Face Tracking* Preprint submitted to Elsevier Preprint. 2000

4. S. Valente. *Analyse, Synthèse et Animation de Clones dans un contexte de Téléréunion Virtuelle.* Ph.D. thesis presented the 2nd of November 1999 at the Institut Eurécom.

5. ISO/IEC 14496-1 MPEG-4 Part 1: Systems. Atlantic City, November 1998

6. ISO/IEC 14496-2 MPEG-4 Part 2: Visual. Maui, December 1999

7. *Tutorial Issue on the MPEG-4 Standard.* Signal Processing Image Communication. Vol. 15, Nos. 4-5, January 2000

8. S. Valente, A. C. Andrés del Valle, J.-L. Dugelay. *Analysis and Reproduction of Facial Expressions for Realistic Communicating Clones.* To appear in The Journal of VLSI and Signal Processing, Autumn 2000.