PERSONALIZED FACE ANIMATION IN SHOWFACE SYSTEM

Ali Arya
Babak Hamidzadeh

Dept. of Electrical & Computer Engineering, University of British Columbia,
2356 Main Mall, Vancouver, BC, Canada V6T 1Z4, Phone: (604) 822-9181, Fax: (604) 822-5949
Email: {alia,babak}@ece.ubc.ca

The problem of Personalized Face Animation is addressed in this paper by proposing *ShowFace* streaming structure. This structure is based on most widely accepted industry standards in multimedia presentation like MPEG-4 and SMIL, and extends them by defining image transformations required for certain facial movements, and also by providing a higher level Face Modeling Language (FML) for modeling and control purposes. It defines a comprehensive framework for face animation consisting of components for parsing the input script, generating and splitting the audio and video "behaviors", creating the required images and sounds, and eventually displaying or writing the data to files. This component-based design and scripted behavior make the framework suitable for many purposes including web-based applications.

Keywords – Face Animation, Multimedia, Image Transformation, Talking Heads, Modeling Language, XML.

1. Introduction

Recent developments in the areas like Virtual Reality, Video Conferencing, Games, and Agent-based Online Applications have drawn a considerable attention to character animation. Replacing audio/visual data of "real people" with multimedia presentations based on "virtual agents" seem to be very beneficial and in some cases necessary. Saving bandwidth in video conferencing by replacing video data with animation "commands" can be considered an example of the former cases, while creating new scenes with "unavailable" characters is an example of the latter.

Personalized Face Animation includes all the information and activities required to create a multimedia presentation resembling a specific person. The input to such system can be a combination of audio/visual data and textual commands and descriptions. A successful face animation system needs to have efficient yet powerful solutions for providing and displaying the content, i.e. a content description format, decoding algorithms, and finally an architecture to put different components together in a flexible way.

In a simple example, the input data can be simply the compressed (encoded) image frames, and the output will be the same images, decompressed (decoded) and displayed on screen one after another without any specific processing or other components. But more complicated cases can be imagined when the input is a textual description of desired scenes, and decoding involves creation of audio/visual data as requested. Here a set of different components may be necessary to parse and interpret the input, create audio and video streams, synchronize them, and eventually play the multimedia presentation.

Considering three major issues of Content Delivery, Content Creation, and Content Description, following features can be assumed as important requirements in a multimedia presentation system:

- Streaming, i.e. continuously receiving/displaying data
- Structured Content Description, i.e. a hierarchical way to provide information about the required content from high level scene description to low level moves, images, and sounds
- Generalized Decoding, i.e. creating the displayable content based on the input. This can be decoding a compressed image or making a new image as requested.
- Component-based Architecture, i.e. the flexibility to rearrange the system components, and use new ones as long as a certain interface is supported.
- Compatibility, i.e. the ability to use and work with widely accepted industry standards in multimedia systems.
- Minimized Database of audio/visual footage.

The technological advances in multimedia systems, speech/image processing, and computer graphics, and also new applications specially in computer-based games, telecommunication, and online services, have resulted in a rapidly growing number of publications regarding these issues. These research achievements, although very successful in their objectives, mostly address a limited subset of the above requirements. A comprehensive framework for face animation is still in conceptual stages. The *ShowFace* system, proposed in this paper, is a step toward such a framework. In Section 2, some of the related works are briefly reviewed. The basic concepts and structure of *ShowFace* system are discussed in Section 3 to 5. Some experimental results and conclusions are the topics of Sections 6 and 7, respectively.

2. Related Works

2.1. View Generation

3D head models have long been used for facial animation [3,16,18]. Such models provide a powerful means of head reconstruction in different views and situations, but they usually need expensive hardware and complicated algorithms and lack the realistic appearance. Recent approaches have shown successful results in creating 3D models from a limited number of 2D photographs [16].

2D image-based methods are another alternative to face construction. Image morphing is an early mechanism for generating new images based on existing ones [1,10]. The most difficult task in morphing is finding control points, which is usually done manually. MikeTalk [10] is an image-based system, which uses optical flow to solve the correspondence problem. It performs automated morphing and creates visual speech based on pre-recorded *visemes*. The main issues are the limited ability in creating different facial images (e.g. moves and expressions), non-effectiveness of optical flow in detecting corresponding facial features specially in movements, and also required image database for each person.

Bregler , et al [5], combine a new image with parts of existing footage (mouth and jaw) to create new talking views. This method is also limited to a certain view where the recordings have been made. No transformation is proposed to make a talking view after some new movements of the head. In a more recent work based on [10], Graf, et al [12], propose recording of all visemes in a range of possible views, so after detecting the view (pose) proper visemes will be used. This way talking heads in different views can be animated but the method requires a considerably large database.

TalkingFace [1] combines optical flow and facial feature detection to overcome these issues. It can learn certain image transformations needed for talking (and to some degrees, expressions and head movements) and apply them to any given image. Tiddeman, et al, [19] show how such image transformations can be extended to include even facial texture.

2.2. Speech Synthesis

Text-To-Speech (TTS) systems which produce an audio output based on a text input, have long been studied [8]. There are several algorithms for TTS conversion. Two main approaches are concatenative and rule-based. The former uses a pre-recorded set of speech units and creates a desired speech by connecting them. The latter is based on a model of the vocal tract and synthetically generates the speech.

In either case, the basic functions are text normalization (separating words), word pronunciation (creating a sequence of phonemes), prosodic analysis (determining the pitch, speed, and volume of syllables), and finally audio production (synthetic or concatenative). Commercial and free toolboxes are available for partial help in these functions.

2.3. Content Description

Multimedia Content description has been the subject of some research projects and industry standards. In case of face animation, Facial Action Coding System (FACS) [9] was one the first attempts to model the low level movements, which can happen on a face. MPEG-4 standard [2] extends this by Face Definition/Animation Parameters for facial features and their movements. These parameters can define low-level moves of features (e.g. jaw-down) and also higher-level set of moves to form a complete facial situation (e.g. visemes or expressions). The standard does not go any further to define a dynamic description of the scene (facial activities and their temporal relation) and is limited to static snapshots. Decoding these parameters and the underlying face model are out of scope of MPEG-4 standard.

Indirectly related is Synchronized Multimedia Integration Language, SMIL [6], an XML-based language for dynamic (temporal) description of the events in a general multimedia presentation. It defines time containers for sequential, parallel, and exclusive actions related to contained objects, in order to synchronize the events. SMIL does not act as dynamic content description for facial animation or any other specific application.
BEAT [7] is another XML-based system, specifically designed for human animation purposes. It is a toolkit for automatically suggesting expressions and gestures, based on a given text to be spoken. BEAT uses a knowledge base and rule set, and provides synchronization data for facial activities, all in XML format. This enables the system to use standard XML parsing and scripting capabilities. Although BEAT is not a general content description tool, but it demonstrates some of the advantages of XML-based approaches.

Scripting and behavioral modeling languages for virtual humans are considered by other researchers as well [11,13,14,16]. These languages are usually simple macros for simplifying the animation, or new languages which are not using existing multimedia technologies. Most of the time, they are not specifically designed for face animation.

2.4. Architectural Issues

Different architectures are also proposed to perform facial animation, specially as an MPEG-4 decoder/player [4,17]. Although they try to use platform-independent and/or standard technologies (e.g. Java and VRML), they are usually limited to certain face models and lack a component-based and extensible structure, and do not propose any content description mechanism more than standard MPEG-4 parameters.

In a more general sense, a variety of structures and standards have been proposed for multimedia streaming. Currently, the most considerable platforms available are Windows Media, Real Player, and QuickTime [15]. They try to comply with the industry standards, as mentioned above, and relying on one of them, although results in some limitations, can provide the benefit of using existing system support and functionality, e.g. in multimedia streaming.

3. Structured Content Description

3.1. Design Ideas

Describing the contents of a multimedia presentation is a basic task in multimedia systems. It is necessary when a client asks for a certain presentation to be designed, when a media player receives input to play, and even when a search is done to retrieve an existing multimedia file. In all these cases, the description can include raw multimedia data (video, audio, etc) and textual commands and information. Such a description works as a Generalized Encoding, since it represents the multimedia content in a form not necessarily the same as the playback format, and is usually more efficient and compact. For instance a textual description of a scene can be a very effective "encoded" version of a multimedia presentation that will be "decoded" by the media player when it recreates the scene.

Although new streaming technologies allow real-time download/playback of audio/video data, but bandwidth limitation and its efficient usage still are, and probably will be, major issues. This makes a textual description of multimedia presentation (in our case facial actions) a very effective coding/compression mechanism, provided the visual effects can be recreated with a minimum acceptable quality.

Efficient use of bandwidth is not the only advantage of facial action coding. In many cases, the "real" multimedia data does not exist at all, and has to be created based on a description of desired actions. This leads to the whole new idea of representing the spatial and temporal relation of the facial actions. In a generalized view, such a description of facial presentation should provide a hierarchical structure with elements ranging from low level "images", to simple "moves", more complicated "actions", to complete "stories". We call this a Structured Content Description, which also requires means of defining capabilities, behavioral templates, dynamic contents, and event/user interaction. Needless to say, compatibility with existing multimedia and web technologies is another fundamental requirement, in this regard.

Face Modeling Language (FML) is a Structured Content Description mechanism based on eXtensible Markup Language. The main ideas behind FML are:

- Hierarchical representation of face animation
- Timeline definition of the relation between facial actions and external events
- Defining capabilities and behavior templates
- Compatibility with MPEG-4 FAPs
- Compatibility with XML and related web technologies

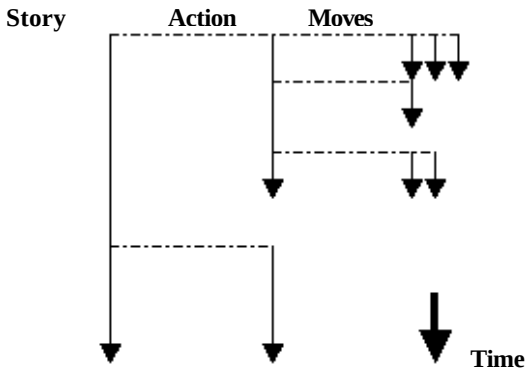


Fig. 1. FML Timeline and Temporal Relation of Face Activities

FACS and MPEG-4 FAPs provide the means of describing low-level face actions but they do not cover temporal relations and higher-level structures. Languages like SMIL do this in a general purpose form for any multimedia presentation and are not customized for specific applications like face animation. A language bringing the best of these two together, customized for face animation, seems to be an important requirement. FML is designed to do so.

Fundamental to FML is the idea of Structured Content Description. It means a hierarchical view of face animation capable of representing simple individually-meaningless moves to complicated high level stories. This hierarchy can be thought of as consisting of the following levels (bottom-up):

- Frame, a single image showing a snapshot of the face (Naturally, may not be accompanied by speech)
- Move, a set of frames representing linear transition between two frames (e.g. making a smile)
- Action, a "meaningful" combination of moves
- Story, a stand-alone piece of face animation

The boundaries between these levels are not rigid and well defined. Due to complicated and highly expressive nature of facial activities, a single move can make a simple yet meaningful story (e.g. an expression). The levels are basically required by content designer in order to:

- Organize the content
- Define temporal relation between activities
- Develop behavioural templates, based on his/her presentation purposes and structure.

FML defines a timeline of events (Figure 1) including head movements, speech, and facial expressions, and their combinations. Since a face animation might be used in an interactive environment, such a timeline may be altered/determined by a user. So another functionality of FML is to allow *user interaction* and in general *event handling* (Notice that user input can be considered a special case of *external event*.). This event handling may be in form of:

- Decision Making; choosing to go through one of possible paths in the story
- Dynamic Generation; creating a new set of actions to follow

A major concern in designing FML is compatibility with existing standards and languages. Growing acceptance of MPEG-4 standard makes it necessary to design FML in a way it can be translated to/from a set of FAPs. Also due to similarity of concepts, it is desirable to use SMIL syntax and constructs, as much as possible.

3.2. Primary Language Constructs

FML is an XML-based language. The choice of XML as the base for FML is based on its capabilities as a markup language, growing acceptance, and available system support in different platforms. Figure 2 shows typical structure of an FML.

An FML document consists, at higher level, of two types of elements: **model** and **story**. A **model** element is used for defining face capabilities, parameters, and initial configuration. A **story** element, on the other hand, describes the timeline of events in face animation. It is possible to have more than one of each element but due to possible sequential execution of animation in streaming applications, a **model** element affect only those parts of document coming after it.

```
<fml>
     <model>           <!-- Model Info -->
          <model-info />
     </model>
     <story>           <!-- Story Timeline -->
          <action>
               <time-container>
               <move-set>
                    < . . . >
               <move-set>
               < . . . >
               </time-container>
               < . . . >
          </action>
          < . . . >
     </story>
</fml>
```

Fig. 2. FML Document Map; Time-container and move-set will be replaced by FML time container elements and sets of possible activities, respectively.

Face animation timeline consists of facial activities and their temporal relations. These activities are themselves sets of simple Moves. The timeline is primarily created using two time container elements, **seq** and **par** representing sequential and parallel move-sets. A **story** itself is a special case of sequential time container. The begin times of activities inside a **seq** and **par** are relative to previous activity and container begin time, respectively.

```
<seq begin="0">
     <talk begin="0">Hello World</talk>
     <hdmv begin="0" end="5" type="0" val="30" />
</seq>
<par begin="0">
     <talk begin="1">Hello World</talk>
     <exp begin="0" end="3" type="3" val="50" />
</par>
```

Fig. 3. FML Primary Time Container

FML supports three basic face activities: talking, expressions, and 3D head movements. They can be a simple Move (like an expression) or more complicated (like a piece of speech). Combined in time containers, they create FML Actions. This combination can be done using nested containers, as shown in Figure 4.

```
<action>
<par begin="0">
<seq begin="0">
          <talk begin="0">Hello World</talk>
     <hdmv begin="0" end="5" type="0" val="30" />
</seq>
     <exp begin="0" end="3" type="3" val="50" />
</par>
</action>
```

Fig. 4. Nested Time Container

FML also provides the means for creating a behavioral model for the face animation. At this time, it is limited to initialization data such as range of possible movements and image/sound database, and simple behavioral templates (subroutines). But it can be extended to include behavioral rules and knowledge bases, specially for interactive applications. A typical **model** element is illustrated in Figure 5, defining a behavioral template used later in **story**.

```
<model>
     <img src="me.jpg" />
     <range dir="0" val="60" />
     <template name="hello" >
<seq begin="0">
     <talk begin="0">Hello</talk>
          <hdmv begin="0" end="5" dir="0" val="30" />
</seq>
     </template>
</model>
<story>
     <behavior template="hello" />
</story>
```

Fig. 5. FML Model and Templates

3.3. Event Handling and Decision Making

Dynamic interactive applications require the FML document to be able to make decisions, i.e. to follow different paths based on certain events. To accomplish this **excl** time container and **event** element are added. An event represents any external data, e.g. the value of a user selection. The new time container associates with an event and allows waiting until the event has one of the given values, then it continues with action corresponding to that value.

```
<event id="user" val="-1" />
<excl ev_id="user">
     <talk ev_val="0">Hello</talk>
     <talk ev_val="1">Bye</talk>
</excl>
```

Fig. 6. FML Decision Making and Event Handling

3.4. Compatibility

The XML-based nature of this language allows the FML documents to be embedded in web pages. Normal XML parsers can extract data and use them as input to an FML-enabled player, through simple scripting. Such a script can also use XML Document Object Model (DOM) to modify the FML document, e.g. adding certain activities based on user input. This compatibility with web browsing environments, gives another level of interactivity and dynamic operation to FML-based system, as illustrated in Section 5.

Another aspect of FML is its compatibility with MPEG-4 face definition/animation parameters. This has been achieved by:

- Translation of FML documents to MPEG-4 codes by the media player.
- Embedded MPEG-4 elements (**fap** element is considered to allow direct embedding of FAPs in FML document)

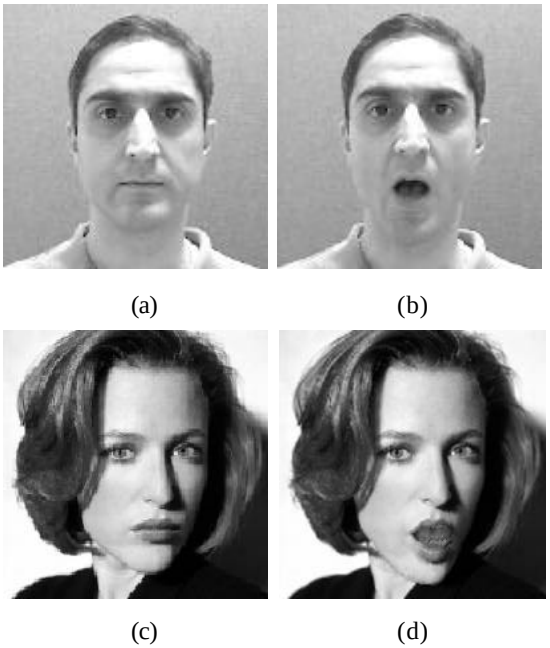
4. Content Creation

4.1. Feature-based Image Transformation

FML parser component of *ShowFace* system determines the visual activities required in the face. These activities are transitions between certain face *states* like a viseme or expression. In a training phase, a set of image-based transformations is learned by the system, which can map between these face states. Transformations are found by tracking facial features when the model is performing the related transitions, and then applied to a given image, as illustrated in Figure 7. A library of transformations is created based on following facial states:

- Visemes in full-view
- Facial expressions in full-view
- Head movements

For group 1 and 2, mappings for all the transitions between a non-talking neutral face and any group member are stored. In group 3, this is done for transitions between any two neighboring states (Figure 8).



(a) (b)

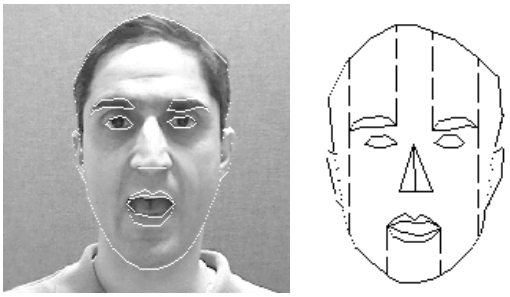
(c) (d)

Fig. 7. Facial Features and Image Transformations, (a) model state 1, (b) model state 2, (c) target character in state 1, (d) target character in state 2

Each transformation is defined in form of $T=(F,M)$ where T is the transformation, F is the feature set in the source image, and M is the mapping values for features. Source image information is saved to enable scaling and calibration, explained later. The feature set for each image includes face boundary, eyes and eye-brows, nose, ears, and lips. These feature lines, and the facial regions created by them are shown in Figure 9

The solid lines are feature lines surrounding feature regions, while dashed lines define face patches. The patches are defined in order to allow different areas of the face to be treated differently. Covisibility is the main concern when defining these face patches. Points in each patch will be mapped to points in the corresponding patch of the target image, if visible.

Fig. 8. Moving Head States. The same three states exist for rotating to left, in addition to a full-view image, at the center.



(a) (b)

Fig. 9. Facial Regions are defined as areas surrounded by two facial feature lines, e.g. inside eyes or between lower lip and jaw. Some face patches are removed from (b) for simplicity.

The transformations are done by first applying the mapping vectors for the feature points. This is shown in Figure 10. Simple transformations are those which have already been learned, e.g. T_{12} and T_{13} (assuming we only have *Image-1*) Combined transformations are necessary in cases when the target image is required to have the effect of two facial state transitions at the same time, e.g. T_{14}.

Due to non-orthographic nature of some head movements, combined transformations involving 3D head rotation can not be considered a linear combination of some known transformations. Feature mapping vectors for talking and expressions (which are learned from frontal view images) need to be modified when applied to "moved" heads.

$$T_{14} = a\ T_{12} + b\ T'_{13}$$
$$T'_{13} = f_p(T_{12}, , T_{13}) = T_{24}$$

where f_p is Perspective Calibration Function, and a and b are coefficients between 0 and 1 to control transition progress. T_{13} will also be scaled based on face dimensions in source/target images.

When the *Image-2* is given, i.e. the new image does not have the same orientation as the one used in learning, the required transformation is T_{24} which still needs scaling/perspective calibration based on T_{13} and T_{12}.

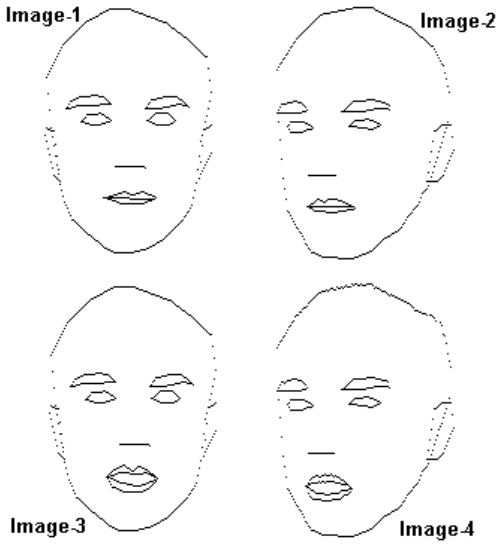


Fig. 10. Feature Transformation. T_{ij} is the transformation between Image-i and Image-j.

4.2. Facial Region Transformation

The stored transformations only show the mapping vectors for feature lines. Non-feature points are mapped by interpolating the mapping values for the feature lines surrounding their regions. This is done based on the face region to which a point belongs.

Face regions are grouped into two different categories:

- Feature islands, surrounded by one or two "inner" feature lines
- Face patches, covering the rest of the face as shown in Figure 4-b.

The mapping vector for each point inside a group-1 region is found using the following formula:

$$\boldsymbol{d}_{r,c} = w(\boldsymbol{d}_{u,c}, \boldsymbol{d}_{l,c})$$

where the function w is a weighted average with distance as the weights, r and c are row and column in image for the given point, u and l are the row number for top and bottom feature points, and $\boldsymbol{d}$ is the mapping vector.

Face patches are defined based on covisibility, i.e. their points are most likely to be seen together. Defining the patches is necessary in order to preserve the geometric validity of the transformation. The mapping vector of each point in a patch is the weighted average of mapping of all the patch corners. Extra checking is performed to make sure a point inside a patch will be mapped to another point in corresponding patch of target image.

4.3. Speech Synthesis

To achieve the best quality with minimum database requirements, *ShowFace* uses a concatenative approach to speech synthesis. Diphones (the transitions between the steady-state of a phoneme to the steady-state of another) are used as the basis of this approach. An off-line diphone-extraction tool is designed to create a database of diphones from existing audio footage. This database is normalized for power and pitch to provide a smooth initial set. The diphones are then dynamically connected based on the phoneme list of a given text to be spoken.

An FFT-based comparison of diphones finds the best connection point for two diphones at run time. This results in a dynamic time length calculation for diphones and words which will then be used to find the necessary duration of the corresponding visual transitions and the number of frames to be generated, in order to achieve a lip-synchronized audio-visual stream.

5. Showface Framework

5.1. System Architecture

The basic structure of *ShowFace* system is illustrated in Figure 11. Five major parts of this system are:

- Script Reader, to receive an FML script from a disk file, an Internet address, or any text stream provider.
- Script Parser, to interpret the FML script and create separate intermediate audio and video descriptions (e.g. words and viseme identifiers)
- Video Kernel, to generate the required image frames
- Audio Kernel, to generate the required speech
- Multimedia Mixer, to synchronize audio and video streams

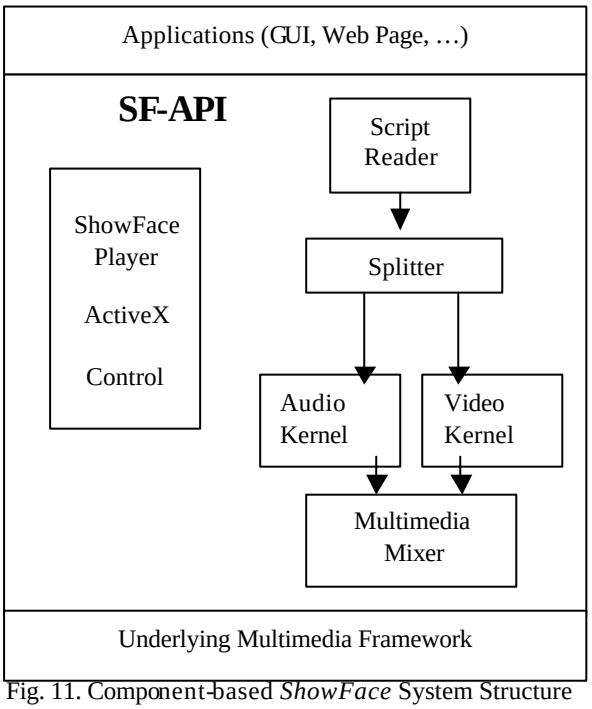


Fig. 11. Component-based *ShowFace* System Structure

ShowFace relies on the underlying multimedia technology for audio and video display. The system components interact with each other using ShowFace Application Programming Interface, SF-API, a set of interfaces exposed by the components and utility functions provided by *ShowFace* run-time environment. User applications can access system components through SF-API, or use a wrapper object called *ShowFacePlayer*, which exposes the main functionality of the system, and hides programming details.

ShowFace system is designed and implemented with the concept of openness in mind. By that we mean the ability to use and connect to existing standard components and also independent upgrade of the system modules. To make most use of existing technologies, *ShowFace* components are implemented as Microsoft DirectShow filters.

DirectShow is a multimedia streaming framework, which allows different media processing to be done by independent objects called *filters*, which can be connected using standard Component Object Model (COM) interfaces. DirectShow will be installed as part of many application programs like browsers and games, and comes with a set of filters like audio and video decoders and renderers. This allows *ShowFace* objects to access these filters easily and rely on multimedia streaming services provided by DirectShow, e.g. receiving data from a URL reader or MPEG-4 decoder and sending data to a video player or file writer.

The *ShowFacePlayer* wrapper object is implemented as an ActiveX control, which can be easily used in web pages and other client applications. An off-line tool, *ShowFace Studio*, is also developed to assist in detecting the features, creating the maps, and recording the FML scripts. Some samples of transformed faces are shown in Figure 4.

6. Case Studies And Experimental Results

6.1. Static Document

The first case is a simple FML document without any need for user interaction. There is one unique path the animation follows. The interesting point in this basic example is the use of loop structures, using **repeat** attribute included in any activity.

The **event** element specifies any external entity whose value can change. The default value for **repeat** is 1. If there is a numerical value, it will be used. Otherwise, it must be an **event** id, in which case the value of that **event** at the time of execution of related activity will be used. An FML-compatible player should provide means of setting external events values. *ShowFacePlayer* has a method called *SetFaceEvent,* which can be called by the owner of player object to simulate external events.

```
<event id="select" val="2" />
< . . . >
<seq repeat="select">
     <talk begin="0">Hello World</talk>
<exp begin="0" end="3" type="3" val="50" />
</seq>
```

Fig. 12. Repeated Activity. Using event is not necessary

6.2. Event Handling

The second example shows how to define an external event, wait for a change in its value, and perform certain activities based on the value. An external event corresponding to an interactive user selection is defined, first. It is initialized to –1 that specifies an invalid value. Then, an **excl** time container, including required activities for possible user selections, is associated with the event. The **excl** element will wait for a valid value of the event. This is equivalent to a pause in face animation until a user selection is done.

It should be noted that an FML-based system usually consists of three parts:

- FML Document
- FML-compatible Player
- Owner Application

In a simple example like this, it could be easier to simply implement the "story" in the owner application and send simpler commands to a player just to create the specified content (e.g. face saying Hello). But in more complicated cases, the owner application may be unaware of desired stories, or unable to implement them. In those cases, e.g. interactive environments, the owner only simulates the external parameters.

```
function onLoad()
{
     facePlayer.ReadFML("test.fml");
     facePlayer.Run();
}
function onHelloButton()
{
     facePlayer.SetFaceEvent("user", 0);
}
function onByeButton()
{
     facePlayer.SetFaceEvent("user", 1);
}
```

Fig. 13. JavaScript Code for FML Event shown in Figure 6

6.3. Dynamic Content Generation

The last FML example to be presented illustrates the use of XML DOM to dynamically modify the FML document and generate new animation activities.

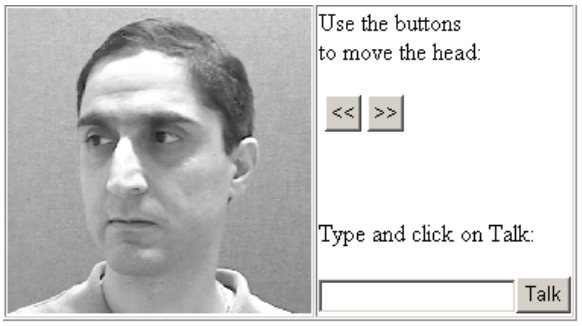


Fig. 14. Dynamic FML Generation

The simplified and partial JavaScript code for the web page shown in Figure 10 looks like this:

```
function onRight()
{
      var fml = fmldoc.documentElement;
      var new = fmldoc.createElement(“hdmv”);
      new.setAttribute(“dir”,”0”);
      new.setAttribute(“val”,”30”);
      fml.appendChild(new);
}
```

More complicated scenarios can be considered, using this dynamic FML generation, for instance, having a form-based web page and asking for user input on desired behavior, and using templates in **model** section of FML.

6.4. Sample Images

Figure 15 shows some sample outputs of the image transformations.

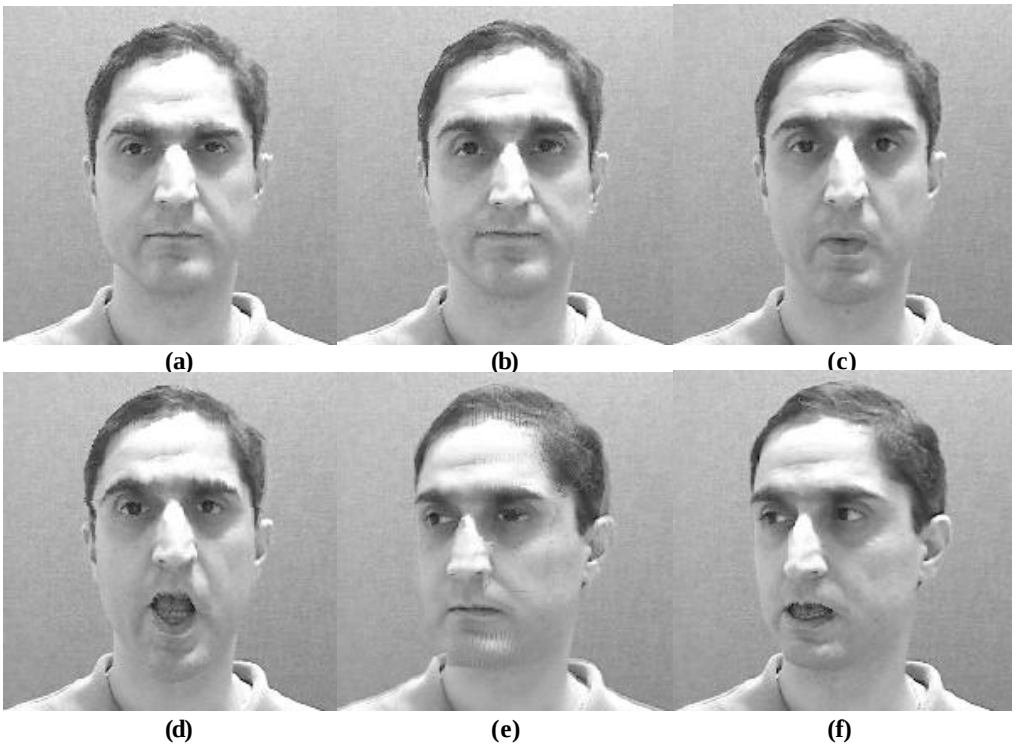


Fig. 15. Transformed Faces, mapped from 7-a. frown (a), smile (b), visemes for sounds “oo” and “a” in “root” and “talk” (c and d), rotate to right (e), and non-frontal talking (f)

7. Conclusion and Future Works

An approach to a comprehensive framework for face animation is proposed. The main objectives of such a system are mentioned to be streaming capability, structured input description, on-demand audio/video generation, and flexible open architecture. A component-based architecture is designed and implemented, which uses standard interfaces to interact internally and also with other objects provided by underlying platform.

An XML-based Face Modeling Language (FML) is designed to describe the desired sequence of actions in form of a scenario. FML allows event handling, and also sequential or simultaneous combination of supported face states, and will be parsed to a set of MPEG-4 compatible face actions. Future extensions to FML can include more complicated behavior modeling and better coupling with MPEG-4 streams.

The image-based transformations used in the video kernel are shown to be successful in creating a variety of facial states based on a minimum (sometimes only one) input images. They can be extended to include facial textures for better results, and the system allows even a complete change of image generation methods (e.g. using a 3D model), as long as the interfaces are supported.

Better feature detection is a main objective of our future work, since any error in detecting a feature point can directly result in a wrong transformation vector. This effect can be seen in cases like eyebrows where detecting the exact corresponding points between a pair of learning images is not easy. As a result, the learned transformation may include additive random errors which causes non-smooth eyebrow lines in transformed feature set and image.

Combination of pre-learned transformations is used to create more complicated facial states. As discussed, due to perspective nature of head movements, this may not be a linear combination. Methods for shrinking/stretching the mapping vectors as a function of 3D head rotation are being studied and tested. Another approach can be defining

the mapping vectors in term of relative position to other points rather than numeric values. These relational descriptions may be invariant with respect to rotations.

References

[1] A. Arya and B. Hamidzadeh, "TalkingFace: Using Facial Feature Detection and Image Transformations for Visual Speech," *Proc Int Conf Image Processing (ICIP)*, 2001.

[2] S. Battista, et al, "MPEG-4: A Multimedia Standard for the Third Millennium," *IEEE Multimedia*, October 1999.

[3] V. Blanz and T. Vetter, "A Morphable Model For The Synthesis Of 3D Faces," *Proc ACM SIGGRAPH*, 1999.

[4] C. Bonamico, et al, "A Java-based MPEG-4 Facial Animation Player," *Proc Int Conf Augmented Virtual Reality& 3D Imaging*, 2001.

[5] C. Bregler, et al, "Video Rewrite," *ACM Computer Graphics*, 1997

[6] D. Bulterman, "SMIL-2," *IEEE Multimedia*, October 2001.

[7] J. Cassell, et al, "BEAT: the Behavior Expression Animation Toolkit," *Proc ACM SIGGRAPH*, 2001.

[8] T. Dutoit, *An Introduction to TTS Synthesis*, Kluwer Academic Publishers, 1994.

[9] P. Ekman and W. V. Friesen, *Facial Action Coding System*, Consulting Psychologists Press Inc., 1978.

[10] T. Ezzat and T. Poggio, "MikeTalk: A Talking Facial Display Based on Morphing Visemes," *Proc IEEE Conf Computer Animation*, 1998.

[11] J. Funge, et al, "Cognitive Modeling: Knowledge, Reasoning, and Planning for Intelligent Characters," *Proc ACM SIGGRAPH*, 1999.

[12] H. P. Graf, et al, "Face Analysis for the Synthesis of Photo-Realistic Talking Heads," *Proc IEEE Conf Automatic Face and Gesture Recognition*, 2000.

[13] N. Hirzalla, et al, "A Temporal Model for Interactive Multimedia Scenarios," *IEEE Multimedia*, Fall 1995.

[14] M. Kallmann, and D. Thalmann, "A Behavioral Interface to Simulate Agent-Object Interactions in Real Time," *Proc IEEE Conf Computer Animation*, 1999.

[15] G. Lawton,"Video Streaming," *IEEE Computer*, July 2000.

[16] W. S. Lee, et al, "MPEG-4 Compatible Faces from Orthogonal Photos," *Proc IEEE Conf Computer Animation*, 1999.

[17] I. S. Pandzic, "A Web-based MPEG-4 Facial Animation System," *Proc Int Conf Augmented Virtual Reality & 3D Imaging*, 2001.

[18] F. Pighin, et al, "Synthesizing Realistic Facial Expressions from Photographs," *Proc ACM SIGGRAPH*, 1998.

[19] B. Tiddeman, et al, "Prototyping and Transforming Facial Textures for Perception Research," *IEEE CG&A*, September 2001.