

bradscholars

On the development of an Interactive talking head system based on the use of PDE-based parametric surfaces

Item Type	Article
Authors	Athanasopoulos, Michael;Ugail, Hassan;Gonzalez Castro, Gabriela
Citation	Athanasopoulos, M., Ugail H. and Gonzalez Castro, G. (2011). On the development of an Interactive talking head system based on the use of PDE-based parametric surfaces. In: Transactions on Computational Science XII. Lecture Notes in Computer Science, Vol. 6670. A. Sourin and O. Sourina (Eds.).
DOI	https://doi.org/10.1007/978-3-642-22336-5
Rights	(c) 2011 Springer Verlag. Reproduced in accordance with the publisher's self-archiving policy.
Download date	2026-03-15 01:22:59
Link to Item	http://hdl.handle.net/10454/4977

On the development of a talking head system based on the use of PDE-based parametric surfaces

Michael Athanasopoulos, Hassan Ugail, and Gabriela González Castro

Centre for Visual Computing, School of Computing, Informatics and Media,
Bradford BD7 1DP, United Kingdom
mathana1, h.ugail, g.gonzalezcastro1@bradford.ac.uk

Abstract. In this work we propose a talking head system for animating facial expressions using a template face generated from Partial Differential Equations (PDEs). It uses a set of preconfigured curves to calculate an internal template surface face. This surface is then used to associate various facial features with a given 3D face object. Motion retargeting is then used to transfer the deformations in these areas from the template to the target object. The procedure is continued until all the expressions in the database are calculated and transferred to the target 3D human face object. Additionally the system interacts with the user using an artificial intelligence (AI) chatterbot to generate response from a given text. Speech and facial animation are synchronized using the Microsoft Speech API, where the response from the AI bot is converted to speech.

Keywords: Facial animation, Speech animation, Motion re-targetting, PDE method, Parametric surface representation and Virtual interactive environments.

1 Introduction

A talking head system is an environment where a 3D human head is talking and interacting with a user. In such system, it is required to provide all the necessary tools for creating a computer-human interaction process. To that extend, a text-to-speech (TTS) system is required to produce a sequence of phonemes from an input text. A phoneme is the basic unit of acoustic speech. A visual representation of the phoneme is called viseme. In speech animation visemes are often used to represent the position of the lips, jaws and tongue in a given particular phoneme. Many phoneme sounds are visually ambiguous when pronounced. Therefore, one viseme can be used to represent several phonemes. This technique is very popular for generating realistic speech animation without having to manually set the key frame positions for a set of visemes [1]. The conventional lip synchronization technique consists initially of decomposing the speech into a set of phonemes. These phonemes will be visually represented in the system as a set of visemes. A mapping between the phonemes in the speech signal and the visemes in the database is carried out to construct the appropriate lip shape.

In virtual worlds or environments such as cyberworlds, it is necessary to keep the data transmission size as small as possible for real time performance. Facial animation usually requires a large set of expressions to produce any given text. The facial data will need to be as small as possible to generate the speech animation without any lag or de-phase over the network. Additionally, the system will require a real time response to a users input. Real time dialog systems can include personality, emotions and interactive dialog in a computer-human or computer-computer situation. The process of human-computer interaction is facilitated with the use of chatterbots where actions such as personality, emotions and respond can be integrated within the talking head system [2]. Moreover, emotional tags embedded in the dialogue database can be used to generate facial expressions.

A new technique is introduced here that reuses facial animation to different 3D human face target models. The system uses a set of pre-configured viseme poses to generate a template human face model. Each new viseme pose will be then re-targetted to a given 3D target model for speech animation. The process can be repeated until all the required visemes are calculated. The 3D human head is synchronized with a Text To Speech (TTS) engine to generate voice, whereas a text-to-visemes function will return the current visemes of a given text in real time. Moreover, the system integrates an AI bot engine to calculate response from user input text. The engine used in this work is the Rebecca (Artificial IntelligenceMarkup Language) AIML library that implements the Alice bot language processing chatterbox. Chatterbots are computer systems that can produce a human-computer interaction in nature language. The user can enter a question or phrase and the AI bot will generate the appropriate answer to facilitate a real time conversation. Text response from the bot is then captured by the TTS engine and converted to a set of visemes to synchronize and animate the 3D human face. Animation is carried out by linearly interpolating any given set of visemes to generate the in-between transition of different visemes.

1.1 Related work

The problem that arises in facial animation consists of portraying the realism of the movement. The natural contact with facial expressions and the availability of better and more powerful hardware demand an ongoing improvement of the animation techniques for facial animation. A number of works in the area previously have introduced new techniques for achieving realistic motion. These include:

A Talking Head System for Korean Tex animates the face of a speaking 3D avatar in such a way that it realistically pronounces the given Korean text [3]. The proposed system consists of SAPI compliant text-to-speech engine and MPEG-4 compliant face animation generator. The input to the engine is a Unicode text that is to be spoken with synchronized lip shape. The TTS engine generates a phoneme sequence with their duration and audio data. The TTS applies the co-articulation rules to the phoneme sequence and sends a mouth animation sequence to the face modeler.

Greta: A Simple Facial Animation Engine is a 3D facial animation engine compliant with MPEG-4 specifications [4]; the aim in this work was to simulate in a rapid and believable manner the dynamics aspect of the human face. Greta, a 3D proprietary facial model with the look of a young woman, is the core of an MPEG-4 decoder and is compliant with the Simple Facial Animation Object Profile” standard. The 3D model uses a pseudo-muscular approach to emulate the behaviour of face tissues and also includes particular features such as wrinkles and furrow to enhance its realism. Facial features such as wrinkles have been implemented using bump mapping which that allows a high quality 3D facial model with a relative small polygonal complexity.

Real-time Lip Synchronization Based on Hidden Markov Model; A lip synchronization method that enables re-using of training videos when input voice is similar to training voice sequences [5]. The face sequences are clustered from video segments, then by making use of sub-sequence Hidden Markov Models, the system builds a correlation between speech signals and face shape sequences. This decreases the discontinuity between two consecutive output faces and obtains accurate and realistic synthesized animations. The system can synthesize faces from input audio in real-time without noticeable delay. Since acoustic feature data calculated from audio is directly used to drive the system without considering its phonemic representation, the method can adapt to any kind of voice, language or sound. Movement Realism in Computer Facial Animation is another work targeting realism in movement and behaviour of agents or avatars in virtual environments [6]. The system is using co articulation rules for visual speech and facial tissue deformation producing expressions and wrinkles. The aim of the work presented here consists of generating a template talking head system that will retarget realistic facial animation to a given human face model. It uses a pre-configured database of visemes curves that are used to compute the PDE method for generating the template surfaces. Each generated viseme surface will be then transferred to a different 3D human face object [7,8]. Every template viseme surface is grouped into 4 facial areas; each of these areas is associated with the target model to transfer the facial features for the current viseme. This is a pre-processed procedure that takes place at the loading time of the application. The use of the PDE method for surface generation and representation provides several advantages for the manipulation of that surface. For instance, most of the information required to define a surface is contained at the boundary or the outline curves representing that object. The representation of the template human face expressions is controlled through the adjustments of parameters associated with these curves. Thus, the user can interactively transform the parameters and change the shape of the surface without having any knowledge of the mathematical theory behind the PDE method [9,10].

2 Implementing a talking head system

In a Data-driven facial animation [11] system, the expressions are usually pre-configured and blended together in order to produce a sequence of letters; whereas

in a 3D human head, the animation is synchronized with a text-to-speech engine to generate speech according to a set of phonemes for a given word. The system is usually interactive and can incorporate changes of emotions dependant to the users input. Thus, it is necessary to build a viseme-driven talking head system where a given human head mesh model with similar topology, can be animated without the need to generate all the necessary expressions for the operation. The overall process consists of generating a set of PDE-based expressions that are used internally as the template expressions for any given human head model. The PDE method has been used for representing the surface template expressions; thus utilizing the advantages of parametric surfaces. The PDE method provides us with tools to control the surface resolution or smoothness of the facial geometry by adjusting the u, v parameters. At the same time, the PDE method enables us to generate facial animation from a given complex face model by adjusting only a small set of boundary curves. This methodology enable us to produce a set of template mesh surfaces that are used to transfer a given motion sequence to a given human head mesh model. This section will explain in more details the implementation of the talking head environment.

The PDE method produces a parametric surface, which in summary is the graphic representation of the solution to an elliptic PDE of the form

$$\left(\frac{\partial^2}{\partial u^2} + a \frac{\partial^2}{\partial v^2} \right)^n \chi(u, v) = 0, \quad (1)$$

where a represents an intrinsic parameter and n determines the order of the PDE. This work focuses on the use of the biharmonic equation; that is the values of a and n had been set equal to 1.0 and 2 on each respective case. Moreover, this work restricts the use of the PDE method to periodic boundary conditions where by $0 \leq u \leq 1.0$ and $0 \leq v \leq 2\pi$. Thus an approximate analytic solution to Equation 1 is found and it is given by

$$\chi(u, v) = \mathbf{A}_0(u) + \sum_{n=1}^{n=N} [\mathbf{A}_n(u) \cos(nv) + \mathbf{B}_n(u) \sin(nv)] + \mathbf{R}_n(u, v), \quad (2)$$

where $\mathbf{A}_0(u)$ is a cubic polynomial, and, $\mathbf{A}_n(u)$ and $\mathbf{B}_n(u)$ are functions resulting from linear combination of exponential functions and powers u . The term $\mathbf{R}_n(u, v)$ is known as the remainder term and it is responsible for exactly satisfying the original boundary conditions imposed to Equation 1. Further details on the mathematical foundations of the the PDE method, the reader is referred to [10] where additional details are given.

2.1 Generating expressions

Viseme-driven speech animation approaches [11] often require manual design of key mouth poses in order to generate realistic speech animations. The first step for building the facial animation system consists of generating the facial expression data [12]. The template face, Figure 1.a, is represented as a set of

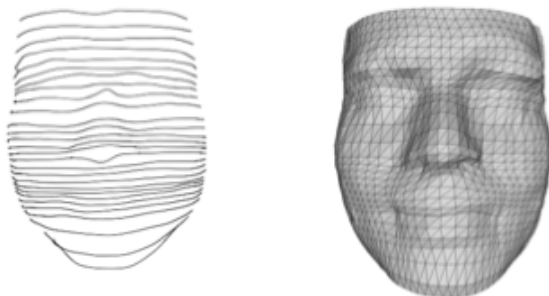


Fig. 1. The neutral expression template curve set (left). The resulting surface (right).

28 boundary curves acquired from a laser-scanned 3D face model in its neutral configuration.

The curve-set covers the entire face area, describing the most basic facial features. The template face surface is then reconstructed using a combination of nine different fourth orders PDEs that guarantee surface continuity. Note that the u, v resolution of the PDE surface is set to 35×35 . As mentioned before, a set of preconfigured expressions is required for animating speech cite13. A database of 22 visemes is used to identify the corresponding viseme of a given word or phrase within the TTS engine as shown in Figure 3. The talking head database consists of 15 viseme expressions; some viseme can be repeated to save loading time e.g. viseme AW and OY can be represented using the same viseme.

Each viseme is represented as a curve-set derived from a face mesh model as seen in Figure 1.b. The viseme curve-set poses have been obtained from the work presented in [14]. The authors have developed a PDE-based facial deformation technique that uses the boundary curves of a PDE surface to deform a face model using the MPEG-4 compliant facial feature points. The idea behind this approach is that it utilizes the boundary curves for complex facial deformations rather than using conventional control points and surface interpolation techniques. A group of boundary points anatomically related to facial features, such as right corner of left eyebrow, left corner of inner lip contour, are selected as feature points according to the MPEG-4 definition. Animation of the face model is achieved by adjusting the position of the boundary curves before each calculation of the PDE surface. The deformations have been computed using a series of weighted sinusoids to parameterize the FAP-driven facial animation with the feature points. Additional boundary points have also been selected and used as weights in areas around the feature points in order to guarantee surface smoothness. The mouth and eyes area is achieved using sinusoidal animation of the corresponding feature points. For stretching the lip corners, linear interpolation is applied to the boundary curve points representing the lips.

The resulted curve-sets are stored in an internal viseme database and they will be used for computing a PDE surface that passes though the control points of each curve to generate a surface representation. The process can be seen in

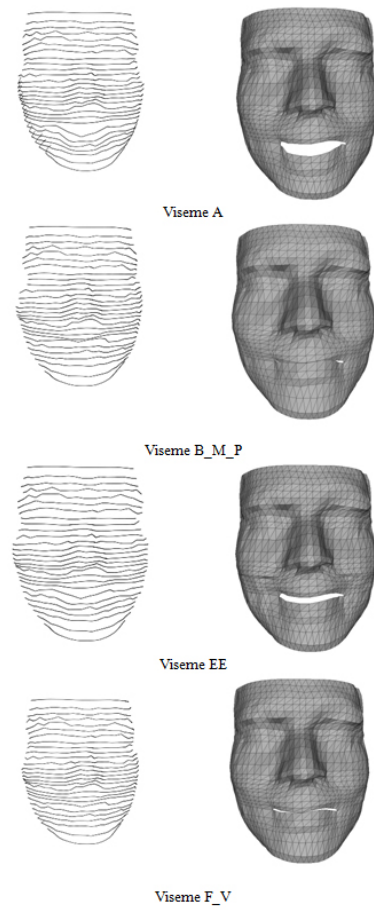


Fig. 2. Curved-based visemes (left). Template surfaces for each corresponding curve set (right).

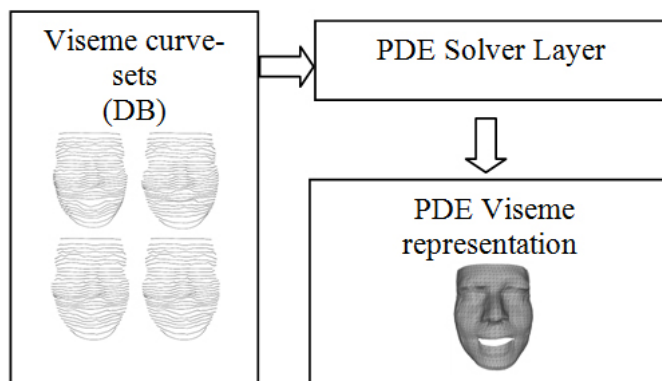


Fig. 3. PDE-based viseme process.

Figure 3. Each curve-set viseme is used as the boundary conditions required for calculating the PDE method. The new surface is stored and used later for re-targeting the facial deformations of the corresponding viseme to a different face mesh model. The preconfigured viseme curve-sets contain all the necessary information to generate the PDE-based speech expressions for the animation system. Storing only a set of curves rather than a mesh object for each required viseme pose gives us the advantage in keeping the storage requirements minimum. This technology can be exploited to reduce network transmission bit-rates, by sending only animation parameters rather than the video sequence. Figure 2 contains various template viseme poses for calculating the required template expressions. The curve sets seen in Figure 2 (right), are used to generate the human face mesh for the given viseme and apply motion re-targeting to transfer the deformations to a different target human face model. Visemes A, B/M/P, EE and F/V used for the speech animation are shown in both configurations, curve and mesh representation.

3 Text-to-speech engine integration

Various important developments in speech synthesis and natural language processing techniques resulted in the concept of text-to-speech synthesis. A text-to-speech synthesis can be defined as automatic production of speech, through a grapheme-to-phoneme transcription of the sentences to utter [15]. Figure 4 shows the general functional diagram of a synthesizer. It consists of a Natural Language Processing module (NLP), capable of producing a phonetic conversion of the text read together with the desired voice tone and rhythm (also called prosody), and a Digital Signal Processing module (DSP), which transforms the symbolic information it receives into speech. Phonetic and prosody information

together can generate the symbolic linguistic representation that is output to the front-end application. The back-end application usually provided from the API, converts the symbolic linguistic representation into sound usually referred as the synthesizer. Details about the architecture of a TTS system is outside the scope of this paper, additional information can be found in [15].

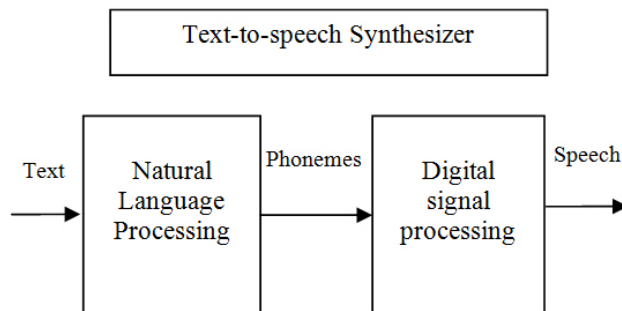


Fig. 4. General function diagram of the TTS system used in this work.

The text-to-speech system converts normal language text into speech; it recognizes the input text and using a synthesized voice, chosen from several pre-generated voices, speaks the written text. The TTS engine used for this work is the Microsoft TTS which is fully programmable from the Speech API 5.0. The SAPI acts as a software layer that allows speech-enabled applications to communicate with both speech recognition and TTS engines.

Additionally, the SAPI is responsible for a number of functions in a speech system, such as:

- Controlling audio input, whether from a microphone, files, or custom audio source.
- Converting audio data.
- Storing audio and serializing results for later analysis.
- Using SAPI grammar interfaces and loading dictation.
- Performing speech recognition.

The speech API provides a high-level interface between an application and the speech engine. All the low-level details needed to control the real-time operations of various speech engines are implemented in a collection of libraries and classes. With the help of the API, the TTS operation, which is required to convert a given text into a set of viseme, can be performed real time. The text-to-viseme process is used to identify which letter from the database is required to animate a given text and synchronize it with the voice. Using events,

the application synchronizes the output speech with real time actions such as the phonemes or visemes generated from the input text.

In this case, the generated viseme is used to query from the database the current viseme index that is required for the speech animation. Table 3 contains the first 10 indexes used to identify which visemes are needed to simulate the speech animation. The TTS engine contains 22 visemes in total used for reproducing the sounds from any given input text. Note that many phoneme sounds are visually ambiguous when pronounced. Therefore, one viseme can be used to represent several phonemes. For example letters F and V can be reproduced by using the same viseme. Consonant letters such as B, M and P can be represented as well with the same viseme. To that extend, the animation system can link various similar visemes together to minimize loading and calculation processing time.

SP_VISEME_1	ae, ax, a
SP_VISEME_2	aa
SP_VISEME_3	ao
SP_VISEME_4	ey, eh, uh
SP_VISEME_5	er
SP_VISEME_6	y
SP_VISEME_7	w, uw
SP_VISEME_8	ow
SP_VISEME_9	aw
SP_VISEME_10	oy

Table 1. List containing the the first 10 viseme indexes association.

A total of 15 visemes are used in the talking head database to simulate speech. The remaining 7 visemes from the TTS engine have been associated with other similar visemes. For example, SP_VISEME_1, 2 and 3 from Table 3 can be represented as SP_VISEME_1.

The integration of the TTS system in the talking head environment was developed using the .NET framework. Using the Component Object Model (COM) ISpVoice interface applications can control the TTS functionality. Initialization of the engine is achieved simply by creating a spVoice object, whereas for text to speech synthesis a single call to ISpVoice::Speak is required. Additional functionality for manipulation of voice and synthesis is provided; various function calls can control the speaking rate, the output speech volume and the current speaking voice.

The speak method can be operated synchronously or asynchronously, an important feature for synchronizing the speech output with the rendering API as well as adjusting speech properties in real time. The SAPI communicates with the applications by sending events using standard windows callback mechanisms. Applications can then sync to real-time actions such as word boundaries,

phoneme or viseme boundaries. Events are used for synchronizing the output speech. Each audio stream generated from the SAPI engine contains an event id where the application can identify position and status of the stream. A viseme event id is associated with the list in Table 3 to find the current viseme. This information will help calculate the interpolation time between previous and current viseme. Additionally, the TTS engine will playback the audio stream from the input text.

Communication between the application and the SAPI is processed as a two step operation. The application first receives a general window message from the SAPI. This message is similar to other window messages used by the operating system, such as mouse, keyboard, and window events. The second step in the communication process is determining which action occurred. The application needs to determine the exact action that is taking place. A list of all the SAPI defined action can be found using the **SPEVENTENUM** list. Using **SPEVENT** and **GetEvents** method the SAPI can identify specific information about the current event. Actions such as **START_INPUT_STREAM**, **END_INPUT_STREAM** and **WISEME** are used to determine the current activity that is taking place. Once the current event is determined, the application needs to take the necessary action. When a **START_INPUT_STREAM** event is capture, the input stream from a **Speak** call has begun synthesizing to the output, whereas an **END_INPUT_STREAM** is identified as the end of the **Speak** event. In the case of the **WISEME** event id, the SAPI returns the relevant viseme with some additional information for the current Speech operation.

4 A.L.I.C.E's Brain

4.1 Introduction

Once the speech engine is configured to capture and process real time events, the following step consists integrating an Artificial IntelligenceMarkup Language or AIML chatterbot to generate response from a given text. A.L.I.C.E. (Artificial Linguistic Internet Computer Entity), also referred to as Alicebot, or Alice, is a language processing chatterbox based on an experiment specified by Alan M. Turing in 1950. A chatterbox is a program that engages in a conversation with a user by applying some heuristical pattern matching rules to the human's input [11, 16], A.L.I.C.E. software utilizes AIML, an XML based language used for creating chat robots.

It contains a class of data objects defined in the XML specification [17] called AIML objects and describes the behaviour of computer programs that process them. Various Alicebot clones have been created based upon the original implementation of the program and its AIML knowledge base [18]. Some of these AIML interpreters are:

- RebeccaAIML(C++, Java, .NET/C#, Python)
- Program D(Java)
- Program R(Ruby)

– Program O(PHP)

This work uses the Rebecca AIML library for generating real-time responses. The system establishes a local connection with the Rebecca AIML bot to process and generate response to a given input text. The response is then handled as an input text for the SAPI text-to-speech engine.

4.2 Previous work

Persona-AIML [19] presents the Persona-AIML architecture for the creation of chatterbots in AIML (Artificial Intelligence Markup Language) with personality. Computational models of personality are in general adapted from some psychology model or theory. The Personality Component defines the beliefs, the personality elements, and the rules that determine chatterbots behaviour.

The work presented in [20] demonstrates an emotional MPEG-4 compliant talking head system based on AIML. It uses Alicebot to generate response and emotion from given input text. Emotions are embedded in the AIML database as a set of predefined emotion tags. These emotional tags are passed to the personality model to simulate believable behaviours. The personality model, depending upon the current mood and the input emotional tags, updates the mood. Depending upon the output of the personality model, mood processing is done to determine the next emotional state. This processing determines the probabilities of the possible emotional states. Additionally, the system generates lip sync from the visemes generated from the Text-To-Speech engine.

TQ-Bot in [21], presents an Intelligent Tutoring System using AIML whose aim is in providing personalized instruction to students. The authors have developed an open e-Learning platform for helping the students during their learning process and for supporting the activities of the teacher. The bot is able to analyze the requests made by the learners in written natural language and to provide adequate and domain specific answers orienting the student to the right course contents. Additionally, TQ-Bot is able to track and supervise the student progress by means of personalized questionnaires.

The brain of A.L.I.C.E. consists of 41,000 elements called categories [2]. Each category contains a question and answer, called the "pattern" and "template". The patterns are stored in a tree structure managed by an object called the Graphmaster, which implements a matching algorithm. Graphmaster matching is a special case of backtracking, depth-first search. In most cases matching is handled by a linear traversal of the graph from the root to a terminal node.

The AIML architecture allows the use of different models of personality in the construction of chatterbots. It implements various tags to introduce randomness in answers, and to keep track of small dialogue history. Although it does not use any syntactic or semantic language analysis techniques to generate the response, the content embedded in AIML is enough to engage the user in believable conversation to a certain degree. A.L.I.C.E contains a learning mode, called supervised learning since a botmaster is required to create and manage the content. The botmaster will monitor the bot conversations and can create new

AIML content to make the bot responses more believable, accurate or human like.

Moreover, every AIML object has both a logical and a physical structure. The physical structure consists of units called topics and categories, while the logical structure is composed of elements and character references.

4.3 AIML elements

The basic unit of knowledge in AIML is called a category [2]. The term category was borrowed from pattern recognition theory. Each category consists of an input question, an output answer, and an optional context. The question is called pattern, the response is called template and lastly, the optional context is divided into two types called that and topic. The AIML pattern tag consists only of words, spaces, and the wildcard symbols. Words must contain only letters and numbers separated by a single space whereas the wildcard characters can function like words. A pattern element must appear in each category and it must always be the first child element of that category element. A pattern does not have any attributes.

Additionally, AIML supports interaction with other languages and systems. For example the `<system>` tag can be used to execute any program or command accessible from the operating system and insert the result in the reply. Alternatively, the `<javascript>` tag can be used to allow scripting inside the templates. The `<template>` tag it is the most basic AIML tag and is always paired with a `<pattern>` tag. It always appears within `<category>` elements and it doesn't contain any attributes. The `<template>` must follow the `<that>` element or follow the `<pattern>` element. The majority of AIML content is within the template. The template may contain zero or more AIML template elements mixed with character data.

Figure 5.b shows an AIML example code, the pattern defines the input and the template tag defines the bots response to that input. The syntax of an AIML category is:

The above AIML code matches the client text input, in this case the word "Hello" and sends back to the client the response "Hi! How are you?".

The optional context in the category tag consists of two elements, called `<that>` and `<topic>`. The `<that>` element appears inside the category, and its pattern must match the bots last response. The `<that>` element is a special type of pattern element used for context matching. It is optional in a category, but if it exists it must occur no more than once, and must follow the `<pattern>` and `<template>` element. Remembering the last response is important for creating a more believable conversation. In the example below, Figure 6.a, the `<category>` element is activated when the client says yes. The bot must find out what was the question to that answer. If the bot asked, "Do you like AIML?", the category matches the `<that>` element and it continues the conversation using the `<template>` response element.

The `<topic>` is an optional element that might appear outside a `<category>` element, and it is used to group together categories. The `<topic>` element al-

<pre> <aim!> <category> <pattern>InputText pattern> <template>Response template> </category> </aim!> (a) </pre>	<pre> <aim!> <category> </ <pattern> Hello </pattern> <template> Hi! How are </you? </template> </category> </aim!> (b) </pre>
---	--

Fig. 5. General syntax of an AiML category tag (a). Simple example of input and output response (b).

allows the bot to store duplicate patterns in different topics, this way the bot can generate different responses to the same input patterns depending on the topic. A `<topic>` element has a `requirednameattribute` that must contain a simple pattern expression. A `<topic>` element may contain one or more `category` elements. The botmaster uses the `<set_topic>` tags to set the topic of current `<category>` element. Once the topic is set, for any new query from the client the bot will start looking for a response in the categories that match the current `<topic>` tag. If there is not a category defined in the current topic, then any categories that are not defined in topic tags are searched.

Figure 6.b presents an example of using the `<that>` element. In this case, if the client says something that the bot does not have a specific response for, it could still respond within the current topic. For example, the response for any undefined pattern element under the AiML topic will be "My favourite interpreter is RebeccaAiML".

AiML consists of various elements offering learning capabilities and intelligent response to achieve realistic human computer interaction. Moreover, the `<random>` element tag can generate different responses to the same input text. Each possible template response for the current pattern element needs to be separated with the `` tag element.

AiML is extensible; the botmaster can include an infinite number of new tags for application specific properties. Predicate tags can be used according to a client based "set" and "get" method to generate an endless variety of responses.

Recursive categories can be used to map one input to other inputs, either for language simplification or to identify similar patterns. The AiML implementation for recursion is the tag `<srai>`. Figure 7.a presents a basic recursion example, if the user says "HI", "Hello", "Hi there", etc., the response template will be the same as for the "Hi" pattern element.

Recursions are useful for a variety of tasks, some of these include:

- Symbolic Reduction: Reduction and simplification of complex input patterns.

<pre> <category> <pattern>YES</pattern> <that>DO YOU LIKE AIML?</that> <template>WHAT IS YOUR FAVORITE INTERPRETER? </template> </category> (a) </pre>	<pre> <topic name="AIML"> <category> <pattern>*</pattern> <template>MY FAVORITE INTERPRETER IS RebeccaAIML </template> </category> (b) </pre>
--	---

Fig. 6. <That> element example (a). <Topic> element example (b).

- Divide and Conquer: Split an input into two or more parts, and combine the responses to each.
- Synonyms: Generate different ways of saying the same thing to the same reply.
- Spelling or grammar corrections.
- Detecting keywords anywhere in the input.
- Conditionals: Branching can be implemented with the <srai> operator.

A common application for recursive categories is simplification and reduction of complex input patterns. A combination of the <star> tag and <srai> recursive calls can be used to produce endless combinations. For instance, Figure 7.b shows an example of recursion using the <star> tag. The bot will match a pattern starting with "What is" and use the "*" value to define a recursion call to find the best match for the transformed input.

<pre> <category> <pattern>HI THERE! </pattern> <template> <srai>HI</srai> </template> </category> (a) </pre>	<pre> <category> <pattern>WHAT IS * </ pattern> <template> <srai>WHAT IS<star/> </srai> </template> </category> (b) </pre>
--	--

Fig. 7. Basic recursion example (a). Simplification of input pattern using recursion and the <star> element (b).

4.4 Emotions

One of the many advantages of using AIML is that it is fully customizable. The bot master can include new AIML content with custom handling of input text. As discussed in the previous paragraph, implementing custom AIML elements can add intelligence to the bot and make the human-computer interaction more believable, accurate and human like. Additionally, an infinite number of new tags can be added to extend the functionality of the bot. The selection of current mood in the talking head system is an example of AIML customization. The emotions have been embedded in the AIML files by introducing a new tag called `<emotion>` and a "name" property.

The property name will hold the mood name of the current AIML pattern element. This information has been added manually in a Custom AIML file to support a range of different input pattern elements. Figure 8 shows an example of emotion handling in AIML. When the input text matches the pattern element, the template element contains additional information. In this example, the response is produced by a `<random>` tag that contains two different responses and the emotion that is set for current template is happy. However, the new `<emotion>` tag requires additional processing since the AIMLBot interpreter does not recognize custom tags. A simple XML node processing function searches the `<category>` node element for each new response to check if it contains a valid emotion element. If an emotion element is found, it is parsed to the Blend Shape layer to produce the new expression. Alternatively, if the current template does not contain any emotion element, then the current mood is set to neutral.

```

<pattern>I LIKE YOU</pattern>
<template>
  <emotion name="happy" />
  <random>
    <li> I Like you too.</li>
    <li> You are very kind. Thank you!</li>
  </random>
</template>

```

Fig. 8. An AIML example containing emotion data.

5 Blend shape visemes

Once the curve-sets have used to computed the PDE method, the generated PDE surfaces for each viseme in the database are stored and represented in the system as a group of polygon meshes. This is a pre-processed procedure that takes place

at the loading time of the application and it is repeated for all the required visemes in the database. The next step consists of generating animation for any two given facial expressions. The talking head system must be able to blend any required viseme poses in real time. For this task shape interpolation between two key-frames is used, where two or more facial expressions are captured and in between frames are computed by linear interpolation.

Linear interpolation is used to approximate a value using two known visemes; these indexes are computed from the text-to-speech engine and assigned a value according to the viseme mapping in the database. The interpolation is calculated for all the vertices in the coordinates of each viseme. Additionally, the normals of the new surface are calculated using the same technique.

The talking head system presented in this work integrates emotions that are produced from the AIML chatterbox engine according to the current user input. This requires the shape interpolation of an additional expression of the overall speech animation. In this case, the additional expression will be included in the interpolation function with an additional variable to control the amount or intensity of the blend shape. The intensity variable is passed to the interpolation in real time during the calculation to produce a realistic conversation during the speech animation.

The process of speech animation is explained in the diagram shown in Figure 9, which shows the communication between the various layers of the speech animation. Animating two viseme poses is dependant to the user input. Input text is send to the chatterbot process to generate a response; response from the AIML engine is then processed by the TTS engine to produce the appropriate speech and viseme information used for the animation. The corresponding visemes will be parsed to the Blend Shape process to produce the interpolated shape at any given time. Additionally, a third expression is parsed to the shape interpolator indicating the mood change. This information is obtained from the AIML engine in real time.

6 Motion retargetting

Finally, once the system is initialized with the PDE-based viseme expressions and synchronized with the AIML engine, it is required to retarget [22] the deformations to a different target face mesh model. The process of transferring the animation from a set of PDE-based template representations to a given face mesh model is carried out as follows:

- Alignment between the neutral viseme surface and the target mesh model in the same initial position. Key features of the model have to be positioned so that they nearly overlap. This will ensure a correct correspondence between the two surfaces.
- Mapping correspondence between models. This process consists of associating each point of the mesh model with the nearest point of the template surface in ‘their initial configurations. This way, each point in the mesh

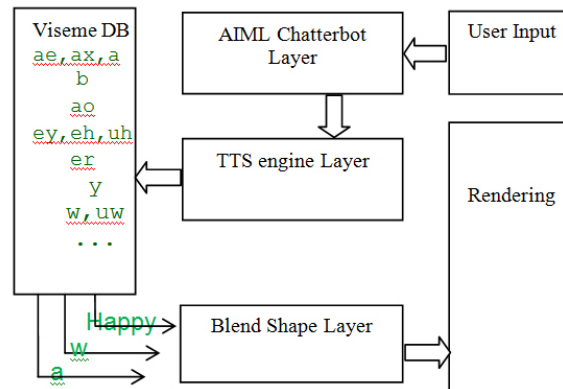


Figure 9. The speech animation process layers.

Fig. 9. The speech animation process layers.

model is represented on the template surface and it is assumed that this point remain as the closest one to the same particular point in every viseme surface in the database.

- Animation of the mesh model is carried out by finding the difference between the corresponding viseme pose and the original neutral surface for each point. This difference is then added to each point of the original mesh model according to the mapping correspondence previously found.

The above procedure is repeated for every expression in the viseme database. There are cases where a better mapping correspondence is required, e.g. mouth regions between the two objects, to achieve a more detailed representation. To solve the problem, the template face and the target object are split into face region maps, such as bottom and top mouth, nose and eyes. Each map contains the index of each vertex that is included in that region. The collection of these points is handled using the Autodesk Maya environment, where the selection and output of the correspondent vertices is performed using MEL scripts. The correct selection of these points is very important since the motion transfer is based on the vertex indexes each map contains.

Figure 10, contains the top and bottom mouth region maps for the template face and the target human face object. Motion retarget is applied separately to each region map until all the required facial areas are processed. Once the motion transfer is complete, the resulting object can be associated with the expression it represents in the viseme database. Note that the quality of the mapping correspondence between the template viseme and the output face model depends also on the resolution of the grid used to compute template surface representation. Thus, using a grid with a similar number of points to the number of vertices in the original mesh will produce a better mapping correspondence. As seen in the diagram in Figure 11, the motion retargeting process can be

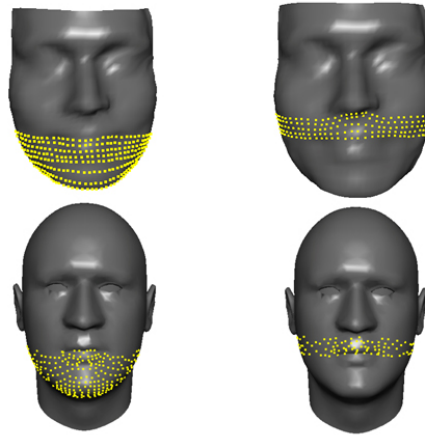


Fig. 10. Region maps used for motion re-target. PDE-based template (top). Mesh model target (bottom).

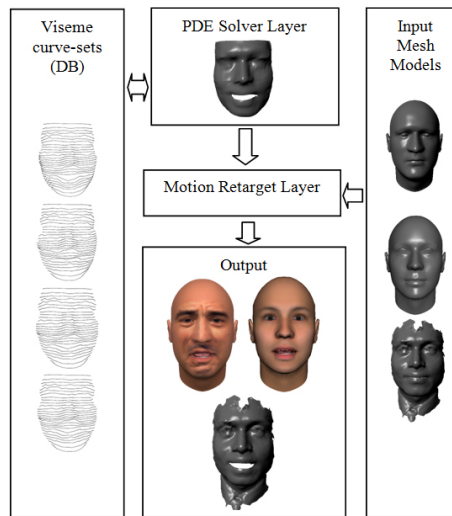


Fig. 11. Motion re-target process for each viseme in the database.

visualized as an additional layer in the talking head system that communicates between the PDE surface layer and the Input Mesh model.

7 Examples

Figure 12, contains a sequence of viseme expressions that are transferred from the template mesh Figure 12 (a, b, c) to the different target face mesh model Figure 12 (d, e, f). The motion retargeting technique employed in this work requires a mapping correspondence between the two objects, such as each point of the target mesh model is associated with the nearest point of the template surface. This way each point of the target mesh will be represented on the template model. Finally, motion retargeting is carried out by adding the difference between each point in the source model and the corresponding point in the target model.

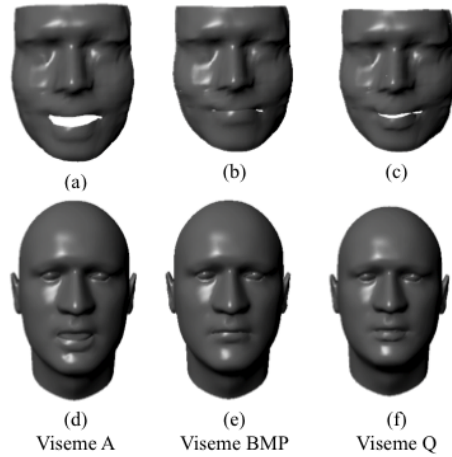


Fig. 12. Viseme templates. PDE-based visemes templates (a, b, c). Visemes retargetted to human head mesh model (d, e, f).

Next, sequence of expressions in Figure 13, shows several examples of motion retargeting on expressions that are used within the talking head system to simulate change of mood during the speech. These expressions are included in the blend shape process to adjust the current viseme according to the current mood expression. The current Mood selection can be controlled by certain input text, duration or from the AIML engine [19]. AIML elements can encapsulate the response and certain mood that is generated from each parsed input text. This way, the response from the bot can also contain the appropriate mood expression.

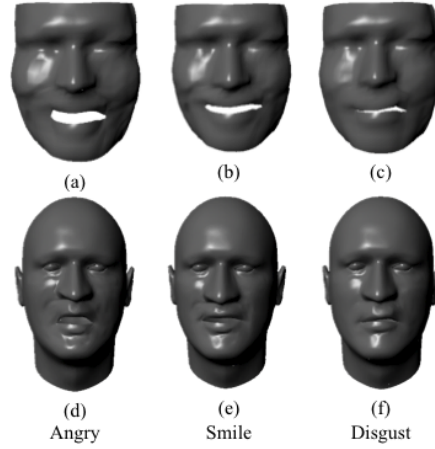


Fig. 13. Emotion templates. PDE-based emotion templates (a, b, c). Emotions re-targetted to human head mesh model (d, e, f).

Another example of motion re-targetting is shown in Figure ??; the target 3D human face has been replaced with a model acquired from a 3D scanner. The initial alignment of the two models plays a very important role in the correct motion transferring between the facial region maps. There are cases where facial regions of the target mesh need to be removed from the motion re-target process. The generic face template used in this work contains only 4 basic regions maps; for more realistic facial deformations the number of face area maps can be increased according to the facial features of the target objects.

The final textured version of the animated target object is shown in Figures 15 and 16. The animated face contains teeth, tongue and mouth to increase the realism of the speech animation. The position of the teeth and tongue is interpolated alongside with the viseme blend shape process.

8 Conclusions

This work presents a technique for animating facial expressions within a talking head system. It uses a set of preconfigured curves representing various viseme poses to calculate a template PDE-based surface. The resulting surface is used to associate various facial feature areas with a different target face mesh model. Motion re-target is then applied to transfer the deformations in these areas from the template to the target model. This technique offers animation re-usage, since all the necessary viseme poses for the animation are pre-calculated and re-used for a different target mesh model. Additionally, it minimizes the storage requirements by storing only a small set of curves for each expression. The system interacts with the user using an AIML chatterbot to generate response from input text.

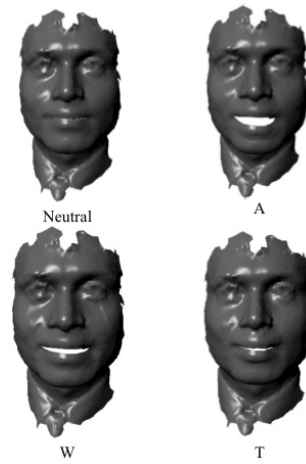


Fig. 14. Motion retarget to a 3D scan mesh model.

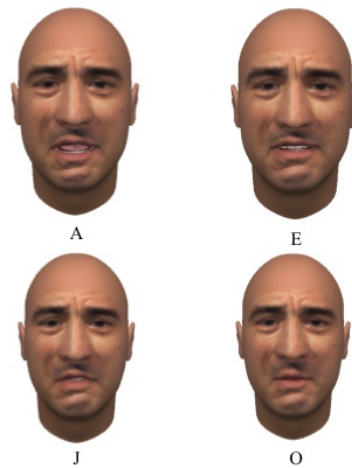


Fig. 15. Final rendering of the talking head system 3D face. The face model contains mouth, teeth and tongue animated according to each viseme.

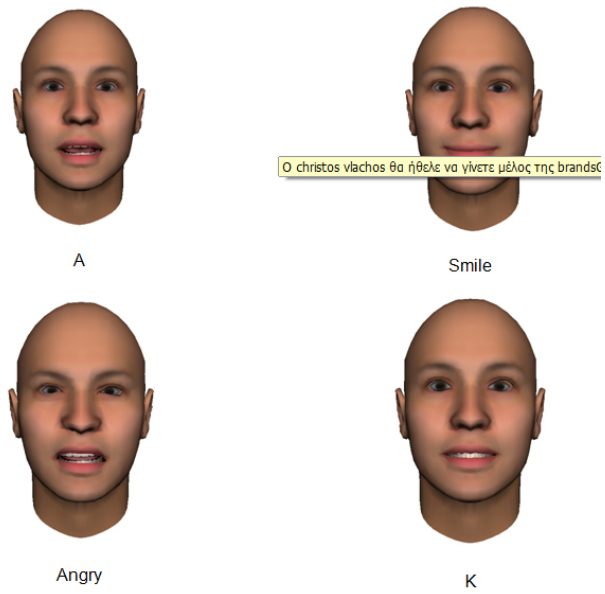


Fig. 16. Final rendering of the different face model retargeted to the talking head system. The face model contains mouth, teeth and tongue animated according to each viseme.

The user can enter a question or phrase and the AI bot will generate the appropriate answer to facilitate a real time conversation. The response is then captured and converted to speech from the text-to-speech engine; each word is split into a sequence of viseme poses that are used to synchronize the facial animation. The animation is carried out by linearly interpolating a given set of visemes to generate the in between transition of different visemes.

An improvement to be included in this technique is a more generic template model representation that could enable the animation to a larger variety of character models. Future work could be undertaken in automating the facial map extraction process. This process is required to associate various facial areas between the two objects for a seamless motion re-target.

References

1. Lee, R., Terzopoulos, D. and W. K.: Realistic Modeling for Facial Animation. Proceedings of the 22nd annual conference on Computer graphics and interactive technique, 55–62 (1995)
2. Wallace, S.R.: The Anatomy of A.L.I.C.E.
3. Kim, S. W. et al., A Talking Head System for Korean Text. World Academy of Science, Engineering and Technology, 50 (2005)
4. Pasquariello, R. and Pelachaud, C.: Greta: A Simple Facial Animation Engine. In Proceedings of the 6th Online World Conference on Soft Computing in Industrial Applications, (2001)
5. Huangm, Y., et al.: Real-time Lip Synchronization Based on Hidden Markov Models. The 5th Asian Conference on Computer Vision, Melbourne, Australia, (2002)
6. Maddock S. , Edge, J. and Sanchez, M. I.: Movement Realism in Computer Facial Animation. Workshop on Human-Animated Characters Interaction, 4 (2005)
7. Fedorov, A. et al.: Talking Head: Synthetic Video Facial Animation in MPEG-4. International Conference Graphicon Moscow, Russia (2003)
8. Balci, K.: Xface: MPEG4 Based Open Source Toolkit for 3D Facial. Proceedings of the working conference on Advanced visual interfaces, 399 - 402 (2004)
9. González Castro, G. et al.: A Survey of Partial Differential Equations in Geometric Design, The Visual Computer, 24(3), 213-225 (2008)
10. Ugail, H., Bloor, M.I.G. and Wilson, M.J.: Manipulation of PDE surfaces using an interactively defined parameterization, Computers and Graphics, 23(4), 525-534 (1999)
11. Deng, Z. and Noh, J.: Computer Facial Animation: A Survey in Data-driven 3D facial animation. Springer, (2007)
12. Marschner, S.R., Guenter, B. and Raghupathy, R.: Modeling and Rendering for Realistic Facial Animation, Proceedings of the Eurographics Workshop on Rendering Techniques, 231–242 (2000)
13. Haber, J. et al.: Face to Face: From Real Humans to Realistic Facial Animation. Proceedings Israel-Korea Binational Conference on Geometrical Modeling and Computer Graphics, 73–82 (2001)
14. Sheng, Y. et al.: PDE-Based Facial Animation: Making the Complex Simple Proceedings of the 4th International Symposium on Advances in Visual Computing, 723–732 (2008)
15. Dutoit, T.: High-quality text-to-speech synthesis, Springer (2001)

16. Galvo, A. M. et al.: Adding Personality to Chatterbots Using the Persona-AIML Architecture, in Book *Advances in Artificial Intelligence*, Springer Berlin / Heidelberg, 963–973 (2004)
17. Wallace, R.: Artificial Intelligence Markup Language (AIML) v1.0.1, A.L.I.C.E. AI Foundation Working Draft (2001)
18. Mana M. and Pianesi F.: HMM-based Synthesis of Emotional Facial Expressions during Speech in Synthetic Talking Heads. Proceedings of the 8th international conference on Multimodal interface, 380–387, (2006)
19. Galvo, A.M. et al.: Persona-AIML: An Architecture for Developing Chatterbots with Personality, Third International Joint Conference on Autonomous Agents and Multiagent Systems - New York, USA, 3 (2004)
20. Giacomo, T.D., Garchery, S. and Thalmann, N.M.: Expressive Visual Speech Generation in Data-driven 3D facial animation, Springer (2007)
21. Fonte, A.M.: TQ-Bot: An AIML-based Tutor and Evaluator Bot Fernando, *Journal of Universal Computer Science, Austria*. 15(7),
22. Pighin, F. et al.: Synthesizing Realistic Facial Expressions from Photographs. Proceedings of the 25th annual conference on Computer graphics and interactive techniques, 75–84 (1998)